Computers and Intractability: A Guide to Algorithmic Lower Bounds[*]

by

Erik D. Demaine,
William Gasarch, and
Mohammad Hajiaghayi

DRAFT from August 24, 2022

hardness-book@mit.edu

https://hardness.mit.edu

[*]Copyright in this work has been licensed exclusively to The MIT Press, https://mitpress.mit.edu, which will be releasing the final version to the public in 2024. All inquiries regarding rights should be addressed to the MIT Press, Rights and Permissions Department.

Contents

Preface

Some Problems Are Easy, Some Are Probably Hard

The following question permeates all of computer science and mathematics:

Given a problem, how hard is it to solve?

There are many aspects to hardness: time, space, communication, number-of-disk-calls, and others.

Why is it important to know that a problem is hard? If you know that obtaining (say) an exact solution to a problem is hard then you may look for an approximation. If you know that the general case is hard then you may want to look at special cases. There are many other scenarios.

Consider the following two problems:

1. Given a graph G is there an ***Eulerian Cycle***, which is a cycle that visits every *edge* exactly once? We call this problem Euler Cycle.

2. Given a graph G, is there a ***Hamiltonian Cycle***, which is a cycle that visits every *vertex* exactly once? We call this problem Ham Cycle.

For these two problems we will consider ***polynomial time*** (P) to be ***easy*** and NP-completeness to be ***hard***. We will define both of these terms in Chapter 0.

In 1736 Euler showed (in modern terminology) that a graph G has an Eulerian Cycle if and only if every vertex has even degree. Hence Euler Cycle is easy. Mathematicians later tried to find a characterization for Ham Cycle. Note that the problem of finding a "characterization" was not well defined. In 1970 Cook and Levin developed the theory of NP-completeness and in 1972 Karp showed that Ham Cycle is NP-complete, meaning that it is unlikely to be in P. Thus the theory of NP-completeness not only showed that it is unlikely Ham Cycle has a characterization, it also provided a way to state the questions rigorously.

P, NP, NP-Complete, Cook, Levin, Karp, Garey, Johnson, and Us

Cook [Coo71] and Levin [Lev73] (see [Tra84] for a translation of Levin's article into English and some historical context) independently proved that SAT (given a Boolean Formula, is there a satisfying assignment?) is NP-complete. Shortly thereafter Karp [Kar72] showed that 21 more natural problems are NP-complete problems. Ham Cycle was one of them. Since then literally thousands of problems have been proven NP-complete.

The initial papers used different definitions and notation. Garey & Johnson [GJ79] wrote a book that unified the definitions and notation, included most of what was known at the time, and became a bible for the field.

Garey & Johnson's book was published in 1979. There has been a lot of work since then on proving problems hard to solve. Hence a sequel is needed. While it is impossible for one book to encompass all of the work since then, this book will present some of that material. In particular we will cover, in broad terms, the following.

1. Since Garey & Johnson, several specific techniques have emerged for proving NP-hardness. We try to cover many of the major techniques, often illustrating with example problems that involve games and puzzles. In addition to personal preference, these examples make explicit the fun of finding reductions, and are useful for getting students excited about computer science.

2. Nuances on NP-completeness: hardness of approximation, counting (e.g., the number of satisfying assignments), truly exponential lower bounds (ETH), fixing parameters (FPT), and others.

3. Problems that are likely harder than NP (e.g., PSPACE, EXPTIME). For example, EXPTIME-complete problems are definitely not in P.

4. Problems that are in P but seem to require a certain polynomial amount of time (e.g., 3SUM-hard, APSP-hard).

Who Is the Audience for This Book

Ideally the reader should have had an undergraduate course in algorithms and have been at least exposed to NP-completeness. However, the book is self contained, so someone with the mathematical maturity that comes with being an undergraduate math major, even without a course in algorithms, could also read this book.

Acknowledgments

This book evolved from scribe notes taken by students in courses taught by Erik Demaine and Mohammad Hajiaghayi in the Fall of 2014. We thank the scribes from both courses:

Scribes for Erik Demaine's course:
Aviv Adler, Hugo Akitaya, Jeffrey Bosboom, Aahlad Gogineni, Neil Gurram, Chennah Heroor, Adam Hesterberg, James Hirst, Justin Kopinsky, Jason Ku, Andrea Lincoln Quanquan Liu, Fermi Ma, Billy Moses, Asa Oines, Nathan Pinsker, William Qian, Mikhail Rudoy, Ariel Schvartzman, Jeffrey Shen, Rajan Udwani, Chelsea Voss, Erik Waingarten, Jonathan Weed, Kevin Wu, Patrick Yang, and Yun William Yu. Edited by TAs Sarah Eisenstat and Jayson Lynch.

Scribes for Mohammad Hajiaghayi's course:

Karthik Abinav, Melika Abolhassani, Aditya Acharya, Ahmed Abdelkader Philip Dasler, Sina Dehghani, Soheil Ehsani, Huijing Gong, Thomas Pensyl, Manish Purohit, Saaed Seddighin, and Ateeq Sharfuddin.

We thank the following people for proofreading and helpful suggestions: Rob Brady, Hossein Esfandiari, Vahid Liaghat, Erik Metz, Kiarash Banihashem, Juan Chang, Jacob Gilbert, Jacob Ginzburg, Tom Hanson, Nathan Hayes, Edmund Horsch, Gihan Jayatilaka, Matt Kovacs-Deak, Yi Lee, George Li, Erika Melder, Alexander Mendelson, Erik Metz, Anthony Ostuni, Jacob Prinz, Joel Rajakumar, Benjamin Sela, Max Springer, Georgios Tsimos, and Patrick Yang.

We thank the following people for extra proofreading and scribe notes: Kiarash Banihashem, Samira Goudarzi, Jacob Gilbert, Nathan Hayes, Peyman Jabarzade, Erika Melder, and Jan Olkowski.

We thank Daniel Smolyak and Auguste Gezalyan for many of the figures. We thank Auguste Gezalyan for technical support. We thank Costis Daskalakis who was a guest lecturer in the course.

We thank Robert Hearn who was an initial collaborator on this book, in particular setting up the repository and the software used to write this book.

The first author thanks his father Martin for support throughout his life, in particular when creating this course and then book, and specifically for cinematography recording the lecture videos.

The second author thanks his wife Carolyn for her moral support and for helping him out when GitHub did something unexpected.

The third author thanks his wife Arefeh and his three children Ilya, Sana, and Sarah for their moral support and help. In particular thanks goes to Sana (7) and Ilya (9) for not proving $\mathsf{P} = \mathsf{NP}$ hence not making the book obsolete.

Chapter 0

Computational Complexity Crash Course

We begin with a brief introduction to the computational complexity needed to understand the rest of this book. In particular, we define some basic complexity classes, such as P, NP, coNP, PSPACE, EXPTIME, R, and how reductions define hardness and completeness. Figure 0.1 gives a visual overview of several of these classes and their relative difficulty. This chapter also serves as a general introduction to the entire book, with explicit pointers to the more advanced chapters.

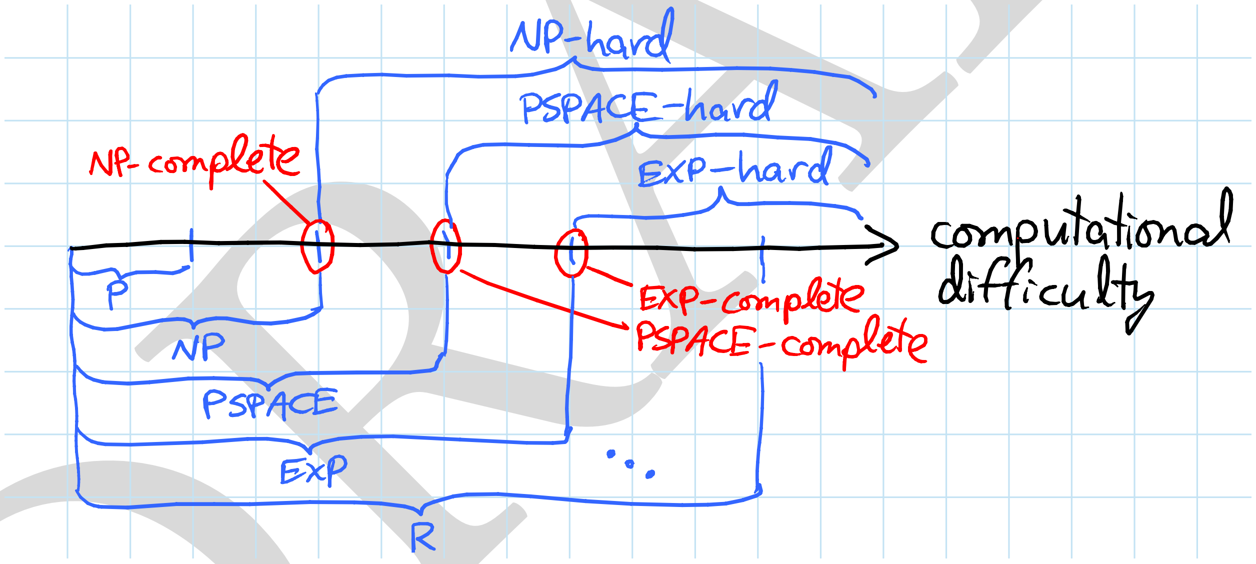


Figure 0.1: Hierarchy of Classes

0.1 P: Polynomial Time

Let A be a set of strings. The ***set membership problem*** for A asks for an algorithm that, given an input string x, quickly determines whether $x \in A$. How can we measure speed? Intuitively, we expect that longer strings take more time. Hence, to answer our question, we will study the algorithm's running time as a function of $n = |x|$, the length of the input x.

Definition 0.1.

1. A ***decision problem*** is a set membership problem. For example, "given a graph, does it have an Euler circuit?" is a decision problem, equivalent to set membership in the set of all Eulerian graphs.

2. A ***problem*** will usually mean decision problem. It will sometimes mean function evaluation: given x, compute $f(x)$. On rare occasions it will mean relations: given x, find *some* y such that $R(x, y)$ is true.

3. We also use the term ***instance*** to refer to the input x into a problem.

Definition 0.2.

1. P is the set of decision problems that can be solved in time that is bounded above by a polynomial in n, the size of the input.

2. FP is the set of functions that can be computed in time that is bounded above by a polynomial in n, the length of the input.

We sometimes write poly(n) or $n^{O(1)}$ for a quantity upper bounded by a polynomial in n.

For both P and FP we sometimes measure relative to another notion of size that is polynomially related to the input length. For example, a dense graph on n vertices takes $\Theta(n^2)$ to represent; however, we will cheat and take n to be the length of the input. For the purpose of telling whether a problem is in P, this cheat does not matter.

To prove that a problem is in P, one can write an algorithm and show that its runtime is bounded above by some polynomial. How do you prove that a problem is *not* in P? This needs a model of computation (as does making the definition of P rigorous). One common formal model is the Turing machine, which we will not define. A more familiar and equivalent model is the ***word RAM***, where an algorithm consists of a sequence of steps like "add the two integers in these locations in memory and put the result in this memory location" or "if this location in memory stores a positive integer, go to step 5", and each integer is limited to a number of bits equal to the machine word size w. All you need to know is the following:

1. Turing machines and word RAM work in discrete steps. Hence one can measure the number of executed steps and thus running time.

2. Let A be a decision problem. The following are equivalent:

   (a) There exists a program M in Python (or whatever your favorite programming language is) and a polynomial p such that, on input x, $M(x)$ will output $A(x)$ and terminate in $\leq p(|x|)$ steps.

   (b) There exists a word-RAM algorithm M and a polynomial q such that, on input x, $M(x)$ will output $A(x)$, and terminate in $\leq q(|x|)$ steps.

   (c) There exists a Turing machine M and a polynomial q such that, on input x, $M(x)$ will output $A(x)$, and terminate in $\leq q(|x|)$ steps.

3. We believe that *anything that can be computed at all* can be computed by a Turing machine or a word RAM. This assumption is often called the ***Church–Turing Thesis***. See Soare's article [Soa13] for a historical prospective on this thesis.

4. The ***Extended Church–Turing Thesis*** says furthermore that anything that can be computed in polynomial time can be computed in polynomial time by a Turing machine (or a word RAM). But this stronger thesis is inconsistent with some modern models of computation, such as quantum computing, leading to other complexity classes than P like BQP.

0.2 Reductions

To show that a problem is not in P (or likely to not be in P) we need a notion of the following: *if B can be solved (quickly) then A can be solved (quickly).*

Definition 0.3. Let A and B be decision problems. $\boldsymbol{A \leq_p B}$ if there exists $f \in \mathsf{FP}$ such that, for all x,

$$x \in A \text{ if and only if } f(x) \in B.$$

We say ***A reduces to B*** or ***there is a reduction from A to B***.

Exercise 0.4. Prove the following:

1. If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.

2. If $B \in P$ and $A \leq_p B$, then $A \in \mathsf{P}$.

3. If $A \notin P$ and $A \leq_p B$, then $B \notin \mathsf{P}$. (This is just the contrapositive of the last item; however, it is very important.)

Exercise 0.4.3 gives a technique to show that a problem is not in P: to show $B \notin P$, show that $A \leq_p B$ for some $A \notin \mathsf{P}$. Great! But we need some $A \notin P$ to start with. Alas: proving $A \notin P$ is hard. We consider a particular problem which will motivate our approach.

The reduction $A \leq_p B$ is defined so that, to answer whether $x \in A$, you get to ask *one* question to B and the answer must be the same. These are called ***polynomial-time many-one reductions***[1] or ***Karp reductions***. Polynomial-time reductions where you are allowed to ask many questions are called ***polynomial-time Turing reductions*** or ***Cook reductions***. In this book we will almost always deal with Karp reductions.

0.3 NP: Nondeterministic Polynomial Time

Consider the following classic decision problem:

SAT (short for SATISFIABILITY)
Instance: A Boolean formula $\varphi(x_1, \ldots, x_n)$.
Question: Is there a satisfying assignment for φ? That is, does there exist $(b_1, \ldots, b_n) \in \{\textsc{true}, \textsc{false}\}^n$ such that $\varphi(b_1, \ldots, b_n) = \textsc{true}$?

[1] "Many-one" or "many-to-one" means that the reduction can map multiple inputs to the same output. There are also 1-reductions where f has to be one-to-one.

The naive algorithm is to try all 2^n $b_1, \ldots, b_n \in \{\text{TRUE}, \text{FALSE}\}^n$. This shows SAT can be solved in time $2^{O(n)}$. But note the following: if you *already had* $b_1, \ldots, b_n$ *it would be easy to tell whether* $\varphi(b_1, \ldots, b_n) = \textit{TRUE}$. This does not make SAT any easier; however, this motivates our definition of NP below.

First we define some very useful notation and give an example of it.

Notation 0.5. Let $\boldsymbol{\exists^p y}$ denote "there exists y of length $p(|x|)$", where p is a polynomial and x is understood. Similarly, let $\boldsymbol{\forall^p y}$ denote "for all y of length $p(|x|)$", where p is a polynomial and x is understood.

Example 0.6. Let

$$A = \{(G, y) \mid y \text{ is a 3-coloring of } G\}.$$

The set of graphs that are 3-colorable can be expressed as

$$\text{3COL} = \{G \mid \exists y : (G, y) \in A\}.$$

The coloring y can be represented as a string over $\{0, 1\}$ of length $2n$, where n is the number of vertices in G.

Hence

$$\text{3COL} = \{G \mid \exists y : |y| = 2n \wedge (G, y) \in A\}.$$

The length of y does not really matter to us so long as it is a polynomial. Hence

$$\text{3COL} = \{G \mid \exists^p y : (G, y) \in A\}.$$

Exercise 0.7.

1. Show how to represent a (not necessarily proper) 3-coloring of a graph on n vertices with an element of $\{0, 1\}^{2n}$.

2. Let $c \geq 3$. Find r such that any (not necessarily proper) c-coloring of a graph on n vertices can be represented by an element of $\{0, 1\}^{rn}$.

Definition 0.8. $A \in \text{NP}$ if there exists a set $B \in \text{P}$ such that

$$A = \{x \mid \exists^p y : (x, y) \in B\}.$$

(NP stands for "Nondeterministic Polynomial" which comes from an equivalent definition of NP in terms of nondeterministic Turing machines which we do not use.)

The intuition is that

- If $x \in A$, then there is a *short* witness (y) that can be used to *verify* $x \in A$ quickly.

- If $x \notin A$, then there is no such witness.

We list many NP problems. For the first few we supply the witness. For the rest we leave finding the witness and hence proving the problem is in NP as an exercise.

Example 0.9.

1. 3COL: Given a graph, is it 3-colorable? The 3-coloring is the short quickly verified witness.

2. Clique: Given a graph G and a number k, does G have a clique of size k? (A ***clique*** is a set $U \subseteq V$ such that all distinct $u, v \in U$ satisfy $\{u, v\} \in E$.) The clique is the short quickly verified witness.

3. Independent Set: Given a graph G and a number k, does G have an independent set of size k? (An ***independent set*** is a set $U \subseteq V$ such that all distinct $u, v \in U$ satisfy $\{u, v\} \notin E$.)

4. Vertex Cover: Given a graph $G = (V, E)$ and a number k, does G have a vertex cover of size k? (A ***vertex cover*** is a set $U \subseteq V$ such that every edge $e \in E$ has an endpoint in U.)

5. Euler Cycle: Given a graph, does it have an ***Eulerian cycle*** (a cycle that visits every edge exactly once)? This problem happens to also be in P because a graph is Eulerian if and only if every vertex has even degree. So the statement Euler Cycle $\in$ NP is correct but not as interesting as Euler Cycle $\in$ P.

6. Factoring: Given a pair of numbers (n, a), is there a non-trivial factor of n that is $\leq a$? The factor is the short quickly verified witness. Note that the length of the input (n, a) is roughly $\lg n + \lg a$ because we represent numbers in binary.

7. Graph Isomorphism: Given a pair of graphs (G, H), are they isomorphic? In other words, is there is bijection between the vertices of G and the vertices of H that preserves the existence of edges?

8. Ham Cycle: Given a graph, does it have a ***Hamiltonian cycle*** (a cycle that visits every vertex exactly once)?

9. TSP: Given a weighted graph and a number k, is there a Hamiltonian cycle of weight $\leq k$? Note that Ham Cycle is a subcase of TSP.

10. Set Cover: Given sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ and an integer k, are there k of the S_i's whose union is all of $\{1, \ldots, n\}$?

11. Shortest Path: Given a graph G, two vertices s, t, and an integer c, is there a path from s to t that uses $\leq c$ edges? This problem is in P by using Breadth-First Search or Dijkstra's algorithm. So the statement Shortest Path $\in$ NP is correct but not as interesting as Shortest Path $\in$ P.

12. Subset Sum: Given integers $(a_1, \ldots, a_n, B)$ does some subset of $a_1, \ldots, a_n$ sum to B? The integers $a_1, \ldots, a_n, B$ are in binary, hence the length of the input is approximately $\lg a_1 + \cdots + \lg a_n + \lg B$.

13. Tetris: Given a position of a Tetris game, and knowledge of future falling pieces, is there a sequence of moves that does not lose the game? This puzzle or "perfect-information" version of Tetris is not how the game is usually played, where a player does not know the future falling pieces. Intuitively, though, if this problem is hard, then the version where the player does not know the future pieces is also hard.

14. 0-1 Programming: Given a matrix M of integers and vectors $\vec{b}, \vec{c}, \vec{d}$ of integers, does there exist a vector $\vec{x}$ where each component is 0 or 1 such that $M\vec{x} \le \vec{b}$ and $\vec{x} \cdot \vec{c} \ge d$?

15. Integer Programming: Given a matrix M of integers and vectors $\vec{b}, \vec{c}, \vec{d}$ of integers, does there exist a vector $\vec{x}$ of integers such that $M\vec{x} \le \vec{b}$ and $\vec{x} \cdot \vec{c} \ge d$? For this problem, membership in NP is more challenging than for any other problem on this list. The reason is that you need to show that, if there is *some* $\vec{x}$, then there is an $\vec{x}$ whose components are not to big.

Some of the problems in NP have been worked on for many years (predating the formal definition of P); however, no polynomial-time algorithm for them is known. We need a way to say "these are the problems in NP that we think are not in P".

Definition 0.10. Let B be a problem.

1. B is ***NP-hard*** if, for all $A \in$ NP, $A \le_p B$.

2. A problem B is ***NP-complete*** if $B \in$ NP and B is NP-hard.

It seems like it would be hard to find a natural problem that is NP-complete. To show B is NP-complete one needs to show that $A \le_p B$ for *every* $A \in$ NP. Every $A \in$ NP! Really! However, there is a natural NP-complete problem. Actually there are thousands! In the early 1970's, Cook [Coo71] and Levin [Lev73] (see [Tra84] for a translation of Levin's article into English and some historical context) independently proved the following:

Theorem 0.11 (Cook–Levin Theorem)**.** *SAT is NP-complete.*

Shortly after Cook showed there is one natural NP-complete problem, Karp [Kar72] showed 21 natural problems are NP-complete. We mention a few to give a sense of the variety of the problems: Ham Cycle, Subset Sum, and 0-1 Programming.

The proof of Theorem 0.11 codes Turing machine computations into formulas. Once SAT was shown to be NP-complete, Turing machines are no longer needed: to show A is NP-complete, just show SAT $\le_p A$. Or if B is already known to be NP-complete then show $B \le_p A$. This was what Karp did. As noted earlier, there are now thousands of NP-complete problems.

The problems in Example 0.9 are NP-complete except (1) Euler Cycle $\in$ P, (2) Shortest Path $\in$ P, (3) Factoring and Graph Isomorphism are in NP but not known to be either in P or NP-complete. We discuss these two problems later in this chapter.

For all the problems in Example 0.9, the proof that they are in NP is very easy. This is typical. When proving that a problem B is NP-complete we will usually omit the (easy) proof that $B \in$ NP. In the rare case that proving $B \in$ NP is hard or unknown, we will point it out.

The following notation is probably familiar to you; however, we include them for completeness (not NP-completeness) because they are needed for the next exercise.

Notation 0.12. Let A, B be set of strings.

1. $A \cdot B = \{xy \mid x \in A \text{ and } y \in B\}$. This is called the ***concatenation of A and B***.

2. $A^0 = \{e\}$ where e is the empty string.

3. $A^i = A \cdot A \cdot \cdots \cdot A$ where the A appears i times.

4. $A^* = A^0 \cup A^1 \cup A^2 \cup \cdots$.

Exercise 0.13. For each of the following statements, either prove it, disprove it, or state that it is unknown to science.

1. If $A, B \in \mathsf{P}$, then $A \cup B \in \mathsf{P}$.

2. If $A, B \in \mathsf{P}$, then $A \cap B \in \mathsf{P}$.

3. If $A, B \in \mathsf{P}$, then $A \cdot B \in \mathsf{P}$.

4. If $A \in \mathsf{P}$, then $\overline{A} \in \mathsf{P}$.

5. If $A \in \mathsf{P}$, then $A^* \in P$.

6. If $A, B \in \mathsf{NP}$, then $A \cup B \in \mathsf{NP}$.

7. If $A, B \in \mathsf{NP}$, then $A \cap B \in \mathsf{NP}$.

8. If $A, B \in \mathsf{NP}$, then $A \cdot B \in \mathsf{NP}$.

9. If $A \in \mathsf{P}$, then $\overline{A} \in \mathsf{NP}$.

10. If $A \in \mathsf{P}$, then $A^* \in \mathsf{NP}$.

Min Formula
Instance: A formula $\varphi(x_1, \ldots, x_n)$ and a number L.
Question: Is there a formula $\psi(x_1, \ldots, x_n)$ of length $\leq L$ that is equivalent to φ (i.e., always produces the same output)?

Example 0.14.

1. $\varphi(w, x, y, z) = (w \vee x) \wedge (w \vee y) \wedge (w \vee z)$. There is a shorter equivalent formula, namely $w \vee (x \wedge y \wedge z)$.

2. $\varphi(w, x, y, z) = w \vee x \vee y \vee z$. There is no shorter equivalent formula.

Exercise 0.15.

1. Show that, if $\mathsf{P} = \mathsf{NP}$, then Min Formula $\in \mathsf{P}$.

2. Vary the notion of length in various ways (e.g., number of $\wedge$, number of $\neg$) and determine whether $\mathsf{P} = \mathsf{NP}$ implies that this variant of Min Formula is in P.

0.4 coNP

The complement of SAT is the set of all formulas that have *no* satisfying assignment. We call this set CONTRA (for "contradiction"). Formally:

$$\text{CONTRA} = \{\varphi \mid \forall \vec{b} : \varphi(\vec{b}) = \text{FALSE}\}.$$

In terms of polynomial time, clearly SAT $\in$ P if and only if CONTRA $\in$ P because $\varphi \in$ SAT if and only if $\varphi \notin$ CONTRA. Is CONTRA in NP? Probably not. Contrast the following:

1. For Alice to convince Bob that $\varphi \in$ SAT, Alice can give Bob a satisfying truth assignment. The assignment is short and can be verified by Bob in polynomial time.

2. For Alice to convince Bob that $\varphi \in$ CONTRA, Alice can give Bob the truth table for φ. This is exponential in the size of φ and would take Bob exponential time to verify.

The question "is CONTRA in NP?" is asking whether there is a short verifiable witness that $\varphi \in$ CONTRA. This does not appear to be the case.

So how to classify CONTRA? We define a class ***coNP*** in two ways. This class is where CONTRA naturally lies.

Definition 0.16. $A \in$ coNP if there exists a set $B \in$ P such that

$$A = \{x \mid \forall^p y : (x, y) \in B\}.$$

(coNP stands for "co-Nondeterministic Polynomial" which comes from an equivalent definition of NP which we do not use.)

The intuition is that

- If $x \notin A$, then there is a *short* witness (y) that can be used to *verify* $x \in A$ quickly.
- If $x \in A$, then there is no such witness.

Another definition:

Definition 0.17. $A \in$ coNP if $\overline{A} \in$ NP.

It is widely believed that NP $\neq$ coNP.

We can define ***coNP-hardness*** and ***coNP-completeness*** similar to Definition 0.10. CONTRA is coNP-complete. Assuming NP $\neq$ coNP, CONTRA is not NP-complete.

0.5 The Winner is P $\neq$ NP

Gasarch [Gas19] has done three polls of what theorists think of P vs. NP, and other questions. In the last one, in 2019, 88% thought P $\neq$ NP. Why do theorists largely think P $\neq$ NP? Why do we think so?

1. There are thousands of problems that are NP-complete. For many of them people have worked on getting fast algorithms for a long time, predating the definition of P and NP. In a nutshell: *if (say) SAT is in P then someone would have discovered the algorithm for it by now.*

2. Intuitively, *verifying* seems easier than *finding*. As an example: finding a proof of a theorem is often hard, but verifying a proof is easy (or should be).

3. The assumption $\mathrm{P} \neq \mathrm{NP}$ has great explanatory power. We give one example. Consider the function version of Set Cover which is, given $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, find the minimal k such that k of the S_i's cover $\{1, \ldots, n\}$.

   (a) In 1979 Chvatal [Chv79] showed that a simple greedy algorithm yields a polynomial-time $\ln n$-approximation. That is, if the algorithm outputs x, then the answer is $\leq x \ln n$. Much work went into trying to obtain a $(1 - \varepsilon) \ln n$ approximation.

   (b) In 2013 Dinur and Steurer [DS13] showed that, if there exists $\varepsilon > 0$ such that there is a $(1 - \varepsilon) \ln n$ approximation, then $\mathrm{P} = \mathrm{NP}$.

   The algorithm and the lower bound (based on $\mathrm{P} \neq \mathrm{NP}$) are independent of each other. This phenomena—where a long-standing approximation is shown to be optimal assuming $\mathrm{P} \neq \mathrm{NP}$—is common. Hence the assumption $\mathrm{P} \neq \mathrm{NP}$ seems to settle many open problems in the direction we think they should go.

0.6 Strong NP-Completeness [Ch. 6]

Recall Subset Sum. The input consists of $n + 1$ numbers *in binary*. As such the problem is NP-complete. But what if the input numbers were specified *in unary*? Then the problem turns out to be in P:

Exercise 0.18. Show that, if the inputs are in unary, then Subset Sum $\in$ P.
Hint: Use dynamic programming.

Hence the NP-completeness of Subset Sum is related to the fact that the inputs are in binary. Some problems are NP-hard even when the inputs are in unary:

Definition 0.19.

1. A problem is ***weakly NP-hard*** if it is NP-hard when the inputs are in binary.

2. A problem is ***weakly NP-complete*** if it is in NP and weakly NP-hard.

3. A problem is ***strongly NP-hard*** if is NP-hard when the inputs are in unary.

4. A problem is ***strongly NP-complete*** if it is in NP and strongly NP-hard.

In Chapter 6 we will show that many problems are strongly NP-hard.

0.7 Intermediate Problems

In Example 0.9, we noted (without proof) that

- Euler Cycle $\in$ P.
- Clique, SAT, Set Cover, Subset Sum, Tetris, and 0-1 Programming are all NP-complete.

The above dichotomy is typical; most problems in NP turn out to be either in P or NP-complete. But there are exceptions (assuming P $\neq$ NP).

Definition 0.20. A problem is ***NP-intermediate*** if it is in NP but neither in P nor NP-complete.

Ladner [Lad75] showed the following:

Theorem 0.21. *If P $\neq$ NP, then there exists an NP-intermediate problem.*

The problem from Ladner's theorem is a contrived set constructed for the sole purpose of being in NP, not NP-complete, and not in P. But there are also a few natural and well-studied problems that are *conjectured* to be NP-intermediate:

1. ***Factoring***. We approach integer factoring via the set

   $$\text{Factoring} = \{(N, a) \mid \text{there is a nontrivial factor of } N \text{ that is } \leq a\}.$$

   It is easy to see that, if Factoring $\in$ P, then the problem of, given a number, find a nontrivial factor of it (or output that there isn't one) would be in FP. This problem is very important because many modern crypto system can be cracked if factoring is easy. Hence it has gotten much more attention than other problems.

   Currently there is no polynomial-time algorithm for Factoring, nor is there a proof that it is NP-complete. Algorithms for factoring are hard to analyze and depend on (widely believed) conjectures in number theory. The fastest known algorithm, the General Number Field Sieve, is believed to have runtime roughly $2^{1.93\, n^{1/3} (\lg n)^{2/3}}$, where $n = \lg N$ is the length of the input number N. This bound is small enough that the algorithm is practical for moderately large inputs. The naive algorithm for factoring takes time $2^{n/2}$. Hence reducing the time to roughly $2^{n^{1/3}}$ is a real improvement. In general, a ***subexponential*** bound of 2^{n^α} where $0 < \alpha < 1$ is strictly in between any polynomial bound of n^c and any exponential bound of 2^{n^c} where c is a positive integer.

   There is no clear consensus on whether Factoring is in P; however, if it is, then proving this will require new techniques. Here is why:

   (a) The last improvement in factoring algorithms was the Number Field Sieve in 1988.

   (b) There are reasons to think that the current methods yield algorithms with runtimes of the form $2^{L^t (\ln L)^{1-t}}$, where $0 < t < 1$. The General Number Field Sieve achieves $t = 1/3$. It is plausible that current techniques will solve Factoring in time (say) $2^{L^{1/10} (\ln L)^{9/10}}$ but not in P.

There is another possible route to fast factoring: Peter Shor [Sho94, Sho99] showed that factoring can be done in polynomial time on a quantum computer. This result has led to an explosion of interest in quantum computing.

So is Factoring NP-complete? Unlikely: if Factoring is NP-complete, then NP = coNP. We will guide you through a proof in Exercise 0.22. For more about factoring, see Wagstaff's book [Wag13].

2. ***Discrete Log.*** We describe discrete logarithm as a function and leave it to the reader to give the decision version. We write Discrete Log for both the function and decision version.

   Discrete Log: Given prime p, generator g, and $a \in \mathbb{Z}_p$, find x such that $g^x \equiv a \pmod{p}$.

   Much like factoring, (1) Discrete Log is important because, it if was easy to solve, the Diffie–Helman key exchange protocol in cryptography would be cracked; (2) Discrete Log is not known to be in P; (3) Discrete Log is not known to be NP-complete; (4) there are algorithms for Discrete Log that are better than the naive one, though still exponential (see [Wika]); (5) there has not been a better algorithm for Discrete Log since 1998; (6) Shor [Sho94, Sho99] has shown there is a quantum polynomial-time algorithm for Discrete Log; and (7) If Discrete Log is NP-complete, then NP = coNP; hence Discrete Log is unlikely to be NP-complete.

3. ***Graph Isomorphism.*** So far there has been no polynomial-time algorithm for Graph Isomorphism; however, there has also been no proof that it is NP-complete. The fastest algorithm for Graph Isomorphism, due to Babai [Bab16], has runtime $2^{(\lg n)^c}$ for some constant c ($c = 3$ seems likely). It is believed that a ***quasipolynomial*** running time of $2^{(\lg n)^c}$ with $c > 1$ is the best one can do with current methods. So is Graph Isomorphism NP-complete? The consensus is "no" for the following reasons:

   (a) If Graph Isomorphism is NP-complete, then all problems in NP are solvable in quasipolynomial time $2^{(\lg n)^{O(1)}}$ which seems unlikely. In particular, this would violate the Exponential Time Hypothesis described in Section 0.8 below.

   (b) Boppana et al. [BHZ87] proved that, if Graph Isomorphism is NP-complete, then $\Sigma_2 = \Pi_2$ (as defined in Section 0.13 below). The proof is slightly beyond the scope of this book.

4. ***Minimum Circuit Size Problem (MCSP)*** is the following problem: given the truth table of a boolean function f and an integer s, does f have a circuit with at most s logic gates?

   MCSP is not known to be in P, nor has it been proven to be NP-complete. Kabanets & Cai [KC00] and Allender & Hirahara [AH19] have provided arguments for why MCSP is NP-intermediate. See also Allender [All20] for a survey.

Exercise 0.22. Let Primes be the problem of, given a number, to determine whether it is prime. Let UFT be the theorem that every number can be factored uniquely into primes.

1. Look up or prove for yourself that Primes $\in$ NP. (It turns out that Primes $\in$ P [AKS04]; however, all we need for this exercise is that Primes $\in$ NP.)

2. Use UFT and Primes $\in$ NP to prove that $\overline{\text{Factoring}} \in$ NP.

3. Use Factoring $\in$ NP $\cap$ coNP to show that, if Factoring is NP-complete, then NP = coNP.

0.8 ETH [Ch. 7]

So how long does SAT actually take to solve? The hypothesis SAT $\notin$ P still allows for the possibility that, say, SAT $\in n^{O(\log n)}$. It is widely believed that SAT requires exponential time. More precisely, it is believed that there exists a sequence $s_3, s_4, \ldots$ such that kSAT requires $2^{s_k n}$ time. This belief is encapsulated by the ***Exponential Time Hypothesis (ETH)***. We will present this hypothesis and its implications in Chapter 7. In order to use ETH we need the reductions to be linear, not just polynomial.

We give one contrast between assuming P $\neq$ NP and assuming ETH.

1. Assuming P $\neq$ NP, Clique is not in polynomial time.

2. Assuming ETH, Clique requires time $2^{\Omega(n)}$.

The ETH still allows for the possibility that, say, kSAT $\in 2^{n/k}$. It is widely believed that as k gets larger, kSAT gets harder. In fact, it is believed that $\lim_{k\to\infty} s_k = 1$. This belief is encapsulated by the ***Strong Exponential Time Hypothesis (SETH)***. We will present this hypothesis and its implications in Chapter 7. We omit a contrast to P $\neq$ NP or ETH since the problems it applies to are somewhat technical.

ETH and SETH were proposed to get better lower bounds on NP-complete problems. However, perhaps surprisingly, they have also been used on the lower level of showing that some problems (essentially) require quadratic or cubic time. We examine this in Chapters 17 and 18.

0.9 FPT: Fixed Parameter Tractability [Ch. 8]

Consider the following two problems:

$$\text{Clique} = \{(G,k) \mid G \text{ has a clique of size } k\},$$
$$\text{Vertex Cover} = \{(G,k) \mid G \text{ has a vertex cover of size } k\}.$$

As noted after Theorem 0.11, both of these problems are NP-complete. Now let's fix k. Define

$$\text{Clique}_k = \{G \mid G \text{ has a clique of size } k\},$$
$$\text{Vertex Cover}_k = \{G \mid G \text{ has a vertex cover of size } k\}.$$

Both Clique$_k$ and Vertex Cover$_k$ are in time $O(n^k)$ and hence in P. This does not imply that Vertex Cover is in P because the exponent k is a function of the input size. In order to be in P, Vertex Cover would need to be in time $17n^2$ or $84n^3$ or some time bound that has a constant factor and a constant exponent, both of which are independent of the size of the input.

So both Clique$_k$ and Vertex Cover$_k$ seem to have complexity $O(n^k)$. Can this be improved? Does the exponent of the time bound have to depend on k? More to the point, are these problems similar or different? It turns out that they are different:

1. VERTEX COVER$_k$ can be solved in time $O(2^k n)$. (This does not imply that VERTEX COVER is in P because, the bound has the 2^k term which is a function of the input size.)

2. There are reasons to think that CLIQUE$_k$ requires $n^{\Omega(k)}$ time.

Problems like VERTEX COVER$_k$ are called ***fixed parameter tractable*** or FPT. We will study techniques to show that problems are likely not FPT in Chapter 8 and some in Chapter 7.

0.10 Complexity of Functions and Approximation [Ch. 9 & 10 & 11]

So far we have discussed *decision problems*. For example, CLIQUE is the problem of deciding whether a graph has a clique of a given size k. But the real problem people want answered is, given a graph G, to return the size of the largest clique. Or better, return a clique of the maximum size (and because there may be more than one, this would be that rare case where we study the complexity of a relation). If our only question is "can we solve CLIQUE in polynomial time", then this turns out to be a minor issue: all of these problems are essentially equivalent.

Exercise 0.23. Recall that FP is the set of all functions that are computable in polynomial time.

1. Let CLIQUESIZE(G) return the size of the largest clique in G. Show that

$$\textsc{Clique} \in \mathsf{P} \text{ if and only if } \textsc{CliqueSize} \in \mathsf{FP}.$$

2. Let FINDCLIQUE(G) return a clique of size CLIQUESIZE(G). Show that

$$\textsc{Clique} \in \mathsf{P} \text{ if and only if } \textsc{FindClique} \in \mathsf{FP}.$$

3. For all the decision problems A in Example 0.9, define a function version f_A. Show that $A \in \mathsf{P}$ if and only if $f_A \in \mathsf{FP}$.

With regard to polynomial time, computing CLIQUESIZE(G) is as hard as computing CLIQUE(G). But what about *approximating* CLIQUESIZE(G)? We will study the complexity of approximation in Chapters 9, 10, and 11.

0.11 Complexity Classes that Use Randomization

In this section we informally describe two randomized classes for, pardon the pun, completeness. Actually, there is an irony here in that these classes do not have complete problems.

Definition 0.24. A decision problem A is in ***Randomized Polynomial Time (RP)*** if there is a polynomial-time algorithm M that can flip coins satisfying the following properties:

- If $x \in A$, then the probability that $M(x)$ says $x \in A$ is $\geq \frac{1}{2}$.

- If $x \notin A$, then the probability that $M(x)$ says $x \notin A$ is 1.

Some facts about RP:

1. Miller [Mil76] obtained a polynomial-time algorithm for PRIMES that depends on the Extended Riemann Hypothesis being true. Rabin [Rab80] modified the algorithm to be in RP. For many years it remained open whether PRIMES $\in$ P until Agrawal et al. [AKS04] showed that it is.

2. There are very few problems that are in RP that are not known to be in P. We state one: given a polynomial $f(x_1, \ldots, x_n)$ and a prime p, is the polynomial identically zero over $\mathbb{Z}_p$?

3. There are reasons to think that P = RP. Google "Hardness versus Randomness" to see the reasons.

Exercise 0.25.

1. Let $0 < \alpha < \frac{1}{2}$. In the definition of RP, replace $\frac{1}{2}$ with α and call the resulting class RP_α. Show that $\mathrm{RP}_{1/2} = \mathrm{RP}_\alpha$.

2. Let $\alpha(n)$ be a decreasing function from $\mathbb{N}$ to $(0, \frac{1}{2})$. In the definition of RP, replace $\frac{1}{2}$ with $\alpha(|x|)$ and call the resulting class $\mathrm{RP}_{\alpha(n)}$. Is $\mathrm{RP}_{1/n^2} = \mathrm{RP}$? Is $\mathrm{RP}_{1/2^n} = \mathrm{RP}$?

Note that RP was defined to have 1-sided error. We now define a randomized class that has 2-sided error.

Definition 0.26. A set A is in ***Bounded Probabilistic Polynomial Time (BPP)*** if there is a polynomial-time algorithm M that can flip coins satisfying the following properties:

- If $x \in A$, then the probability that $M(x)$ says $x \in A$ is $\geq \frac{3}{4}$.
- If $x \notin A$, then the probability that $M(x)$ says $x \notin A$ is $\geq \frac{3}{4}$.

Some facts about BPP:

1. There are no natural problems that are known to be in BPP but not known to be in RP. There aren't even any known unnatural problems.

2. There are reasons to think that P = BPP. Google "Hardness versus Randomness" to see the reasons.

3. It is not known whether BPP $\subseteq$ NP.

4. Lautemann [Lau83] and Sipser [Sip83] proved that BPP $\subseteq \Sigma_2 \cap \Pi_2$ (as defined in Section 0.13 below). Lautemann's proof is simpler; however, Sipser's paper started the field of time-bounded Kolmogorov complexity.

Exercise 0.27.

1. Let $\frac{1}{2} < \alpha < 1$. In the definition of BPP, replace $\frac{3}{4}$ with α and call the resulting class BPP_α. Show that $\mathrm{BPP}_{3/4} = \mathrm{BPP}_\alpha$.

2. Let $\alpha(n)$ be a decreasing function from $\mathbb{N}$ to $(0, \frac{1}{2})$ In the definition of BPP, replace $\frac{3}{4}$ with $\alpha(|x|)$ and call the resulting class $\mathrm{BPP}_{\alpha(n)}$. Is $\mathrm{BPP}_{1/n^2} = \mathrm{BPP}$? Is $\mathrm{BPP}_{1/2^n} = \mathrm{BPP}$?

0.12 Counting Problems [Ch. 12]

Recall that SAT is the problem of, given a Boolean formula, *does there exist* a satisfying assignment? Consider the counting version:

> #SAT
> *Instance:* A Boolean formula $\varphi(x_1, \ldots, x_n)$.
> *Question:* For how many $\vec{b} \in \{\text{TRUE}, \text{FALSE}\}^n$ does $\varphi(\vec{b}) = \text{TRUE}$? That is, how many satisfying assignments does φ have?

Clearly SAT is easier (or just as easy) as #SAT. Note that the counting version is a function, not a set. It turns out that #SAT seems to be much harder than SAT. What about counting versions of other problems? For example, the problem #Clique is the following: given a graph G and a number k, how many cliques of size k are in G?

One way to show that #Clique is hard is to come up with a reduction f with the following property:

$$\text{If } \#\text{SAT} = L \text{ then } \#f(\text{Clique}) = L.$$

In short, the reduction preserves the number of witnesses. Such a reduction is called *parsimonious*. We show such a reduction in Chapter 12. However, we can get by with a weaker type of reduction. Let $A \in \text{NP}$ and $\#A$ be the number of witnesses. One way to show $\#A$ is hard is to come up with a reduction f and a injection g whose inverse is computable in P such that

$$\text{If } \#\text{SAT} = L \text{ then } \#f(\text{Clique}) = g(L).$$

We study the hardness of counting versions of problems in Chapter 12.

0.13 Polynomial Hierarchy

Consider an alternate formulation of Min Formula that asks whether a formula φ is the smallest formula with the same truth table:

$$\text{Min Formula} = \{\text{Boolean formula } \varphi \mid \forall \psi, |\psi| < |\varphi| : \exists \vec{b} : \varphi(\vec{b}) \neq \psi(\vec{b})\}.$$

This formulation of Min Formula does not put it into NP. We seem to need a $\forall$ and then a $\exists$.

We now define classes that use more quantifiers, so we will have a place to put Min Formula. Stockmeyer [Sto76] defined this "polynomial hierarchy".

Definition 0.28. Let $k \in \mathbb{N}$.

1. $A \in \Sigma_k$ if there exists $B \in \text{P}$ such that

$$A = \{x \mid \exists^p y_1 : \forall^p y_2 : \exists^p y_3 : \forall^p y_4 : \cdots : Q^p y_k : (x, y_1, \ldots, y_k) \in B\}$$

(Note that $\Sigma_1 = \text{NP}$.)

2. $A \in \Pi_k$ if there exists $B \in \text{P}$ such that

$$A = \{x \mid \forall^p y_1 : \exists^p y_2 : \forall^p y_3 : \exists^p y_4 : \cdots : Q^p y_k : (x, y_1, \ldots, y_k) \in B\}$$

(Note that $\Pi_1 = \text{coNP}$.)

Definition 0.29. The following interwoven hierarchies are known collectively as the ***Polynomial Hierarchy (PH)***:

$$\begin{tikzcd}
 & \Sigma_1 \arrow[r, "\subseteq"] \arrow[rdd, "\subseteq"] & \Sigma_2 \arrow[r, "\subseteq"] \arrow[rdd, "\subseteq"] & \Sigma_3 \arrow[r, "\subseteq"] \arrow[rdd, "\subseteq"] & \cdots \\
\Sigma_0 = \Pi_0 = \mathrm{P} \arrow[ru, "\subseteq"] \arrow[rd, "\subseteq"] & & & & \\
 & \Pi_1 \arrow[r, "\subseteq"] \arrow[ruu, "\subseteq"] & \Pi_2 \arrow[r, "\subseteq"] \arrow[ruu, "\subseteq"] & \Pi_3 \arrow[r, "\subseteq"] \arrow[ruu, "\subseteq"] & \cdots
\end{tikzcd}$$

Note: The notation Σ_i^p and Π_i^p is sometimes used for the polynomial hierarchy. This is because the notation Σ_i and Π_i are also used in computability theory, where there is no polynomial limit on the quantifiers. We will only discuss computability theory in Chapter 15, so there will be no confusion.

Exercise 0.30. Let $i \geq 1$.

1. Show that, if $\Sigma_i = \Pi_i$, then $\Sigma_i = \Pi_j = \Sigma_j$ for all $j \geq i$ ("polynomial hierarchy collapse").

2. Define Σ_i-hard, Σ_i-complete, Π_i-hard, and Π_i-complete.

It is believed that the polynomial hierarchy is proper, i.e., all containment relations in Definition 0.29 are strict, though not with the same confidence as the belief that $\mathrm{P} \neq \mathrm{NP}$.

Any NP problem can be modified to form a (possibly contrived) Σ_2 problem. If the original NP problem is NP-complete, then the modified problem is Σ_2-complete. We give one example.

Σ_2-SAT
Instance: Boolean formula with two sorts of variables: $\varphi(\vec{x}, \vec{y})$.
Question: Does there exist $\vec{b}$ such that, for all $\vec{c}$, $\varphi(\vec{b}, \vec{c}) = \textsc{true}$?

Exercise 0.31.

1. For all NP problems presented in this chapter, come up with a modified version that is in Σ_2 and does not seem to be in Σ_1.

2. Define Σ_i-SAT.

3. For all NP problems presented in this chapter, come up with a modified version that is in Σ_i and does not seem to be in Σ_{i-1}.

The following are known:

1. Σ_2-SAT is Σ_2-complete.

2. Min Formula is in Π_2 but is not known to be Π_2-complete.

We can view Σ_2-SAT as a game. The board is the given formula $\varphi(\vec{x}, \vec{y})$. First you pick an assignment for $\vec{x}$. Then your opponent picks an assignment for $\vec{y}$. If the resulting assignment makes φ TRUE, then you win; otherwise, your opponent wins. Note that $\varphi(\vec{x}, \vec{y}) \in \Sigma_2$ if and only if Alice can win.

One can extend this intuition to Σ_k and Π_k. Intuitively, Σ_k corresponds to a k-move game where you move first, and Π_k corresponds to a k-move game where your opponent moves first. An example of a real-world video game where puzzles become Σ_2-complete is "The Witness" with antibody clues [ABC+20]. We will also encounter Σ_2 in Section 12.12.

Exercise 0.32. Skim the paper by Schaefer & Umans [SU08] which presents over 80 problems complete on some level of the Polynomial Hierarchy.

0.14 EXPTIME, PSPACE, and EXPSPACE [Ch. 13 & 14 & 16]

Definition 0.33.

1. EXPTIME is the set of problems that can be solved in time that is exponential in the size n of the problem instance, i.e., in $2^{n^{O(1)}}$ time.

2. PSPACE is the set of problems solvable in polynomial *space* (memory), without any bound on running time.

3. EXPSPACE is the set of problems that can be solved in space that is exponential in the size n of the problem instance, i.e., in $2^{n^{O(1)}}$ space.

Exercise 0.34.

1. Show that NP $\subseteq$ EXPTIME.

2. Show that NP $\subseteq$ PSPACE.

3. Show that PSPACE $\subseteq$ EXPTIME.

4. Define EXPTIME-hard, EXPTIME-complete, PSPACE-hard, PSPACE-complete, EXPSPACE-hard, and EXPSPACE-complete.

Note: By a simple diagonalization argument, EXPTIME $-$ P $\neq \emptyset$ and EXPSPACE $-$ PSPACE $\neq \emptyset$. Hence, if a problem A is EXPTIME-complete, then $A \notin$ P, and similarly A being EXPSPACE-complete implies $A \notin$ PSPACE. This is in contrast to A being NP-complete, which we think implies $A \notin$ P but we do not know. These separations are corollaries to the more general ***Time Hierarchy Theorem*** and the ***Space Hierarchy Theorem***.

Example 0.35.

1. CHESS is the following problem: given a position in Chess (on an $n \times n$ board), and assuming both players play perfectly, will white win? For most natural versions of Chess, this problem is EXPTIME-complete.

2. In Chapters 13, 14, and 16, we will discuss more games that are complete or hard in these classes.

3. Let QBF (Quantified Boolean Formula) consist of all expressions of the form

$$\exists \vec{x}_1 : \forall \vec{x}_2 : \exists \vec{x}_3 : \forall \vec{x}_4 : \cdots : \exists \vec{x}_{k-1} : \forall \vec{x}_k : \varphi(\vec{x}_1, \ldots, \vec{x}_k)$$

that are true. QBF is PSPACE-complete.

PSPACE thus contains all of the Polynomial Hierarchy (including all Σ_i and Π_i), as QBF allows for an arbitrary number of alternations in quantifiers.

0.15 NSPACE: Nondeterministic Space

Our definition of NP involves a witness y. That is, $x \in A$ if and only if there is a short string y that can be used to prove $x \in A$. Our definition of NPSPACE is similar.

Definition 0.36. $A \in$ NPSPACE if there exists a Turing machine or word-RAM algorithm M such that

1. on an input (x, y), M uses space (memory) bounded above by a polynomial in $|x| + |y|$;
2. if $x \in A$, then there is a y such that $M(x, y)$ outputs "yes"; and
3. if $x \notin A$, then $M(x, y)$ outputs "no" for any y.

In other words, there is problem $B \in$ PSPACE satisfying

$$A = \{x \mid \exists^p y : (x, y) \in B\}.$$

(NPSPACE stands for "Nondeterministic Polynomial Space".)

More generally, for any function $S : \mathbb{N} \to \mathbb{N}$, we can define SPACE$(S(n))$ and NSPACE$(S(n))$ to allow $S(n)$ space on an input of size n. Savitch [Sav70] showed the following beautiful relation between SPACE and NSPACE:

Theorem 0.37. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function, with $f(n) \geq n$. Then NSPACE$(f(n)) \subseteq$ SPACE$(f(n)^2)$. In particular, NPSPACE = PSPACE.*

Exercise 0.38. Either prove Savitch's Theorem or look up its proof, read it, and understand it.

If $S(n)$ is sublinear in n, we can still define SPACE$(S(n))$ and NSPACE$(S(n))$ in a meaningful way. Specifically, we restrict the input (x, y) to be read only, and allow the algorithm only $S(n)$ read/write memory.

One well-studied example is NL = NSPACE$(O(\lg n))$, the class of problems that can be solved in nondeterministic logarithmic space. NL has a notion of NL-completeness. For example, the following problem is NL-complete: given a directed graph G and two vertices s, t, is there a directed path from s to t? The proof is easy because one can view a nondeterministic logarithmic-space computation as a directed graph. There is a belief that L $\neq$ NL but it is not as strong as the belief that P $\neq$ NP.

0.16 R: Decidable Sets [Ch. 15 & 16]

Definition 0.39. R is the set of decision problems that can be solved by a Turing machine or word-RAM algorithm, with no limit on how long it runs other than "finite". Problems in R are also called ***decidable***.

The R stands for ***recursive*** because at one time "decidable" sets were called "recursive". Soare's article [Soa96] has an excellent historical perspective on the reason to change the terminology from "recursive" to "decidable". (Sometimes in the literature R stands for Randomized Polynomial Time. When this comes up, we will use RP.)

Essentially all problems in this book are decidable, i.e., in R. In Chapters 15 and 16, we will discuss problems that are undecidable.

0.17 Arithmetic Hierarchy

Given two problems that are undecidable, is there a way of saying that one is more undecidable? What measure of complexity can you use? The ***Arithmetic Hierarchy*** is a way to classify problems using the number of quantifiers. Given the order we presented the material in, you might think that the Arithmetic Hierarchy is modeled after the Polynomial Hierarchy from Section 0.13; however, it is the other way around. The Arithmetic Hierarchy was defined by Kleene [Kle43] in 1943. More generally, much of the early work in complexity theory was modeled after earlier work in computability theory.

Definition 0.40. We give four equivalent definitions of when a decision problem A is ***computably enumerable***, written $A \in \mathsf{CE}$:

1. If there exists a Turing machine or word-RAM algorithm M such that

$$A = \{x \mid \exists y : M(x) \text{ halts and outputs } x\}.$$

   (So A is the *range* of a computable function. Note that M might not halt on some inputs.)

2. If either $A = \emptyset$, or there exists a Turing machine or word-RAM algorithm M that halts on all inputs and satisfies

$$A = \{x \mid \exists y : M(x) \text{ (halts and) outputs } x\}.$$

   (So A is empty or the *range* of a total computable function.)

3. If there exists a Turing machine or word-RAM algorithm M such that

$$A = \{x \mid M(x) \text{ halts}\}.$$

   (So A is the *domain* of a computable function.)

4. If there exists a $B \in \mathrm{R}$ such that

$$A = \{x \mid \exists y : (x, y) \in B\}.$$

Note: What we call ***computably enumerable (c.e.)*** is also called ***recursively enumerable (r.e.)***. Soare [Soa96] has tried to get the community to change from "r.e." to "c.e." and gives good arguments for the change.

Exercise 0.41. Show that all four definitions of c.e. in Definition 0.40 are equivalent.

The fourth definition of c.e. in Definition 0.40 looks like NP which is identical to Σ_1. The next exercise asks you to define Π_1, Σ_2, Π_2, etc. in the context of decidability.

Exercise 0.42.

1. In Section 0.13 on the Polynomial Hierarchy, we defined Σ_i and Π_i by adding alternating polynomial-bounded quantifiers to P. Define similar classes by adding alternating unbounded quantifiers to R. Call these classes Σ_i and Π_i also (they will only be used within this problem so there should be no confusion). This is the ***Arithmetic Hierarchy***.

2. Let $M_0, M_1, \ldots$ be a list of *all* Turing machines or word-RAM algorithms in some order. For each of the following decision problems, determine which Σ_k or Π_k it is in. Try to make k as low as possible.

$$\begin{aligned} \text{FIN} &= \{i \mid M_i \text{ halts on an infinite number of inputs}\} \\ \text{TOT} &= \{i \mid M_i \text{ halts on all inputs}\} \end{aligned}$$

In Section 15.4 we will briefly discuss one problem that does not involve Turing machines yet is not in the arithmetic hierarchy.

0.18 A Few Separations of Complexity Classes

Separating P from NP seems quite hard. Now that we have introduced more complexity classes, though, there are a few pairs of classes that we know are different. In all cases the proof is by a simple diagonalization argument which we omit.

Theorem 0.43.

1. *P $\subsetneq$ EXPTIME $\subsetneq$ R.*

2. *P $\subseteq$ NP $\subseteq$ EXPTIME $\subsetneq$ R.*

3. *P $\subseteq$ NP $\subseteq$ PSPACE $\subsetneq$ EXPSPACE $\subsetneq$ R.*

4. *P $\subseteq$ NP and NP $\subseteq$ EXPTIME so, by Part 1, one of these two is proper.*

5. *P $\subseteq$ PSPACE and PSPACE $\subseteq$ EXPTIME so, by Part 1, one of these two is proper.*

Refer back to Figure 0.1 for a visual overview.

0.19 Polynomial Lower Bounds [Ch. 17 & 18]

Once we know that a problem is in P the question arises, what is its running time? Can we have a notion similar to NP-hardness to show that problems within P are unlikely to have subquadratic algorithms? Subcubic algorithms?

To show that problems are unlikely to be in subquadratic time (that is, $O(n^{2-\delta})$ for some δ), we need a base problem (similar in spirit to SAT) that is unlikely to be in subquadratic time. There is such a problem.

> 3SUM
> *Instance:* n integers.
> *Question:* Do three of the integers sum to 0?
> *Note:* We will consider any arithmetic operation to have unit cost.

Despite enormous effort, nobody has obtained a subquadratic algorithm for 3SUM in a reasonable model of computation. In Chapter 17 we show that many problems are unlikely to be in subquadratic time by (1) assuming that 3SUM is not in subquadratic time, and (2) using an appropriate notion of reduction.

To show that problems are unlikely to be in subcubic time (that is, $O(n^{3-\delta})$ for some δ), we need a base problem (similar in spirit to SAT) that is unlikely to be in subcubic time. There is such a problem.

> All Pairs Shortest Paths (APSP)
> *Instance:* A weighted directed graph $G = (V, E, w)$. The weights are in $\mathbb{N}$.
> *Question:* For all pairs of vertices x, y, compute $\text{dist}_G(x, y)$, the length of the shortest path between x and y.
> *Note:* We will consider any arithmetic operation to have unit cost.

Despite enormous effort, nobody has obtained a subcubic algorithm for APSP. In Chapter 18 we show that many problems are unlikely to be in subcubic time by (1) assuming that APSP is not in subcubic time, and (2) using an appropriate notion of reduction.

0.20 Online Algorithms [Ch. 19]

An ***online problem*** is one where (1) the input is given in pieces (e.g., requests for memory access), and (2) the answer is a sequence of answers (e.g., assignments of whether to put a page in cache or memory), where each answer must be given each time you get a piece of the input *before* the rest of the input is given.

For such problems the issue is *not* how fast the algorithm runs. The algorithm needs to, as soon as it gets a new piece of information, give a response quickly. There is another goal in mind. For example, in the case of memory access, the goal is to minimize the number of times that a page that is not in the cache is requested. The issue is then how well the algorithm does on its goal *compared to what the optimal would be if we knew all future information.*

We study lower bounds for online algorithms in Chapter 19. The main tool used is adversary arguments, where an adversary will, given the algorithm, find a way to input the data that causes the algorithm to do badly. These lower bounds are unconditional. There is no need to assume some problem is hard.

0.21 Streaming Algorithms [Ch. 20]

Streaming algorithms are similar to online algorithms in that the data comes in pieces. A ***streaming problem*** is one where (1) the input is given in pieces (e.g., edges of a graph), (2) we may allow several passes through the data, and (3) the answer is a string (e.g., an answer to "does the graph have a connected component of size 100").

For such problems, the issue is *not* how fast the algorithm runs. We note that the algorithms involved are usually quite fast. The issue is twofold: how much memory does the algorithm use (it cannot store all of the data as we think of the the data stream as being enormous), and how many passes over the data does the algorithm use. There is often a tradeoff between these two parameters.

We study lower bounds for streaming algorithms in Chapter 19. The main tool used is communication complexity. These lower bounds are unconditional: there is no need to assume some problem is hard.

0.22 Parallel Algorithms [Ch. 21]

A ***parallel algorithm*** is one where many machines work on a problem at the same time. In Chapter 21, we study a recent model of parallelism called ***Massively Parallel Computation MPC***. The key parameters in this model are (1) the number of rounds a computation takes, and (2) the memory each machine has. The lower bounds are often tradeoffs between these two parameters. There are two kinds of lower bounds:

1. **Unconditional.** These lower bounds associate to a computation certain polynomials and can then use algebra to get lower bounds. There is no need to assume some problem is hard.

2. **Conditional.** To show that problems are unlikely to have a good MPC algorithm (we formalize this in Chapter 21), we need a base problem (similar in spirit to SAT) that is unlikely to have a good MPC algorithm. There is such a problem.

1vs2-Cycle
Instance: An undirected graph $G = (V, E)$ which we are promised is either one cycle or the union of two cycles.
Question: Determine whether the graph is one cycle or the union of two cycles.

Despite some effort, nobody has obtained a good MPC algorithm for 1vs2-Cycle. In Chapter 21, we show that some problems are unlikely to have good MPC algorithms by (1) assuming that 1vs2-Cycle does not have a good MPC algorithm, and (2) using an appropriate notion of reduction.

0.23 Game Theory [Ch. 22]

Imagine that Alice and Bob are playing a game. It is a simple game: each player simultaneously chooses 1 out of n options, and that pair (Alice's Choice, Bob's Choice) determines how many points each person gets. They will play this game many times.

Nash showed that, if they are both allowed to have a probabilistic strategy (often called a ***mixed strategy***), then there exists a pair (Alice's strategy, Bob's strategy) such that both players know that, if they deviate from the strategy but their opponent does not, then they will do worse than if they stuck with their strategy. Such an ordered pair is called a ***Nash equilibrium***.

Nash's proof that a Nash equilibrium exists does not give a method for finding it. This leave us in a curious position:

1. The problem "is there a Nash Equilibrium" is trivial: the answer is YES.

2. The problem "find the Nash Equilibrium" seems hard.

There are reasons to think that finding the Nash Equilibrium is not NP-hard. Hence we need another way to prove that it is hard.

There are other problems where a solution always exists but finding it seems to be hard. To show that problems of this type are unlikely to be in polynomial time, we need a base problem (similar in spirit to SAT) that is unlikely to be in polynomial time. There is such a problem.

End of Line EOL
Instance: Two circuits P (for Previous) and N (for Next) on $\{0,1\}^n$ such that, for all $x \in \{0,1\}^n$, $P(x)$ and $N(x)$ each returns either "no" or an element of $\{0,1\}^n$. We interpret the circuits as describing a graph as follows: If $P(x) = y$, then there is an edge from x to y; and if $N(x) = y$, then there is an edge from y to x. Since these are the only edges, every vertex has indegree and outdegree ≤ 1. Exactly one of $P(0^n)$ and $N(0^n)$ is "no". So 0^n is unbalanced, which means that its indegree is not equal to its outdegree.
Question: We know there is another unbalanced node. Find it!

Despite some effort, nobody has obtained a polynomial time for EOL. In Chapter 22 we show that some problems, including finding a Nash Equilibrium, are unlikely to be in polynomial time by (1) assuming the EOL $\notin$ P, and (2) using an appropriate notion of reduction.

Part I

NP and Its Variants

Chapter 1

SAT and Its Variants

1.1 Introduction

In this chapter we first look at many variants of SAT (the base version of which was defined in Section 0.3). Some are in P and some are NP-complete. Curiously, none seem to be intermediate or status-unknown. We will state a theorem that explains this phenomena. Second we will use these variants of SAT to prove many problems NP-complete.

1.2 Variants of SAT

We list many variants of SAT and say which are in P and which are NP-complete. We mostly do not provide proofs; however, you should consider each statement to be an exercise.

1.2.1 CNF SAT, DNF SAT, and Circuit SAT

Definition 1.1. Let φ be a Boolean formula.

1. The terms x_i and $\neg x_i$ are called ***literals***.
2. φ is in ***Conjunctive Normal Form (CNF)*** if

$$\varphi = C_1 \wedge \cdots \wedge C_k$$

   where the C_j's are ORs of literals.

3. if φ is a CNF formula, then the C_i's are called ***clauses***.
4. φ is in ***Disjunctive Normal Form (DNF)*** if

$$\varphi = D_1 \vee \cdots \vee D_k$$

   where the D_j's are ANDs of literals.

5. A ***Boolean circuit*** is a circuit with inputs $x_1, \ldots, x_n$, one output, and the gates are AND, OR, and NOT. See Figure 1.1 for an example. The gates with a straight line base are AND, the gates with a curved line base are OR, and the circle is a negation. We leave it as an exercise to write down the Boolean formula computed by this circuit.

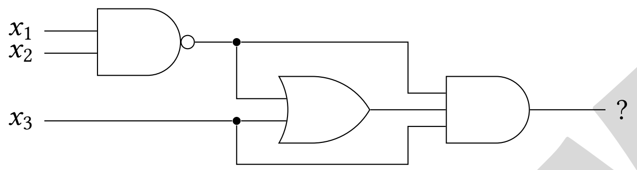


Figure 1.1: A Boolean circuit

Notation 1.2. If we define a type of formula X (e.g., CNF), then the problem XSAT (e.g., CNF SAT) will be, given a formula of the form XSAT, is it satisfiable. (Soon we will soon have a convention that formulas are in CNF form unless otherwise specified by such a prefix.) For Boolean circuits we will use Circuit SAT.

Theorem 1.3.

1. *CNF SAT is NP-complete. Cook [Coo71] and Levin [Lev73] (see [Tra84] for a translation of Levin's article into English and some historical context) proved this, though they did not state it in this form.*
2. *DNF SAT is in P. This is an easy exercise.*
3. *Circuit SAT is NP-complete. This is an easy consequence of CNF SAT being NP-complete.*

Exercise 1.4. Since CNF SAT is NP-complete and DNF SAT is in P, one approach to proving P = NP is to find a function $f \in$ FP that will take a CNF formula φ and return an equivalent DNF formula ψ. Alas this cannot work:

Prove that any DNF formula that is equivalent to

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n)$$

requires 2^n clauses.

Convention 1.5. Unless otherwise noted, we will assume that all the formulas we encounter are in CNF form. Hence, we use SAT to mean CNF SAT.

1.2.2 Variants of SAT that are Mostly NP-Complete

Definition 1.6. 2SAT is SAT restricted to formulas that have ≤ 2 literals per clause. 3SAT is SAT restricted to formulas that have ≤ 3 literals per clause. We will soon generalize this concept.

Exercise 1.7.

1. Show that 2SAT is in P. (This is folklore.)

2. Show that 3SAT is NP-complete. Cook [Coo71] proved this. (This is the most commonly used NP-complete problem for reductions.)

There are many variations of SAT. One can restrict the number of literals in a clause, or other aspects of the input. One can also put conditions on what kind of satisfying assignment you want. There are many papers with different (and inconsistent) notations. Fortunately, Filho [Fil19a], in his Master's Thesis, devised a great system of notation that we use. He also unified many old results and proved some new ones. His system encompassed far more variants of SAT than we will discuss.

We first look at restricting the number of literals in a clause and the number of times a variable can appear in the formula. Even with these simple variants there are subtleties.

Definition 1.8. Let $a, b \in \mathbb{N}$. In all of the problems below the goal is to find a satisfying assignment. What varies is the form of the input formula.

1. aSAT: every clause has $\le a$ literals per clause. Note that if $a \ge 3$ then $(x \vee x \vee y)$ and $(x \vee \neg x \vee y)$ are allowed.
2. EaSAT: every clause has *exactly* a literals per clause. Again note that if $a \ge 3$ then $(x \vee x \vee y)$ and $(x \vee \neg x \vee y)$ are allowed.
3. EUaSAT: every clause has *exactly* a literals per clause, and every variable within a clause occurs *uniquely*. Note that if $a \ge 3$ then $(x \vee x \vee y)$ and $(x \vee \neg x \vee y)$ are *not* allowed.
4. aSAT-b: every clause has $\le a$ literals per clause, and every variable occurs $\le b$ times. (If x occurs and $\neg x$ occurs, then that is 2 occurrences.)
5. aSAT-Eb: every clause has $\le a$ literals per clause, and every variable occurs *exactly* b times. (If x occurs and $\neg x$ occurs, then that is 2 occurrences.)
6. We leave it to the reader to define EaSAT-b, EUaSAT-b, EaSAT-Eb, EUaSAT-Eb.

We state and prove a theorem, due to Tovey [Tov84], that shows that these seemingly small differences in the form of the formula create big differences in complexity.

Theorem 1.9.

1. *3SAT-3 is NP-complete.*
2. *Let φ be a formula such that (1) there are exactly 3 literals per clause, (2) every variable occurs ≤ 3 times, and (3) every variable within a clause occurs uniquely. Then φ is satisfiable.*
3. *EU3SAT-3 $\in$ P since every formula of this form is satisfiable by Part 2.*

Proof.
1) We show 3SAT $\le_p$ 3SAT-3.

Given a formula φ in 3CNF, form we produce a formula φ' such that (1) every variable occurs ≤ 3 times and (2) $\varphi \in$ SAT if and only if $\varphi' \in$ SAT.

For each variable x such that either x or $\neg x$ occurs in φ do the following:

1. If x occurs ≤ 3 times, then do nothing.
2. If x occurs $m \ge 4$ times, then introduce new variables $x_1, \ldots, x_m$, and do the following:
   - Replace the ith occurrence of x with x_i.
   - For $1 \le i \le m-1$, add the clause $x_i \to x_{i+1}$ (formally, this is $\neg x_i \vee x_{i+1}$).
   - Add the clause $x_m \to x_1$.
3. Output φ'.

Clearly (1) φ' is of the right form and (2) $\varphi \in$ SAT if and only if $\varphi' \in$ SAT.

2) Let φ be a formula that satisfies the premise. Consider the bipartite graph with (1) clauses on the left, (2) variables on the right, (3) an edge between C and x if either x or $\neg x$ occurs in C.

Every clause has degree 3. Every variable has degree ≤ 3. Hence, by a corollary to Hall's Theorem (we discuss this in the exercises following this theorem), there is a matching of clauses to variable. That is, there is a set of disjoint edges where every clause is the endpoint of one of them. If C is matched to x, then (1) set xTRUEif $x \in C$, (2) set xFALSEif $\neg x \in C$. This is clearly a satisfying assignment. □

Exercise 1.10. Let $G = (A, B, E)$ be a bipartite graph. A ***matching from A to B*** is a disjoint set of edges where every element of A is an endpoint. If $X \subseteq A$ then,

$$E(X) = \{y \in B \mid \exists x \in X : (x, y) \in E\}.$$

1. Prove the following are equivalent.
   (a) for all $X \subseteq A$, $|E(X)| \ge |X|$
   (b) there is a matching from A to B.

   This equivalence is Hall's Theorem.
2. Let $k \in \mathbb{N}$. Prove that if (1) every vertex in A has degree $\ge k$, and (2) every vertex in B has degree $\le k$, then there is a matching from A to B. This is a corollary to Hall's Theorem.

Exercise 1.11. For each $a, b \in \mathbb{N}$, for each formula type from Definition 1.8, determine the status (P or NP-complete) of all of the SAT problems restricted to that formula-type with parameters a, b. Some we have already done for you. Warning- Some are open.

Exercise 1.12. Show that in the proof of Theorem 1.9 φ and φ' have the same number of satisfying assignments.

We define a SAT problem where either the formula is satisfiable or very far from satisfiable.

GAP 3SAT-5
Instance: A number $\varepsilon > 0$ and a formula φ such that (1) φ has ≤ 3 literals per clause, and (2) all variables of φ occur ≤ 5 times. We are promised that φ is either (1) satisfiable or (2) at most $(1-\varepsilon)$ of its clauses can be simultaneously satisfied.
Question: Determine which is the case. (This problem may seem unnatural. Indeed, it was a means to an end: it was used to show limits on how well you can approximate Set Cover.)

We state (but do not prove) a rather difficult theorem, due to Feige [Fei98], about these kinds of formulas.

Theorem 1.13. GAP *3SAT-5 is NP-hard.*

In all of the above examples, we had SAT as our goal: is there a way to satisfy the formula? We can place conditions on how we want the formula to be satisfied. We give some examples and then say how the notation handles these conditions.

Definition 1.14.

1. The condition NAE- means that, within a clause, not all of its literals get the same truth value. In an NAE- assignmentTRUE andFALSEare symmetric in that every clause must have at least one of each.

2. The condition 1-IN- means that exactly 1 literal from each clause is true. One could replace 1 with 2 or any number.

3. To indicate you are demanding that condition be satisfied, place the condition right before everything else. Example:

1-IN-EUaSAT-b
Instance: A formula φ such that (1) every clause has exactly a literals (2) every variable occurs $\leq b$ times, and (3) every variable in a clause occurs uniquely.
Question: Is there a satisfying assignment of φ where every clause has exactly 1 literal set to true?

In the above examples, we limited the number of literals per clause. We may want to limit other aspects of a clause. We give the only example we will be using and then say how the notation handles this limitation.

Definition 1.15.

1. A formula is ***monotone*** if, for every clause, either all of its literals are positive or all of its literals are negative.

2. We put the word MONOTONE before the condition. We give two examples:

MONOTONE aSAT
Instance: A formula φ such that (1) every clause has $\leq a$ literals (and a clause can have the same variable twice), and (2) every clause has either all the literals positive or all of the literals negative.
Question: Is φ satisfiable?

MONOTONE NAE-aSAT-Eb
Instance: A formula φ such that (1) every clause has $\leq a$ literals (and a clause can have the same variable twice), (2) every variable occurs exactly b times, (3) every clause has either all the literals positive or all of the literals negative.
Question: Is there a satisfying assignment of φ where every clause has both a literal set toTRUEand a literal set to F?

Theorem 1.16.

1. *(Gold [Gol78]) MONOTONE 3SAT is NP-complete.*
2. *(Darmann et al. [DDD18]) MONOTONE 3SAT-E4 is NP-complete. This solved an open problem in an earlier version of this book.*

What if the goal is not to satisfy *all* of the clauses, but instead to satisfy as many as possible? We prepend the word MAX to indicate that goal. So MAX 2SAT is the function that will, given a 2CNF formula φ, returns the maximum number of clauses that can be simultaneously satisfied. Since it is a function, we cannot say it is NP-complete, though we can say it is NP-hard.

Garey et al. [GJS76] proved the following.

Theorem 1.17. *Computing MAX 2SAT is NP-hard.*

Exercise 1.18. Show that the following problem is NP-complete: Given a 3CNF formula, is there a satisfying assignment such that a majority of the clauses have all three literals set to T?

1.2.3 Variants of SAT that are Mostly in P

Theorem 1.19. *The following versions of SAT are in P.*

1. *(Horn [Hor51]) HORN SAT. Each clause has ≤ 1 positive literal.*
2. *(Schaefer [Sch78]) DUAL HORN SAT. Each clause has ≤ 1 negative literal. DUAL HORN SAT $\in$ P follows easily from HORN SAT $\in$ P.*
3. *(Lewis [Lew78]) RENAMEABLE HORN SAT. There is a set of clauses such that if you replace every variable in those clauses with a negation, then the formula is Horn.*

1.2.4 More Variants of 3SAT that are Mostly NP-Complete

Theorem 1.20. *In all of the following problems the formula given is 3CNF.*

1. *(Schaefer [Sch78]) 1-IN-3SAT. We seek a satisfying assignment where* exactly *one literal per clause is TRUE. This problem is NP-complete. This reduction is well known and is even on Wikipedia.*
2. *MONOTONE 1-IN-3SAT. There are no negation signs, and we seek a satisfying assignment where* exactly *one literal per clause is TRUE. This problem is NP-complete. This is not well known, hence we prove it right after we finish stating this theorem. (The terminology is a little confusing, as "monotone" here means all positive, whereas earlier monotone meant all positive or all negative.)*
3. *(Schaefer [Sch78]) MONOTONE NOT-EXACTLY-1-IN-3-SAT. There are no negation signs, and we seek an assignment where either 0,2, or 3 of the variables in each clause are TRUE. This problem is trivially in P.*

4. *(Schaefer [Sch78]) NAE-3SAT. We seek a satisfying assignment where no clause has all 3 variables* TRUE. *Note that we are not allowing any clause to have its literals go* FALSE-FALSE-FALSE *(since then φ would not be satisfied) or* TRUE-TRUE-TRUE *(since then all 3 are* TRUE*). There is a nice symmetry between* TRUE *and* FALSE.

5. *Monotone NAE-3SAT. The formula has no negations, and we seek a satisfying assignment where no clause has all 3 variables* TRUE. *This problem is NP-complete.*

Since the next theorem seems to have never been written down, we provide a proof for it. We will use it to prove Cryptarithms is NP-complete, which is Theorem 1.32.

Theorem 1.21. *1-in-3SAT $\le_p$ Monotone 1-in-3SAT. Hence Monotone 1-in-3SAT is NP-complete.*

Proof. Given $\varphi = C_1 \wedge \cdots \wedge C_k$ (on variables $x_1, \ldots, x_n$) where each C_i has 3 literals, we construct

$$\varphi' = C_1' \wedge \cdots \wedge C_k' \wedge D_1 \wedge \cdots \wedge D_n \wedge E$$

such that the following holds:

1. Each C_i', D_j, E has three positive literals.

2. The following are equivalent:

   - There is a truth assignment that makes φ true by making *exactly* 1 literal per clause true.
   - There is a truth assignment that makes φ' true by making *exactly* 1 literal per clause true.

We construct φ' as follows:

1. Let TRUE, FALSE be suggestively named new variables. We add the clause $E =$ TRUE $\vee$ FALSE $\vee$ FALSE. In any 1-in-3 satisfying assignment of φ', T will be true and F will be false.

2. For $1 \le j \le n$, we create a new variable x_j' and add the clause $D_j = F \vee x_j \vee x_j'$. Note that in any 1-in-3 satisfying assignment of φ', x_j and x_j' will take on opposite values. Hence, effectively, x_j' is $\neg x_j$, but without using a not-sign.

3. For $1 \le i \le k$, we take C_i, and form C_i'. Replace every negative literal $\neg x$ in C_i with x'.

We leave it to the reader to show that φ' satisfied the conditions above. □

The problems 3SAT, 1-in-3SAT, and NAE-3SAT are important for proving problems NP-complete. When proving a graph problem is NP-hard, we often take a formula and make a graph out of it. When doing this, we need to create graphs that model variables and clauses. These graphs are often called ***gadgets***. If we start with a formula of a certain type (e.g., 3SAT), or with a formula where the goal is to satisfy it in a certain way (e.g., 1-in-3SAT), then creating gadgets is often easier.

We will also use the term "gadget" more generally to mean some object we construct to do a reduction. It has no formal definition.

Exercise 1.22. We define a notion of approximate 2-coloring. Let $0 < \varepsilon < 1$. The ***ε-imperfect 2-coloring problem*** is the following: The input is a graph $G = (V, E)$. Determine whether there exists a 2-coloring of V such that at most an ε fraction of the edges have endpoints that are the same color.

1. Show that there exists an $0 < \varepsilon < 1$ such that the ε-imperfect 2-coloring problem is NP-complete.
   Hint: Reduce Monotone NAE-3SAT to this problem (you can assume every clause has exactly 3 variables).

2. (This may be open) Determine for which ε the problem is in P and for which ε the problem is NP-complete.

1.2.5 Schaefer's Dichotomy Theorem

Every variant of SAT we looked at so far has either been in P or NP-complete. This is not a coincidence. Schaefer's Dichotomy Theorem says that, with the right setup, all versions of SAT are either NP-complete or in P.

We motivate the approach by looking at 1-in-3SAT. If a clause in an instance of 1-in-3SAT is $x_1 \vee x_2 \vee x_3$, then what we really want is

$$R_1(x_1, x_2, x_3) = (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3).$$

If a clause in an instance of 1-in-3SAT is $x_1 \vee x_2 \vee \neg x_3$, then what we really want is

$$R_2(x_1, x_2, x_3) = (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3).$$

One could write down $R_3, \ldots, R_8$ for the remaining cases. We view an instance of 1-in-3SAT as being given a conjunction of R_i's (repeats allowed) on 3-sets of variables. For example, the instances

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee \neg x_4)$$

can be rewritten, using the definition of R_1, R_2 above, as

$$R_1(x_1, x_2, x_3) \wedge R_2(x_1, x_3, x_4).$$

We call the R_i's ***relations***. More generally, a relation on m variables is just a truth table or formula on those literals. We will view variants of the SAT problem as a conjunction of relations. We give an absurd example.

Let $R_1(x_1, x_2, x_3, x_4)$ beTRUEif exactly 2 of the x_i's are TRUE.

Let $R_2(x_1, x_2, x_3, x_4)$ beTRUEif either 1 or 4 of the x_i's are TRUE.

Let $R_3(x_1, \ldots, x_{12})$ beTRUEif either 0 or 7 of the x_i's are TRUE.

Crazu SAT is a conjunction of R_1's, R_2's, and R_3's. We seek a way to satisfy all of the relations, though notice that this might not be a satisfying assignment. Is Crazu SAT in P? NP-complete? After we state and understand Schaefer's Dichotomy Theorem, this will be possible to determine. (We invented Crazu SAT as an example. We suspect it has never appeared in the literature before and will never appear in the literature again.)

Definition 1.23. A ***SAT-type problem*** is a set of relations $R_1, \ldots, R_k$. Each one has a fixed arity. We seek to satisfy all of the relations, though notice that this might not be a satisfying assignment. We will assume that none of the R_i are constant.

We now have a way of talking about many variants of SAT. Schaefer's Dichotomy Theorem [Sch78] will tell *exactly* which of these variants are in P and which are NP-complete. (For a modern proof see Chen's papers [Che06],[Che09].) It is remarkable that every variant defined in this way is one or the other.

Theorem 1.24. *Let $R_1, \ldots, R_k$ be relations. If any of the following occur, then the SAT-type problem $\{R_1, \ldots, R_k\}$ is in P. If not, then it is NP-complete.*

1. *For all $1 \le i \le k$, $R_i(\text{TRUE}, \ldots, \text{TRUE}) = \text{TRUE}$.*
2. *For all $1 \le i \le k$, $R_i(\text{TRUE}, , \ldots, \text{TRUE}) = \text{FALSE}$.*
3. *For all $1 \le i \le k$, $R_i(x_1, \ldots, x_m)$ is equivalent to a conjunction of clauses with 1 or 2 variables.*
4. *For all $1 \le i \le k$, $R_i(x_1, \ldots, x_m)$ is equivalent to a Horn clause.*
5. *For all $1 \le i \le k$, $R_i(x_1, \ldots, x_m)$ is equivalent to a dual-Horn clause.*
6. *For all $1 \le i \le k$, $R_i(x_1, \ldots, x_m)$ is equivalent to $x_1, \ldots, x_m$ satisfying a system of linear equations over mod 2.*

Given a set of relations $R_1, \ldots, R_k$, determining whether any of the 6 cases in Theorem 1.24 holds seems hard and probably is. We usually use this theorem by using some SAT-type problem that is NP-complete by the theorem, and then prove something else NP-complete by a reduction.

Exercise 1.25.

1. Determine whether Crazu SAT is in P or NP-complete.
2. Write a program that will, given a set of relations, determine whether the SAT-type problem they define is in P or NP-complete.

Note: Theorem 1.24 says that, up to polynomial-time reductions, there are (assuming P $\neq$ NP) two complexity classes that a SAT-type problem could be in, P or NP-complete. Allender et al. [ABI+09] showed that if a more refined reduction is used then the ones in P can be further differentiated. There end up being 6 classes total: 5 within P, and of course NP-complete.

1.2.6 A Dichotomy Theorem for Graph Formulas

Bodirsky & Pinsker [BP15] proved an analog of Schaefer's Theorem for graph formulas. We describe what they did.

Definition 1.26. A ***graph formula*** is a Boolean Formula where all of the literals are of the form $E(x, y)$ or $x = y$. A graph formula $\varphi(x_1, \ldots, x_n)$ is ***satisfiable*** if there exists a graph G and a set of vertices of G, $v_1, \ldots, v_n$ such that $\varphi(v_1, \ldots, v_n)$ is true in G.

Example 1.27.

1. $E(x,y) \wedge E(y,z) \wedge E(z,x)$ is satisfiable. Just take K_3.

2. $$\left(\bigwedge_{1\le i<j<\le 6} (x_i \neq x_j)\right) \bigwedge \left(\bigwedge_{1\le i<j<\le 6} [E(x_i,x_j) \rightarrow E(x_j,x_i)]\right) \bigwedge$$

$$\left(\bigwedge_{1\le i<jk<\le 6} \neg(E(x_i,x_j) \wedge E(x_i,x_k) \wedge E(x_i,x_j)) \wedge \neg(\neg E(x_i,x_j) \wedge \neg E(x_i,x_k) \wedge \neg E(x_i,x_j)\right)$$

The first part says that $x_1,\ldots,x_6$ are all different. The second part says that the graph restricted to $x_1,\ldots,x_6$ is symmetric. The third part says that the graph restricted to $x_1,\ldots,x_6$ has neither a clique of size 3 or an independent set of size 3. We leave it to the reader to show that there is no such graph, so this formula is not satisfiable.

Schaefer's Theorem classified types of SAT-problems as being either P or NP-complete. Bodirsky & Pinsker [BP15] did the same for types of graph formulas.

Let $\Psi = \{\psi_1,\ldots,\psi_n\}$ be a set of Boolean formulas. We define a problem Graph-SAT(Ψ).

Graph-SAT(Ψ) Let $\Psi = \{\psi_1,\ldots,\psi_n\}$ be a set of Boolean formulas. These are not the input. They are a parameter of the problem.
Instance: A graph formula of the form $\Phi = \varphi_1 \wedge \cdots \wedge \varphi_L$ where each φ_i is one of the ψ_j except that it may use variables other than those used in ψ_j.
Question: Is Φ satisfiable?

Example 1.28.

1. Ψ has the following two formulas:

   - $TRI(x_1,x_2,x_3) = E(x_1,x_2) \wedge E(x_2,x_3) \wedge E(x_2,x_1)$.
   - $SQUARE(y_1,y_2,y_3,y_4) = E(y_1,y_2) \wedge E(y_2,y_3) \wedge E(y_3,y_4) \wedge E(y_4,y_1)$.

   Consider the following instance:

$$\left(\bigwedge_{1\le i<j\le 4} x_i \neq x_j\right) \bigwedge TRI(x_1,x_2,x_3) \bigwedge SQUARE(x_1,x_2,x_3,x_4).$$

This is asking whether there is a graph which has four vertices that form a square and three of them also form a triangle. The answer is YES.

Consider the instance:

$$\left(\bigwedge_{1\le i<j\le 4} x_i \neq x_j\right) \neg TRI(x_1,x_2,x_3) \bigwedge TRI(x_1,x_3,x_5) \bigwedge SQUARE(x_1,x_{2,3},x_4).$$

We leave it to the reader to show that the answer is NO.

The main theorem of Bodirsky & Pinsker [BP15], which is an analog of Schaefer's Dichotomy Theorem, is as follows.

Theorem 1.29.

1. *For all Ψ the problem Graph-SAT(Ψ) is either NP-complete or in P.*

2. *The problem of, given Ψ determining of Graph-SAT(Ψ) is NP-complete or in P is decidable.*

1.3 How to Reduce from 3SAT

There are two general patterns for NP-hardness proofs based on reductions from 3SAT and its variations. (See Chapter 3 for reductions from Circuit SAT.) In both patterns, the idea is to represent the binary choice of each variable by a ***variable gadget***, represent the constraint of each clause by a ***clause gadget***, and connect these gadgets together via ***wire gadgets*** (though sometimes the wire gadget is trivial).

In a ***binary logic*** proof, a wire gadget can be solved in exactly two possible ways. One solution represents TRUE and the other represents FALSE. In this case, we can implement a variable gadget in terms of a wire gadget, as it involves the same kind of binary choice. To complete this implementation, we need a ***split gadget***, which guarantees that the three (or more) incident wires all carry the same value (all TRUE or all FALSE). Then a variable gadget attached to k clauses can be represented by a network of $O(k)$ split gadgets that produce k identical wires. If the SAT variation includes negative literals, we also need a ***NOT gadget*** that transforms a wire from FALSE to TRUE and vice versa, which we can add along the way to a clause. Sometimes we also need a ***terminator gadget*** that ends a wire without enforcing its value to be TRUE or F; this is useful, for example, when the split gadget produces more copies of a wire than needed.

In a ***dual-rail logic*** proof, a wire is replaced by two ***semiwires***. Each semiwire either holds the value TRUE or doesn't; in the latter case, the semiwire often isn't part of the solution at all. The choice of which semiwire holds the value TRUE corresponds to the Boolean value of the original wire or variable. Assuming a split gadget, a variable gadget's role is to force exactly one (or at most one) of two semiwires to be TRUE. Often the variable gadget and split gadgets are integrated into a single variable gadget that produces j semiwires representing when the variable is TRUE and k semiwires representing when the variable is FALSE, where j and k are the number of clauses using this variable in positive and negative literals, respectively.

The two styles of proof have a lot in common, and the distinction can be subtle. You are encouraged to categorize the various reductions described in this book as one or the other.

1.4 2-Colorable Perfect Matching is NP-Complete

2-COL Perfect Matching
Instance: A graph G.
Question: Is there a 2-coloring of the vertices such that every vertex has exactly 1 neighbor of the same color? Figure 1.2 gives an example of a graph with a 2-colorable perfect matching.

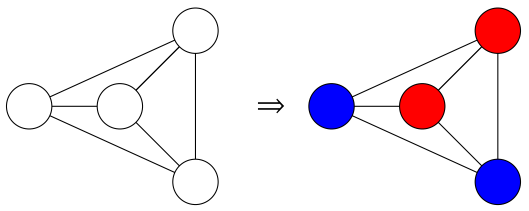


Figure 1.2: A graph that has a 2-Colorable Perfect Matching

Theorem 1.30. *(Schaefer [Sch78]) 2-COL* Perfect Matching *is NP-complete. It remains NP-complete when restricted to planar 3-regular graphs.*

Proof. We show Monotone NAE-3SAT $\le_p$ 2-COL Perfect Matching. Given φ, we find a graph G such that $\varphi \in$ Monotone NAE-3SAT if and only if $G \in$ 2-COL Perfect Matching. As we describe the reduction look at Figure 1.3 for the gadgets we use. We will color the vertices of the graph TRUE and FALSE. Since the goal is to have no clause have all TRUE's or all FALSE's, TRUE and FALSE are symmetric here.

1. For every clause $X \wedge Y \wedge Z$, we have the gadget in 1a. This gadget makes sure that in any 2-coloring satisfying the condition, X, Y, Z cannot all be the same color. Hence, any 2-coloring satisfying the condition will be a Not-All-Equal truth assignment.

2. We will need a variable X to appear several places. Gadget 1b shows how to have two copies of x that both receive the same color.

3. Gadget 1c shows the graph that results if the formula is $(x \vee x \vee y) \wedge (y \wedge z \wedge u)$.

We leave it to the reader to prove that the reduction works.

We now want to make the graph planar and 3-regular. How to make it planar? We introduce the notion of a *crossover gadget*, which we will use both in this chapter and in Chapter 2.

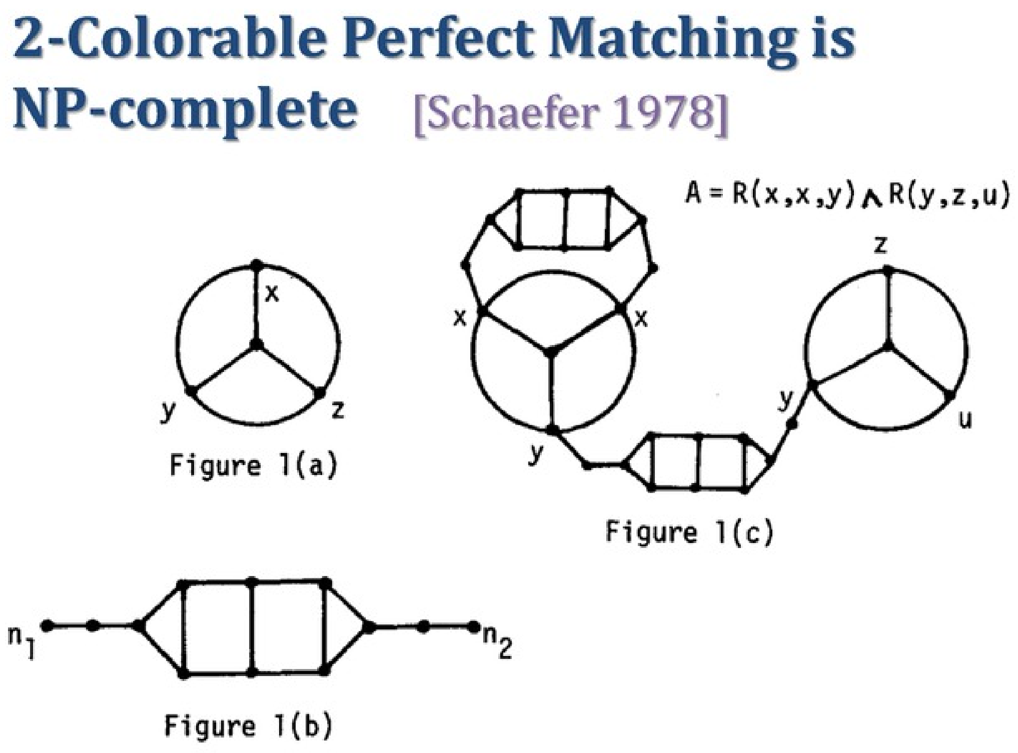
Definition 1.31. Let A be an NP-complete set of graphs. We want to show that the problem is still NP-complete when restricted to planar graphs. We want to map G to G' such that G is in A if and only if G' (which is planar) is in A. We obtain G' from G by looking at each crossing and replacing it with a gadget that does not affect the graph's membership in A. We call this a ***crossover gadget***.

□

We leave it as an exercise to create the following:

1. A crossover gadget to make the proof work for planar graphs.

2. A gadget to split high degree nodes into lower degree ones, possibly with the copy gadget (this idea is due to Adam Hesterberg).



2-Colorable Perfect Matching is NP-complete [Schaefer 1978]

Figure 1.3: Gadgets to Prove 2-COL Perfect Matching NP-Complete

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

Figure 1.4: The SEND MORE MONEY Cryptarithm.

1.5 Cryptarithm is NP-Complete

Cryptarithms are classic puzzles involving arithmetic on words. Figure 1.4 gives an example.

The goal is to replace each letter with a digit, no digit is assigned to two different letter, such that the resulting sum works out.

Here is how one might begin solving the cryptarithm in Figure 1.4:

1. A carry can be at most 1. Hence $M = 1$.

2. Since the left-most column is a carry and $M = 1$, the digit O must be either 0 or 1. Since $O \neq M$, we have $O = 0$, which is convenient.

3. The E, O, N column can't contribute a carry: assume that it did. Then since $O = 0$ we would have that $N + R$ contributes a carry, and $E + 1$ results in a carry. Then $E = 9$ and $N = 0$. But this is impossible since $O = 0$.

4. Since $M = 1$, $O = 0$, and the E, O, N column does not contribute a carry, $S = 9$.

5. To recap, we have $M = 1$, $O = 0$, $S = 9$.

6. Look at the E, O, N column. Since $O = 0$, this column does not contribute a carry, and $E \neq N$ (by the rules of the puzzle), the (N, R, E) column has to contribute a carry. So we have $E + 1 \equiv N \pmod{10}$. Since $O = 0$, $M = 1$, $S = 9$, and all letters map to different digits we have $E, N \notin \{0, 1, 9\}$. We use this and $E + 1 \equiv N \pmod{10}$ to further cut down on what (E, N) can be. Since $N \neq 9$, we get $E \neq 8$. Since $E \neq 1$, we get $N \neq 2$. In summary, $E \notin \{0, 1, 8, 9\}$ and $N \notin \{0, 1, 2, 9\}$. Using this restriction on E, N and also $E + 1 \equiv N \pmod{10}$, we get:

$$(E, N) \in \{(2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8)\}.$$

We stop here; however, notice that we may end up encountering many possibilities. If you work them through, then you will find that the answer is unique and is:

$$S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2.$$

Figure 1.5 shows how to check the answer.

$$\begin{array}{ccccc} & 9 & 5 & 6 & 7 \\ + & 1 & 0 & 8 & 5 \\ \hline 1 & 0 & 6 & 5 & 2 \end{array}$$

Figure 1.5: Solution to The SEND MORE MONEY Cryptarithm.

Is there a trick to these puzzles so that they can always be solved fast and avoid having too many cases? Likely no: David Eppstein [Epp87] showed

$$\text{3SAT} \le_p \text{Cryptarithms},$$

hence Cryptarithms is NP-complete. We will show

$$\text{Monotone 1-in-3SAT} \le_p \text{Cryptarithms}.$$

By Theorem 1.21 we have that Cryptarithms is NP-complete. Our reduction is easier than that of Eppstein; however, we need that Monotone 1-in-3SAT is NP-complete.

We first need to define the problem rigorously.

> Cryptarithms
> *Instance:*
>
> 1. $B, m \in \mathbb{N}$. Let Σ be an alphabet of B letters.
> 2. $x_0, \dots, x_{m-1}$. Each $x_i \in \Sigma$.
> 3. $y_0, \dots, y_{m-1}$. Each $y_i \in \Sigma$.
> 4. $z_0, \dots, z_m$. Each $z_i \in \Sigma$. The symbol z_m is optional.
>
> *Question:* Is there an injection of Σ into $\{0, \dots, B-1\}$ so that the arithmetic statement in Figure 1.6 is true (in base B)?

$$\begin{array}{cccc} & x_{m-1} & \cdots & x_0 \\ + & y_{m-1} & \cdots & y_0 \\ \hline z_m & z_{m-1} & \cdots & z_0 \end{array}$$

Figure 1.6: Solution to Cryptarithms

Theorem 1.32. *Cryptarithms is NP-complete.*

Proof. Given $\varphi = C_1 \wedge \cdots \wedge C_k$, a formula where every literal is positive, we want to create an instance J of Cryptarithms such that the following are equivalent:

- There exists an assignment that satisfies exactly one literal per clause.
- There exists a solution to J.

Let n be the number of variables in φ. As shown, k is the number of clauses.

We will determine the base B, and the length of the numbers m, later.

1) *Constants* We need two letters that we suggestively call 0 and 1 that we force to map to 0 and 1. We use the columns in Figure 1.7.

It is easy to see that the columns in Figure 1.7 force (1) the letter "0" to have the value 0, and (2) the letter "1" to have the value 1. (You will need to use that carries are either 0 or 1.)

To establish the constants 0 and 1 we need (1) the 4 letters $0, 1, p, q$, and (2) 3 columns.

2) *Variables* Let v be a variable in φ.

v is considered true if $v \equiv 1 \pmod 4$ and false if $v \equiv 0 \pmod 4$

$$\begin{array}{ccc} 0 & p & 0 \\ 0 & p & 0 \\ \hline 1 & q & 0 \end{array}$$

Figure 1.7: The Constants Gadget.

Hence we need to ensure that $v \equiv 0 \pmod 4$ or $v \equiv 1 \pmod 4$.
This is accomplished through a string of intermediate sums:

$$\begin{aligned} b &= 2a \\ 2c &= d + C \qquad\qquad C = \text{carry}(c+c) \in \{0,1\} \\ v &= 2b + C \\ &= 4a + C \equiv C \pmod 4 \end{aligned}$$

(Note that a, b, c, d, v are used in the puzzle, whereas C is not.)

$$\begin{array}{cccccc} 0 & b & c & 0 & a & 0 \\ 0 & b & c & 0 & a & 0 \\ \hline 0 & v & d & 0 & b & 0 \end{array}$$

Figure 1.8: The Variable Gadget.

We do not have to consider $\overline{v}$ since our formula has all positive literals.

Each variable v needs (1) the 5 letters a, b, c, d, v and (2) 6 columns. Since there are are n variables, we need $5n$ letters and $6n$ columns for this part.

3) *Clauses* Let C be the clause $x \vee y \vee z$ (x, y, z need not be distinct). Let x, y, z be the letters corresponding to the variables in the cryptarithm. Since our reduction is from MONOTONE 1-IN-3SAT $\leq_p$ CRYPTARITHMS, we need that $x + y + z \equiv 1 \pmod 4$.

We first need to have a number d that can be anything $\equiv 1 \pmod 4$. This is accomplished by the following equations:

$$\begin{aligned} b &= 2a \\ c &= 2b \\ &= 4a \\ d &= c + 1 \\ &= 4a + 1 \end{aligned}$$

We then need to have that $x + y + z = d$. We need an intermediary variable for $x + y$ that we call I (for Intermediate).

$$x + y = I$$
$$I + z = d$$

The result is Figure 1.9.

$$\begin{array}{ccccccccccc}
0 & I & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\
0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\
\hline
0 & d & 0 & I & 0 & d & 0 & c & 0 & b & 0
\end{array}$$

Figure 1.9: The Clause Gadget

Each clause C needs (1) the 5 letters a, b, c, d, I, and (2) 11 columns. Since there are k clauses, we need $5k$ letters and $11k$ columns.

The construction is completed.

The final instance of Cryptarithms uses $5n+5k+4$ letters and $6n+11k+3$ columns. However, we cannot just take $B = 5n + 5k + 4$. We need enough numbers so that, for example, (looking at the clause gadget), we don't have $b + b$ is the same as $x + y$. We revisit the issue of B soon.

Clearly, a solution to the cryptarithm J gives a solution to the Monotone 1-in-3SAT problem φ. The other direction is less clear. Assume we have a solution to the Monotone 1-in-3SAT problem φ. This assigns true or false to each of the variables $v_1, \ldots, v_n$. We translate this to an assignment of the letters $v_1, \ldots, v_n$ to numbers. We do this inductively. Assume letters $v_1, \ldots, v_{i-1}$ and some of the other letters have been assigned.

1. If variable v_i is true, then we will assign letter v_i to a number $\equiv 1 \pmod 4$.

2. Assign to the letter v_i the least number that has the right congruence mod 4, has not been assigned to any other letter, and does not cause any letter to be assigned to an already-used number. This arises (1) with the variable gadgets, since once you assign v, you need to assign a, b, c, d, and (2) with any clause gadget that contains v, where the other variables in it have been assigned.

We leave it to the reader to determine how large B must be to accommodate all these numbers.

- B will be large enough so that many numbers will not be used. Hence, there will be letters that do not map to any number. Note that in the SEND+MORE=MONEY puzzle, many letters (e.g. Z) do not map to any number.

- B will be bounded by a polynomial in k, n. Hence Cryptarithms is strongly NP-complete.

□

Exercise 1.33. Do a direct reduction to show that 3SAT $\leq_p$ Cryptarithms.

Exercise 1.34.

1. Write an algorithm for CRYPTARITHMS, and analyze its run time.
2. How fast is your algorithm when B is a constant?

Exercise 1.35. Read Franck Dernoncourt's paper [Der14] on the NP-completeness of the Truck-Mania problem. The proof uses a reduction of 3SAT. Rewrite the reduction in your own words.

Exercise 1.36. Consider the following puzzle. The board is a grid. Initially some spaces have a red stone, some have a blue stone, and some are blank. The goal is to remove stones so that for every column: (1) there is at least one stone, and (2) all of the stones in it are the same color. Show that the problem of, given an initial position, can the player win, is NP-complete. Use a reduction from 3SAT.

1.6 GRID COLORING is NP-Complete

Everything in this section is from Apon et al. [AGLar].

Notation 1.37.

1. If $x \in \mathbb{N}$ then $[x]$ denotes the set $\{1,\ldots,x\}$. $G_{N,M}$ is the set $[N] \times [M]$.
2. If X is a set and $k \in \mathbb{N}$ then $\binom{X}{k}$ is the set of all size-k subsets of X.

Definition 1.38. A ***rectangle of*** $\boldsymbol{G_{N,M}}$ is a subset of the form $\{(a,b),(a+c_1,b),(a+c_1,b+c_2),(a,b+c_2)\}$ for some $a,b,c_1,c_2 \in \mathbb{N}$ with $c_1,c_2 \geq 1$. Note that we are only looking at the four corners of the rectangle—nothing else. A grid $G_{N,M}$ is ***c-colorable*** if there is a function $\chi : G_{N,M} \to [c]$ such that there are no rectangles with all four corners the same color. In other words, a grid $G_{N,M}$ is c-colorable if there is a function $\chi : G_{N,M} \to [c]$ such that there are no monochromatic rectangles. (If we are ever dealing with colorings that may have rectangles we will use the term ***proper coloring*** to denote those colorings that do not have monochromatic rectangles.)

Fenner et al. [FGGP12] explored the following problem:

Which grids are c-colorable for a given fixed c?

We state some of their results.

1. For all $c \geq 2$, $G_{c+1,\binom{c+1}{2}+1}$ is not c-colorable
2. For all c there exists a finite number of grids such that $G_{N,M}$ is c-colorable if and only if it fails to contain any one of those grids. This set of grids is called ***the obstruction set*** and is denoted by OBS_c.
3. $\mathrm{OBS}_2 = \{G_{3,7}, G_{5,5}, G_{7,3}\}$. This was obtained without the aid of a computer program.
4. $\mathrm{OBS}_3 = \{G_{19,4}, G_{16,5}, G_{13,7}, G_{11,10}, G_{10,11}, G_{7,13}, G_{5,16}, G_{4,19}\}$. A computer aided search was used to find a 3-coloring of $G_{10,10}$.

5. $$\mathrm{OBS}_4 = \{G_{41,5}, G_{31,6}, G_{29,7}, G_{25,9}, G_{23,10}, G_{22,11}, G_{21,13}, G_{19,17}\} \bigcup$$

$$\{G_{17,19}, G_{13,21}, G_{11,22}, G_{10,23}, G_{9,25}, G_{7,29}, G_{6,31}, G_{5,41}\}$$

   The authors were stuck for a long time trying to find 4-colorings of $G_{17,17}$, $G_{17,18}$, $G_{18,18}$, $G_{12,21}$, and $G_{10,22}$ (we omit the symmetric cases which follow automatically, i.e., if there is a 4-coloring of $G_{22,10}$ then there is one for and $G_{10,22}$). They believed these were all 4-colorable. William Gasarch put a bounty of $17^2 = 289$ dollars for a 4-coloring of $G_{17,17}$ and posted this challenge to ComplexityBlog [Gas09a]. Bernd Steinbach and Christian Posthoff found 4-colorings of $G_{17,17}$, $G_{18,18}$, and $G_{12,21}$ and received the reward. Brad Larsen found a 4-coloring of $G_{22,10}$. These results completed the search for OBS_4. Brad Larsen posted the 4-coloring saying he used a SAT-solver but he did not elaborate. Steinbach and Posthoff published their results and their methods. In brief, they used a very deep analysis that allowed for a strong reduction of the problem, and then used the Universal SAT-Solver clasp. See their articles [SP12a, SP12b, SP12c, SP13a, SP13b, SP15] and a book edited by Steinbach [Ste14] that has several chapters explaining how they found a 4-coloring of $G_{12,21}$ in detail. These results completed the search for OBS_4.

6. Finding OBS_5 seems to be beyond current technology.

The difficulty of 4-coloring $G_{17,17}$ and pinning down OBS_5 raise the following question: is the problem of grid coloring hard? We define the ***Grid Coloring Extension Problem (GCE)*** as a way to get at the issue. After we show that GCE is NP-complete we discuss if this really does get at the issue.

Definition 1.39. Let $N, M, c \in \mathbb{N}$.

1. A ***partial mapping χ of $G_{N,M}$ to $[c]$*** is a mapping of a subset of $G_{N,M}$ to $[c]$. See Figure 1.10 for an example.

2. If χ is a partial mapping of $G_{N,M}$ to $[c]$ then χ' is ***an extension of χ*** if χ' is a partial mapping of $G_{N,M}$ to $[c]$ which (1) is defined on every cell that χ is defined, (2) agrees with χ on those cells, and (3) may be defined on more cells.

3. A ***total mapping χ of $G_{N,M}$ to $[c]$*** is a mapping of $G_{N,M}$ to $[c]$. This would normally just be called a mapping, but we use the term total to distinguish it from a partial mapping.

Definition 1.40. Let $c, N, M \in \mathbb{N}$. A partial coloring χ of $G_{N,M}$ to $[c]$ is ***extendable to a c-coloring*** if there is an extension of χ to a total mapping which is a c-coloring of $G_{N,M}$. We will use the term ***extendable*** if the c is understood.

GRID COLORING EXTENSION (GCE)
Instance: $N, M, c \in \mathbb{N}$ and a partial c-coloring χ of $G_{N,N}$.
Question: Is χ extendable?

R								
R								
B		R						
R				B				
R					B			
R								
R	R	R	R	R	R	R	R	R

Figure 1.10: Example of a Partial Coloring

We show that GCE is NP-complete. This result may explain why the original 17×17 challenge was so difficult. Then again—it may not. We discuss this further after we prove GCE is NP-complete.

Before showing that GCE is NP-complete we briefly discuss its status within NP.

Clearly GCE $\in$ NTIME$(O((NM)^4))$. Kreveld & De Berg [vKdB91] proved, in our notation, the following lemma.

Lemma 1.41. There is an algorithm that will, given a set of cells $P \subseteq G_{N,M}$, determine whether P contains a rectangle, in time $O((NM)^{3/2})$.

Exercise 1.42. Using Lemma 1.41 show that GCE $\in$ NTIME$(O(cMN)^{3/2})$. (You can try to do it yourself or look up the proof in Apon et al. [AGLar].)

Theorem 1.43. *GCE is NP-complete.*

Proof. Clearly GCE $\in$ NP.

We give a reduction of 3SAT to GCE. The input will be a 3CNF formula

$$\varphi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$$

with n free variables and m clauses. The output will be (N, M, c, χ) where

- $N, M, c \in \mathbb{N}$,
- χ is a a partial c-coloring of $G_{N,M}$, and
- $\varphi \in$ 3SAT if and only if $(N, M, c, \chi) \in$ GCE.

We can assume that φ never has a clauses that contains either (1) the same literal twice, or (2) a variable and its negation. Condition (1) will be needed in Part III of the construction. Condition (2) will be needed in the proof of Claim 4.

The reduction we show you *does not quite work!*; however, it has most of the ideas needed. There is a problem with it that will be revealed when we try to prove Claim 4. During that proof we will see what goes wrong and modify the construction so that Claim 4 is true.

Visualize the full grid as a core subgrid with additional entries to the left and below. These additional entries are there to enforce that some colors in the core grid occur only once.

Conventions

$(2,4)$							
$(2,4)$							
$(2,4)$							
T		$(2,4)$					
$(2,4)$							
$(2,4)$							
$(2,4)$							
$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$

Figure 1.11: Cell $(2,4)$ is Colored $(2,4)$ and Nothing Else Can Be

1. Throughout this proof "extension" means "an extension that uses the colors TRUE, FALSE on some of the uncolored cells and does not have a monochromatic rectangle". It may or may not extend to the entire grid.

2. In our figures we will have literals labeling some of the rows and clauses labeling some of the columns. These are not part of the construction. The literals and clauses are visual aids. We may refer to "row x_7" or "column C_3".

3. In our figures we will have double lines to separate things. These lines are not part of the construction. These are visual aids.

4. The colors will be TRUE, FALSE, and some of the $(i,j) \in G_{N,M}$. Many of the cells that are in the core grid will be colored (i,j) where that is their position in the core grid. In the figures we will denote the color by D for distinct. Part I of the construction will make sure that no other cell in the core grid can have that color.

The reduction is in four parts. We will mainly construct a core grid which will be $2n+m$ by $2n+2m+1$ (when we later modify the construction the core grid will be bigger though still linear in m,n).

In all figures the left bottom cell of the core grid is indexed $(1,1)$.

Part I: Forcing a color to appear only once in the core grid.

For (i,j) in the core grid we will often set $\chi(i,j)$ to (i,j) and then never reuse (i,j) in the core grid. By doing this, we make having a monochromatic rectangle rare and have control over when that happens.

We show how to color the cells that are not in the core grid to achieve this. Part I will be the final step in the reduction since we need to know the size of the grid before we can apply it; however, we show Part I first.

Say we want the cell $(2,4)$ in the core grid to be colored $(2,4)$ and we do not want this color appearing anywhere else in the core grid. We can do the following: add a column of $(2,4)$'s to the left end (with one exception) and a row of $(2,4)$'s at the bottom. See Figure 1.11.

It is easy to see that any extension of the coloring of the grid in Figure 1.11 the only cells that can have the color $(2,4)$ are those shown to already have that color. It is also easy to see that the

$(5,3)$	$(2,4)$							
$(5,3)$	$(2,4)$							
$(5,3)$	$(2,4)$							
$(5,3)$	T	$(2,4)$						
T	$(2,4)$				$(5,3)$			
$(5,3)$	$(2,4)$							
$(5,3)$	$(2,4)$							
$(5,3)$	$(2,4)$	$(2,4)(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$	$(2,4)$
$(5,3)$	$(5,3)$	$(5,3)(5,3)$	$(5,3)$	$(5,3)$	$(5,3)$	$(5,3)$	$(5,3)$	$(5,3)$

Figure 1.12: $(2,4)$ and $(5,3)$

color T we have will not help to create any monochromatic rectangles since there are no other T's in its column. The T we are using *is* the same T that will later mean "true". We could have used F. We do not want to use new colors since we would have no control over where else they could be used.

What if some other cell needs to have a unique color? Let's say we also want to color cell $(5,3)$ in the core grid with $(5,3)$ and do not want to color anything else in the core grid $(5,3)$. Then we use the grid in Figure 1.12

It is easy that in any extension of the coloring of the grid in Figure 1.12 the only cells that can have the color $(2,4)$ or $(5,3)$ are those shown to already have those colors.

For the rest of the construction we will only show the core grid. If we denote a color as D (short for "Distinct") in the cell (i,j) then this means that

1. cell (i,j) is color (i,j), and
2. we have used the above gadget to make sure that (i,j) does not occur as a color in any other cell of the core grid.

Note that when we have D in the $(2,4)$ cell and in the $(5,3)$ cell, they denote different colors.

Part II: Forcing $(x,\overline{x})$ to be colored (TRUE, FALSE) or (FALSE, TRUE).

The first column of the core grid will have $2n$ blanks and then m D's. We will use the m D's later. Figure 1.13 illustrates what we do in the $n=4$ case.

We will arrange things so that the color of the blanks in Figure 1.13 will all be either T or F. We refer to the color of the cell next to x_i as ***the color of x_i***. Same for $\overline{x}_i$.

It is easy to see that in any coloring of Figure 1.13:

- If x_i is colored T then $\overline{x}_i$ is colored F.
- If x_i is colored F then $\overline{x}_i$ is colored T.

We leave it to the reader to generalize Figure 1.13 to n variables.

	D	D	D	D	D	D	D	D	D
	D	D	D	D	D	D	D	D	D
	D	D	D	D	D	D	D	D	D
	D	D	D	D	D	D	D	D	D
$\overline{x}_4$		D	D	D	D	D	D	T	F
x_4		D	D	D	D	D	D	T	F
$\overline{x}_3$		D	D	D	D	T	F	D	D
x_3		D	D	D	D	T	F	D	D
$\overline{x}_2$		D	D	T	F	D	D	D	D
x_2		D	D	T	F	D	D	D	D
$\overline{x}_1$		T	F	D	D	D	D	D	D
x_1		T	F	D	D	D	D	D	D

Figure 1.13: Literal Gadget with four Variables

We will call the leftmost column, which is blank, ***the literal column***.

This part is what will need to be adjusted. It will turn out that we need several copies of each literal. During the proof of Claim 4 we will see why this is true and how to achieve it.

Part III: Forcing the coloring to satisfy a single clause

For each clause $C = L_1 \vee L_2 \vee L_3$ we will use two columns. These columns will be called ***clause columns***.

Before saying what we put into the columns, Figure 1.14 is the initial setup in the case of $n = 4$ and $m = 4$. We leave it to the reader to generalize to n, m. The X's in Figure 1.14 will be replaced by T's, F's, or blanks in the next step.

Let $C = L_1 \vee L_2 \vee L_3$. Figure 1.15 illustrates how we color, or leave blank, the cells in the C-column.

Note that since we never have the same literal appearing twice in a clause, the construction of the Clause Gadget can be carried out.

We redraw Figure 1.15 as Figure 1.16 for ease of use. We refer to the partial coloring in Figure 1.16 as χ.

Claim 1: Let χ denote the partial coloring in Figure 1.16. If χ' is an extension of χ then χ' cannot have the L_1, L_2, L_3 cells all colored F.

Proof of Claim 1:

Assume, by way of contradiction, that L_1, L_2, L_3 are all colored F. Then we have the partial coloring in Figure 1.17

The reader can verify that if the two blank cells of Figure 1.17 are colored TT, TF, FT, or FF, there will be a monochromatic rectangle.

End of Proof of Claim 1

Claim 2: Let χ' be an extension of the coloring in Figure 1.16 that colors L_1, L_2, L_3 but not the

											C_1	C_1	C_2	C_2	C_3	C_3	C_4	C_4
	D		D	D	D	D	D	D	D	D	D	D	D	D	D	D	T	T
	D		D	D	D	D	D	D	D	D	D	D	D	D	T	T	D	D
	D		D	D	D	D	D	D	D	D	D	D	T	T	D	D	D	D
	D		D	D	D	D	D	D	D	D	T	T	D	D	D	D	D	D
$\overline{x}_4$			D	D	D	D	D	D	T	F	X	X	X	X	X	X	X	X
x_4			D	D	D	D	D	D	T	F	X	X	X	X	X	X	X	X
$\overline{x}_3$			D	D	D	D	T	F	D	D	X	X	X	X	X	X	X	X
x_3			D	D	D	D	T	F	D	D	X	X	X	X	X	X	X	X
$\overline{x}_2$			D	D	T	F	D	D	D	D	X	X	X	X	X	X	X	X
x_2			D	D	T	F	D	D	D	D	X	X	X	X	X	X	X	X
$\overline{x}_1$			T	F	D	D	D	D	D	D	X	X	X	X	X	X	X	X
x_1			T	F	D	D	D	D	D	D	X	X	X	X	X	X	X	X

Figure 1.14: Clause Set Up

		$\cdots$	C	C	$\cdots$
	D	$\cdots$	T	T	$\cdots$
	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$
L_3		$\cdots$	D	F	$\cdots$
	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$
L_2		$\cdots$			$\cdots$
	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$
L_1		$\cdots$	F	D	$\cdots$
	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$

Figure 1.15: The Clause Gadget

		C	C
	D	T	T
L_3		D	F
L_2			
L_1		F	D

Figure 1.16: The Clause Gadget—Easier to Work With

		C	C
	D	T	T
L_3	F	D	F
L_2	F		
L_1	F	F	D

Figure 1.17: L_1, L_2, L_3 All Set to F

other two blank cells. Assume that χ' colors L_1, L_2, L_3 anything except FALSE, FALSE, FALSE. Then χ' can be extended to color the two blank cells.

Proof of Claim 2

There are seven cases based on (L_1, L_2, L_3) being labeled FFT, FTF, FTT, TFF, TTF, TFT, TTT. For each one we give a coloring of the remaining two blank cells so that no monochromatic rectangle is formed.

Case 1

		C	C
	D	T	T
L_3	F	D	F
L_2	F	F	T
L_1	T	F	D

Case 2

		C	C
	D	T	T
L_3	F	D	F
L_2	T	T	F
L_1	F	F	D

Case 3

		C	C
	D	T	T
L_3	F	D	F
L_2	T	T	F
L_1	T	F	D

Case 4

		C	C
	D	T	T
L_3	T	D	F
L_2	F	T	F
L_1	F	F	D

Case 5

		C	C
	D	T	T
L_3	T	D	F
L_2	F	T	F
L_1	T	F	D

Case 6

		C	C
	D	T	T
L_3	T	D	F
L_2	T	T	F
L_1	F	F	D

Case 7

		C	C
	D	T	T
L_3	T	D	F
L_2	T	T	F
L_1	T	F	D

End of Proof of Claim 2

Part IV: Putting it all together

Recall that $\varphi(x_1, \dots, x_n) = C_1 \wedge \cdots \wedge C_m$ is a 3CNF formula. We first define the core grid and later define the entire grid and N, M, c. The core grid will have $2n + m$ rows and $2m + 2n + 1$ columns (when we later modify the construction the core grid will be bigger though still linear in m, n). The $2n$ left-most columns are partially colored, and labeled with literals, as described in Part II. The m top-most rows are colored, and labeled with clauses, as described in Part III. The rest of the core grid is colored as described in Part III.

The core grid is now complete. For every (i, j) that is colored (i, j), we perform the method in Part I to make sure that (i, j) is the only cell with color (i, j). Let the number of such (i, j) be E. The number of colors c is $E + 2$. This will force everything else to be colored T or F. Note that $E = \Theta(NM)$.

In Figure 1.18 we present the instance of GCE obtained if the original formula is

$$(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee x_4) \vee (\overline{x}_2 \vee x_3 \vee x_4).$$

											C_1	C_1	C_2	C_2	C_3	C_3
	D		D	D	D	D	D	D	D	D	D	D	D	D	T	T
	D		D	D	D	D	D	D	D	D	D	D	T	T	D	D
	D		D	D	D	D	D	D	D	D	T	T	D	D	D	D
$\overline{x}_4$			D	D	D	D	D	D	T	F	D	D	D	D	D	D
x_4			D	D	D	D	D	D	T	F	D	D	D	F	D	F
$\overline{x}_3$			D	D	D	D	T	F	D	D	D	F	D	D	D	D
x_3			D	D	D	D	T	F	D	D	D	D	D	D		
$\overline{x}_2$			D	D	T	F	D	D	D	D	D	D	D	D	F	D
x_2			D	D	T	F	D	D	D	D					D	D
$\overline{x}_1$			T	F	D	D	D	D	D	D	D	D	D	D	D	D
x_1			T	F	D	D	D	D	D	D	F	D	F	D	D	D

Figure 1.18: Example with $(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee x_4)$

Claim 3: Let $\varphi(x_1, \ldots, x_n)$ be a 3CNF formula. Let (N, M, c, χ) be the result of the reduction described above. If $(N, M, c, \chi) \in$ GCE then $\varphi \in$ 3SAT.

Proof of Claim 3

Assume that $(N, M, c, \chi) \in$ GCE. According to the construction in Part II the first column gives a valid truth assignment for $x_1, \ldots, x_n$ (and hence also for $\overline{x}_1, \ldots, \overline{x_n}$). By Claim 1, for every clause $C = L_1 \vee L_2 \vee L_3$ this truth assignment cannot assign L_1, L_2 and L_3 all to F. Hence this is a satisfying assignment, so $\varphi \in$ 3SAT.

End of Proof of Claim 3

Claim 4 (which is false): Let $\varphi(x_1, \ldots, x_n)$ be a 3CNF formula. Let (N, M, c, χ) be the result of the reduction described above. If $\varphi \in$ 3SAT then $(N, M, c, \chi) \in$ GCE.

Proof of Claim 4 (which will fail)

Assume $\varphi \in$ 3SAT. Let $(b_1, \ldots, b_n)$ be a satisfying truth assignment where, for $1 \leq i \leq n$, $b_i \in \{\text{TRUE}, \text{FALSE}\}$. We use this to obtain a coloring of $G_{N,M}$ that is an extension of χ.

Color the literal column in the obvious way: the entry labeled with literal L is labeled the truth assignment of L. We now show how we try to color the blank cells in the clause columns.

Let $C = L_1 \vee L_2 \vee L_3$ be a clause. The part of the grid associated to it is in Figure 1.15.

	$\cdots$	D	D	$\cdots$	C	$\cdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	D	$\vdots$
$\overline{x}$	$\cdots$	T	F	$\cdots$	F	$\cdots$
x	$\cdots$	T	F	$\cdots$	F	$\cdots$

Figure 1.19: Case 2 of Claim 4

The literal column we have already colored. Since the assignment was satisfying at least one of L_1, L_2, L_3 was set to T. We use Claim 2 to extend the coloring to the blank cells. This forms a grid coloring. We try to prove this coloring is proper.

Assume, by way of contradiction, that there is a monochromatic rectangle which we call R.

Case 1 There is a clause C such that R uses the two T's associated to C. The only way these T's can be involved in a monochromatic rectangle is if the two blank cells associated to C are colored T. By the 7 cases in Claim 2 this cannot occur.

Case 2 There is a variable x such that R uses the two T's or two F's associated to x. Figure 1.19 shows what this looks like (we only include the relevant parts). We assume x is the first variable in C (the other cases are similar or cannot occur).

No clause-column has two T's in it, so R must be colored F. The only way there can be two F's in the literal column is if they are associated to a literal and its negation, as in Figure 1.19. However, the only way that configuration can happen is if x and $\overline{x}$ are in the same clause. This cannot happen since φ has no clauses with both a variable and its negation in it.

Case 3 R uses the literal column and one of the clause columns. By Claim 3 R is not monochromatic.

Case 4 The only case left is if R uses two clause columns. *This can occur!* This is where the construction fails! We give an example. Recall that Figure 1.18 is the instance of GCE from the formula

$$(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee x_4).$$

Lets say we take the satisfying truth assignment

$$x_1 = \text{TRUE}, x_2 = \text{FALSE}, x_3 = \text{TRUE}, x_4 = \text{FALSE}.$$

If we put these in the literal column and use the proof of Claim 2 to color the blank cells in the clause columns, the result is the coloring of he entire grid seen in Figure 1.20. The boldface colors are the ones the ones caused by the truth assignment. The asterisks show a monochromatic rectangle. Hence the construction produces a non-proper coloring and is incorrect.

End of the Proof of Claim 4 (that failed)

The way to avoid Case 4 is if we had *several* copies of each literal so that if two clauses use the same literal, they will use different copies of it. How many? The number of copies of literal L has to be at least the number of clauses that L appears in. It will be convenient to have the number of copies of L and of $\overline{L}$ be the same. Hence if x appears m_1 times and $\overline{x}$ appears m_2 times we will have both appear $\max\{m_1, m_2\}$ times.

Rather than give the general construction, we do an example with the case that gave us trouble before:

$$(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee x_4).$$

										C_1	C_1	C_2	C_2	C_3	C_3
	D	D	D	D	D	D	D	D	D	D	D	D	D	T	T
	D	D	D	D	D	D	D	D	D	D	D	T	T	D	D
	D	D	D	D	D	D	D	D	D	T	T	D	D	D	D
$\overline{x}_4$	$\boldsymbol{T}$	D	D	D	D	D	D	T	F	D	D	D	D	D	D
x_4	$\boldsymbol{F}$	D	D	D	D	D	D	T	F	D	D	D	F	D	F
$\overline{x}_3$	$\boldsymbol{F}$	D	D	D	D	T	F	D	D	D	F	D	D	D	D
x_3	$\boldsymbol{T}$	D	D	D	D	T	F	D	D	D	D	D	D	$\boldsymbol{T}$	$\boldsymbol{F}$
$\overline{x}_2$	$\boldsymbol{T}$	D	D	T	F	D	D	D	D	D	D	D	D	F	D
x_2	$\boldsymbol{F}$	D	D	T	F	D	D	D	D	$\boldsymbol{F}*$	$\boldsymbol{T}$	$\boldsymbol{F}*$	$\boldsymbol{T}$	D	D
$\overline{x}_1$	$\boldsymbol{F}$	T	F	D	D	D	D	D	D	D	D	D	D	D	D
x_1	$\boldsymbol{T}$	T	F	D	D	D	D	D	D	$F*$	D	$F*$	D	D	D

Figure 1.20: Example with $(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee x_4)$

in Figure 1.21. We leave it to the reader to work out the general case.

□

Exercise 1.44. Work out the general construction for the correct proof of Theorem 1.43 and prove that it works.

The motivation for this paper was

Why was finding if $G_{17,17}$ is 4-colorable so hard?

Towards this goal we showed, in Theorem 1.43, that GCE is NP-complete. But does this really capture the problem we want to study? We give several reasons why not. These will point to further investigations.

1) The reduction in Theorem 1.43 takes a 3CNF formula

$$\varphi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$$

and produces an instance (N, M, c, χ) of GCE such that

$$\varphi \in \text{3SAT if and only if } (N, M, c, \varphi) \in \text{GCE}.$$

In this instance $c = \Theta(NM)$. Hence our reduction only shows that GCE is hard if c is rather large. So what happens if c is small? See next point.

2) What happens if c is small? It is known that GCE is Fixed Parameter Tractable. In particular, the problem is in time $O(N^2M^2) + 2^{O(c^4+\log c)}$. This leads to the following open problem: find a framework to show that problems within FPT are still somewhat hard.

3) The 17×17 challenge can be rephrased as proving that $(17, 17, 4, \chi) \in$ GCE where χ is the empty partial coloring. This is a special case of GCE since none of the cell are pre-colored. It is

																		C_1	C_1	C_2	C_2	C_3	C_3
	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	T	T
	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	T	T	D	D
	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	T	T	D	D	D	D
$\overline{x}_4$		D	D	D	D	D	D	D	D	D	D	D	D	D	D	T	F	D	D	D	D	D	D
x_4		D	D	D	D	D	D	D	D	D	D	D	D	D	D	T	F	D	D	D	F	D	F
$\overline{x}_3$		D	D	D	D	D	D	D	D	D	D	D	D	T	F	D	D	D	F	D	F	D	D
x_3		D	D	D	D	D	D	D	D	D	D	D	D	T	F	D	D	D	D	D	D		
$\overline{x}_2$		D	D	D	D	D	D	D	D	D	D	T	F	D	D	D	D	D	D	D	D	D	D
x_2		D	D	D	D	D	D	D	D	T	F	T	F	D	D	D	D	D	D			D	D
$\overline{x}_2$		D	D	D	D	D	D	T	F	T	F	D	D	D	D	D	D	D	D	D	D	F	D
x_2		D	D	D	D	D	D	T	F	D	D	D	D	D	D	D	D			D	D	D	D
$\overline{x}_1$		D	D	D	D	T	F	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
x_1		D	D	T	F	T	F	D	D	D	D	D	D	D	D	D	D	D	D	F	D	D	D
$\overline{x}_1$		T	F	T	F	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
x_1		T	F	D	D	D	D	D	D	D	D	D	D	D	D	D	D	F	D	D	D	D	D

Figure 1.21: Example with $(x_1 \lor x_2 \lor \overline{x}_3) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x}_2 \lor x_3 \lor x_4)$

possible that the case where χ is the empty coloring is easy. While we doubt this is true, we note that we have not eliminated the possibility. How to deal with this issue?

One could ask about the problem

$$\mathrm{GC} = \{(N, M, c) \colon G_{N,M} \text{ is } c\text{-colorable } \}.$$

However, if N, M are in unary, then GC is a sparse set. By Mahaney's Theorem [Mah82] (see [HL94] for an alternative proof) if a sparse set is NP-complete then P = NP. If N, M are in binary, then we cannot show that GC is in NP since the obvious witness is exponential in the input. The formulation in binary does not get at the heart of the problem, since we believe it is hard because the number of possible colorings is large, not because N, M are large.

1.7 Pushing 1×1 Blocks

In this section we look at a large set of push-games.

Soko-Ban is a game (literally *warehouseman* in Japanese) where you, a 1×1 player, push around 1×1 boxes. Your job is to push boxes into target locations. Some boxes you can't push. You can only push 1 block at a time by one space (e.g. you'll be stuck if you push a box into a corner).

Lots of games have a pushing blocks component. Legend of Zelda has a variation where the blocks are on ice so blocks will fly off to infinity unless encountering an obstacle. The goal is to cover a target space with any block.

There are many variants of the Pushblock game. We describe the variants shown in Figure 1.22.

1. Normal model (upper left). Push. When you push a block, the block moves one space.
2. Push-k (upper right). k is the strength of the pusher (can only push k blocks at a time).
3. PushPush (down left). When you push a block, slides until you hit another block.
4. PushPushPush (bottom right). When you push a block, all slide until you hit an immovable block.
5. For Push-1X and Push-1G see Section 5.6.

Figure 1.23 describes known complexity of many variants using the following parameters:

1. ***Push*** designates how many blocks can be pushed (designated by $k \in \mathbb{N} \cup \{\infty\}$).
2. ***Fixed*** refers to if fixed blocks are available (typically in bounded rectangle).
3. ***Slide*** tries to capture how far things slide when pushed (PushPushPush redefines what "max" means). Recall that for the original game the goal was to push boxes into target locations.
4. ***Goal*** describes other possible goals. Some just have a path goal, to get the pusher to a place. One other variant includes a simple path variant, where you can't cross your own path (useful in games where blocks you walked on previously disappear).

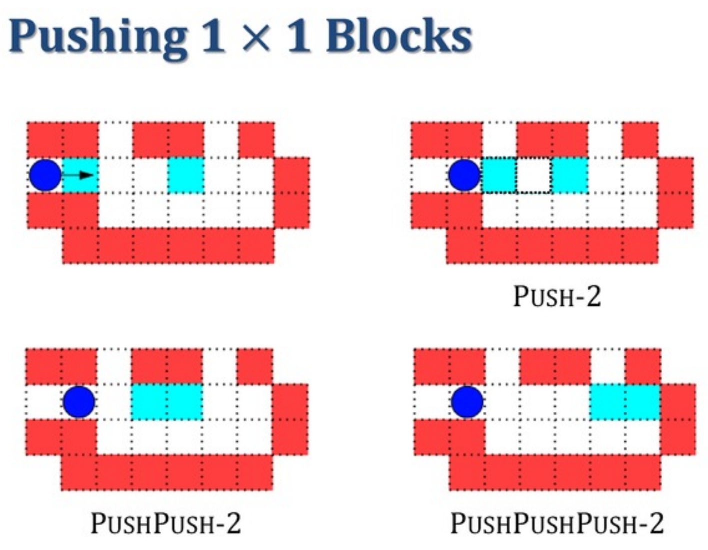


Figure 1.22: Variants of The Pushblock Game

Name	Push	Fixed	Slide	Goal	Complexity	Reference
Push-k	$k \geq 1$	no	min	path	NP-Hard	[DDO00]
Push-$*$	∞	no	min	path	NP-Hard	[Hof00]
PushPush-k	$k \geq 1$	no	min	path	PSPACE-Complete	[DHH04]
PushPush-$*$	∞	no	max	path	NP-Hard	[Hof00]
Push-1F	1	yes	min	path	NP-Hard	[DDO00]
Push-kF	$k \geq 2$	yes	min	path	PSPACE-Complete	[DHH02]
Push-$*$F	∞	yes	min	path	PSPACE-Complete	[BOS94]
Push-kX	$k \geq 1$	no	min	simple path	NP-completely	[DH01]
Push-$*$X	∞	no	min	simple path	NP-Complete	[Hof00]
Sokoban	1	yes	min	storage	PSPACE-complete	[Cul98]

Figure 1.23: Complexity of Push Games

All of these variants are NP-hard. Some are in NP (and hence NP-complete) but some are PSPACE-complete. None are harder than that.

Giving blocks variable weight seems like it would only make things harder.

A Variety of Push games
Instance: A board, initial square, and target square.
Question: Can the goal be achieved. Figure 1.23 contains variants of the game.

1.7.1 Push-* is NP-Complete

Theorem 1.45. *(Hoffmann [Hof00]) Push-* in a fixed box is NP-complete.*

Proof sketch: The model is Push-* in a fixed box. We reduce from 3SAT. Refer to Figures 1.24 and 1.25. Encode SAT gadgets into the blocks. Specifically, we have a Bridge Gadget, Variable Gadget, Clause Gadget, and Connection Gadget.

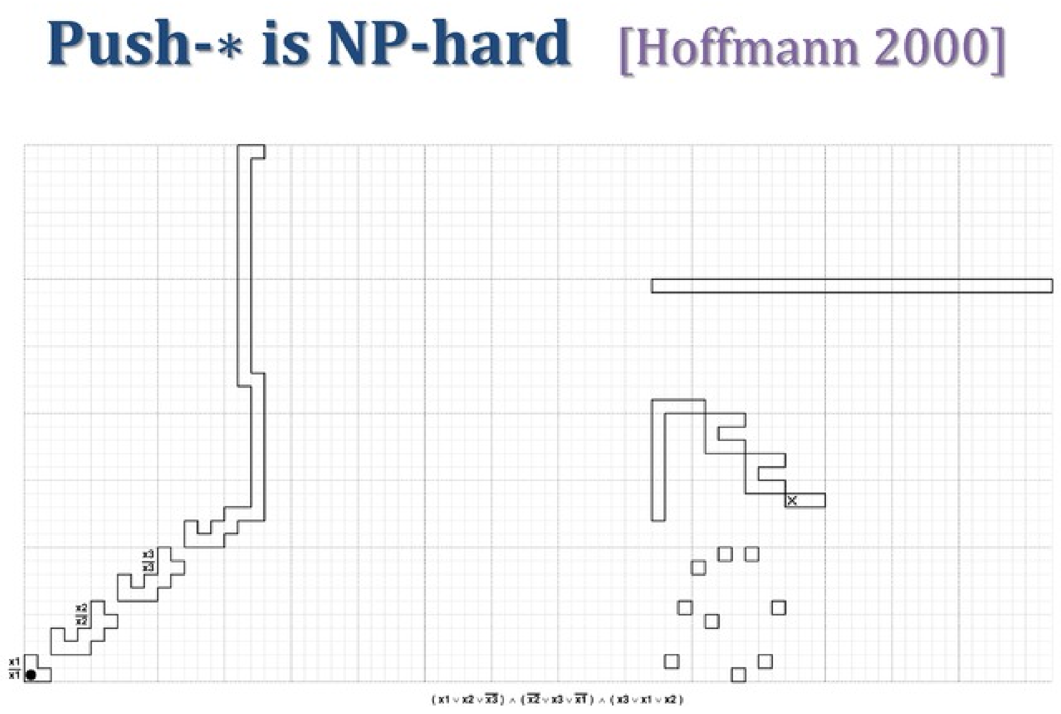

Push-* is NP-hard [Hoffmann 2000]
x1
x̄1
x2
x̄2
x3
x̄3
(x1 ∨ x2 ∨ x̄3) ∧ (x̄2 ∨ x3 ∨ x̄1) ∧ (x3 ∨ x1 ∨ x2)

Figure 1.24: Push-* Board That We Use

We construct a very constrained path where most of the space is occupied by movable blocks. First, we walk through the Variable Block where the pusher has an option to push rows to the right (either a positive or negative instance) into free space in the Connection Block. In general these gadgets are super tight, so there's nothing you can do except push a row to fill everything to your right; pushing both positive and negative instances of a variable only makes things worse further down the line (in the Clause Block).

The Connection Block (bottom right) is a bipartite graph encoded into a matrix with open spaces when variables (rows) connect to clauses (columns). Pushing a satisfying sequence of variables to the right will leave enough space in the Connection Block to maneuver the Clause Block later.

The Up-Bridge and Right-Bridge force the transition from variables to clauses to be one-way and to fill in the space to the left and top of the Clause Block.

The Clause Gadget allows you to make progress only when there is space below you on any of the variables in your clauses (this is why pushing both instance of a variable only hurt you, because it may block off needed empty squares). ∎

Exercise 1.46. Fill in the details of the proof of Theorem 1.45.

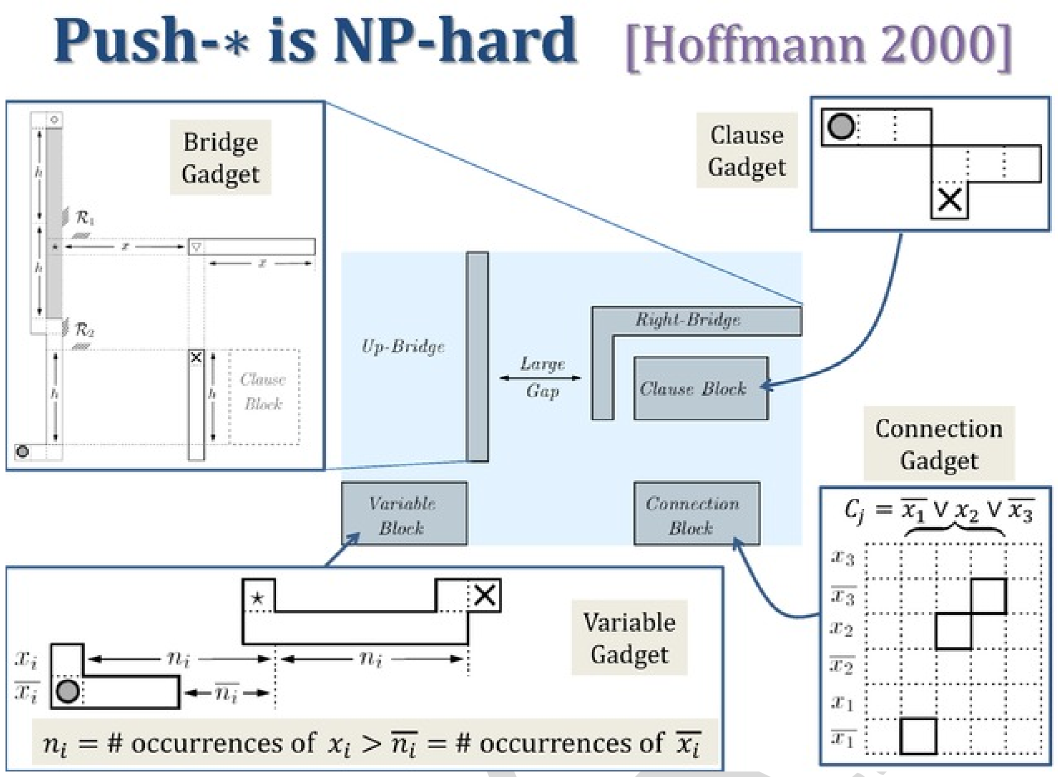

Push-∗ is NP-hard [Hoffmann 2000]
Bridge Gadget
Clause Gadget
Up-Bridge
Large Gap
Right-Bridge
Clause Block
Connection Gadget
Variable Block
Connection Block
Variable Gadget
n_i = # occurrences of x_i > $\overline{n_i}$ = # occurrences of $\overline{x_i}$
$C_j = \overline{x_1} \vee x_2 \vee \overline{x_3}$

Figure 1.25: Push-* Gadgets and How They Fit Together

1.7.2 PushPush-1 in 3D is NP-Complete

In PushPush-1, when you push a block, it slides until it hits another block or wall.

Theorem 1.47. *PushPush-1 in 3D is NP-complete.*

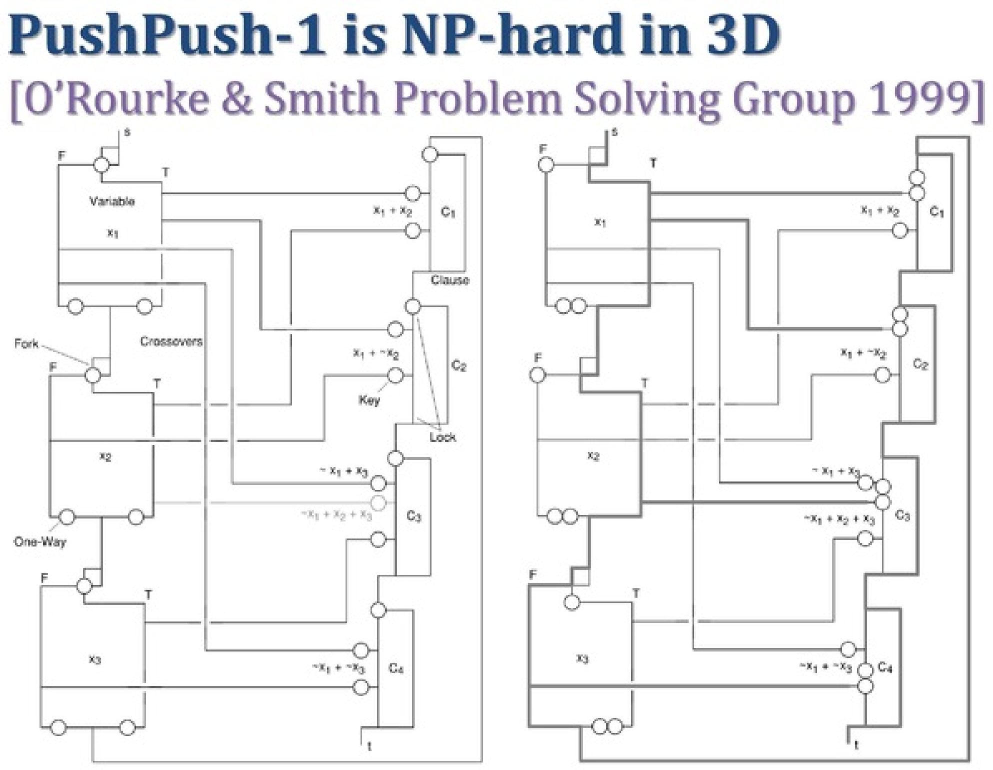
Proof sketch:

This problem involves pushing blocks with sliding in 3D (see Figure 1.26). The paths are little width 1 tunnels. At each variable block, you can push a block in the True or False direction which opens one way, and closes the other way, never to be traversed again. The two blocks at the bottom of each variable gadget keep you from backtracking. At the end of the last variable, you run through all the clauses. If any of the literals that satisfy a clause were visible, then you could have pushed it to the right, blocking off the left vertical tunnel in the clause gadget, allowing the top clause gadget block to be pushed down while keeping the bottom tunnel of the clause gadget open for traversal. This reduction is the basis for many of the proofs of this type; it is a very straight forward reduction from 3SAT.

Because we're in 3D, we don't need a crossover gadget as paths can be routed without crossing. ∎

1.7.3 PushPush-1 in 2D is NP-Complete in 2D

Theorem 1.48. *PushPush-1 in 2D is NP-complete.*


PushPush-1 is NP-hard in 3D

[O'Rourke & Smith Problem Solving Group 1999]

Figure 1.26: PushPush-1 in 3D is NP-Complete

Proof sketch:

For 2D, we need a lock gadget and a crossover gadget (See Figures 1.27, 1.28, 1.29). For the lock gadget (bottom left), the goal is to go from A to V. If you try to go from A to V directly, it's impossible as constructed. But if you visit from U first, you can unlock the lock. We can use this lock to create a clause gadget with three possible entries. If you come down the top path, it prevents you from using the other half of the gadget and escaping along that path. As you come down, you force the gadgets to be in a particular state to force you to go back the way you came. Then, once all clauses are unlocked, you will be able to traverse through all the lock gadgets to complete the path.

Then we have the issue of a crossover gadget in 2D (see Figure 1.29). Going from N to S, we can push Y down, but then can't go W to E. Can only do one or the other. This is called an XOR crossover gadget, usable only once. Not what we want, but we can chain together locks with the XOR crossover to create a unidirectional crossover gadget as shown in Figure 1.29. You can do one of three things: N-S, W-E, or N-S-W-E. Included also are no-reverse gadgets that do not allow you to return the way you came. Then we connect variables and clauses together. We know the order in which the crossovers happen, so we can orient the crossovers in the right orientation. Gadgets are different from Super Mario Bros, but proof structure is the same.

∎

Demaine et al. [DDHO01] have shown more Push games are NP-complete.

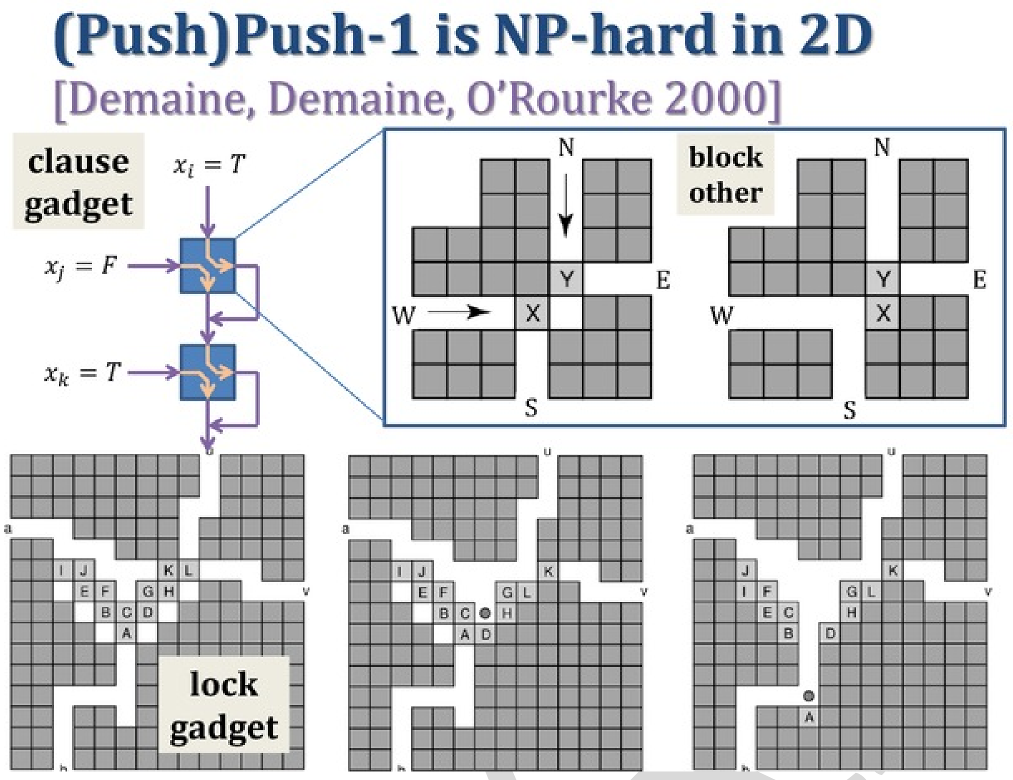


Figure 1.27: PushPush-1 is NP-Complete in 2D

1.8 Video Games that are NP-Hard

In this section we will look at Super Mario Brothers (often just called Mario) both as it is intended to be played, and also how it can be played given some of the glitches in the actual program. Aloupi et al. [ADGV15] showed that both are NP-hard. In that same paper they also showed that Donkey Kong, Legend of Zelda, Metroid, and Pokemon are also NP-hard. ater Demaine et al. [DVW16] showed that Super Mario Brothers is PSPACE-complete.

1.8.1 Super Mario Brothers

Super Mario Brother is a platform game. The human player who controls a character must make the character jump and climb between suspended platforms while avoiding various obstacles. The game *as intended* has limits on what Mario can do. The game *as implemented* has some glitches which allow Mario to make moves that were not intended. We first prove that Super Mario-as-intended is NP-hard and then that Super Mario-with-Glitches is NP-hard.

Mario Brothers as Intended

Theorem 1.49. *Mario Bros is NP-hard.*

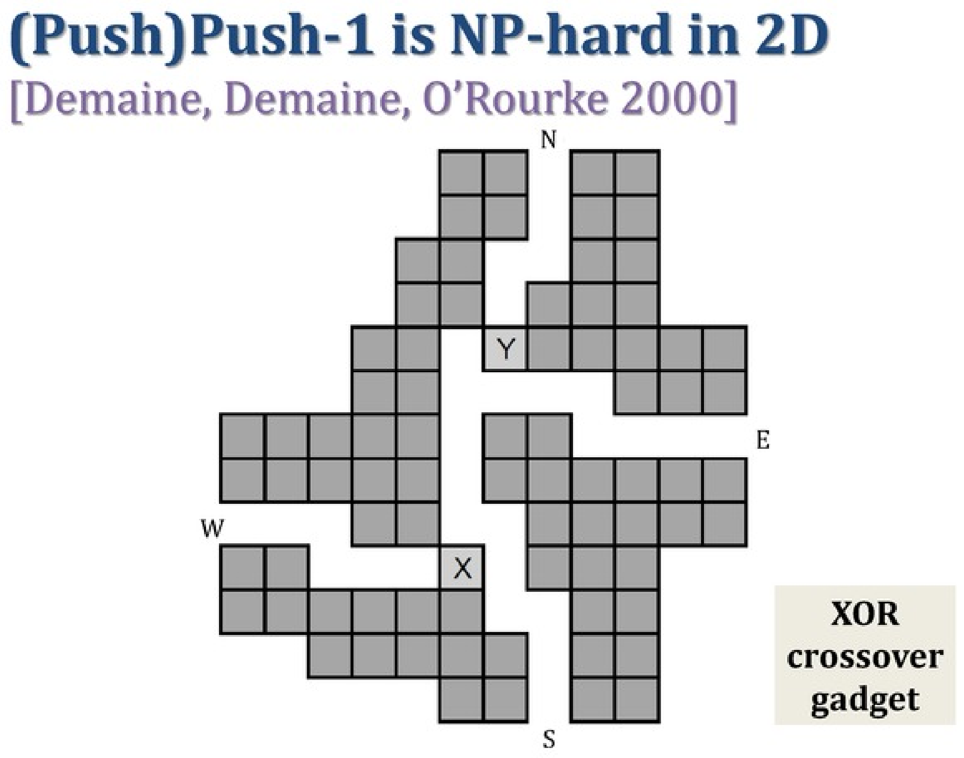

(Push)Push-1 is NP-hard in 2D
[Demaine, Demaine, O'Rourke 2000]
N
Y
E
W
X
S
XOR crossover gadget

Figure 1.28: XOR Crossover Gadget

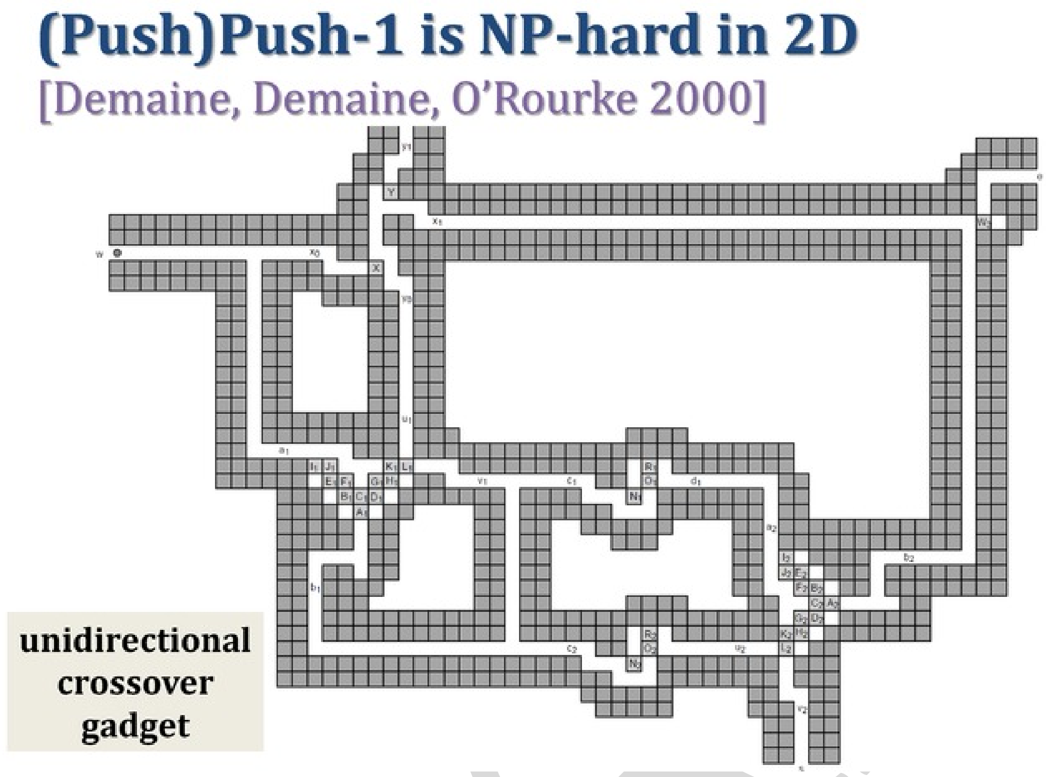
Proof sketch: We show 3SAT $\leq_p$ Mario Bros.

Given an instance of 3SAT, which has some variables, literals, and clauses, we'll make *gadgets* to represent them in Mario.

A variables gadget is a place where you have to fall in one direction or the other, which we think of as corresponding to setting the variable true or false. A clause gadget is a set of fire bars to run through, just after three boxes in the floor containing invincibility stars that last long enough to let you run through the fire bars. Each box containing an invincibility star is breakable from the gadget for exactly one literal in the clause; that is, you can release an invincibility star for a clause gadget if and only if in at least one of that clause's variables' gadgets you made the satisfying choice.

The gadgets are arranged so that you first set each variable (and can pop out all the corresponding stars), then run through the clause gadgets in order, which is possible if and only if one literal from each of them was satisfied, that is, if and only if there's a satisfying assignment to the original 3-SAT problem.

The graph of paths between the variable gadgets and clause gadgets isn't planar, so there's also a crossover gadget for the paths, guaranteeing that when entered from the left it can only be exited from the right, and when entered from the bottom it can only be exited from the top. Note that the crossover gadget breaks if you go through it both ways, but that's ok because we only care about reachability. ∎


(Push)Push-1 is NP-hard in 2D
[Demaine, Demaine, O'Rourke 2000]
unidirectional crossover gadget
Figure 1.29: Unidirectional Crossover Gadget

Mario Brothers with Glitches

As noted above, the way Mario Brothers is implemented has some glitches in it that allow the player to do moves that were not intended. We show that this version is still NP-hard.

We do not define the game Mario Bros with Glitches rigorously. All you need to know is

1. Mario can jump up walls.
2. Mario can jump through walls.
3. There may be other things Mario can do.

Definition 1.50. Mario Bros-G is the problem of, given a starting position for Mario Brothers with glitches, is there a way to complete the game.

Theorem 1.51. *Mario Bros-G is NP-hard.*

Proof. We describe how to modify the proof that 3SAT $\leq_p$ Mario Bros. You could view this formally as a reduction Mario Bros $\leq_p$ Mario Bros-G.

In Mario Bros-G Mario can jump up walls. To prevent this, we put bars on the walls as shown in Figure 1.31.

In Mario Bros-G Mario can jump through walls. To prevent this we replace walls by walls filled with monsters as shown in Figure 1.32.

For anything else Mario can do, there is a way to modify the game, similar to what we did in Figures 1.31 and 1.32, to prevent him from doing it. □

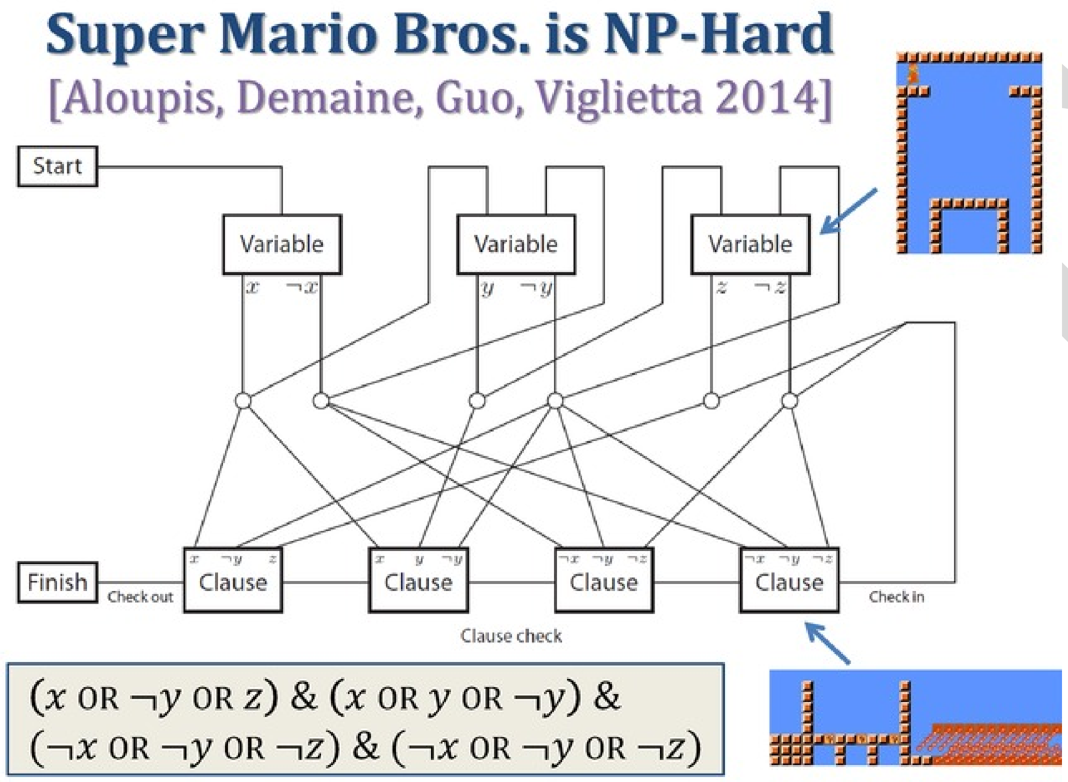

Super Mario Bros. is NP-Hard
[Aloupis, Demaine, Guo, Viglietta 2014]
Start
Variable
x $\neg x$
Variable
y $\neg y$
Variable
z $\neg z$
Finish
Check out
Clause
Clause
Clause
Clause
Check in
Clause check
(x OR ¬y OR z) & (x OR y OR ¬y) &
(¬x OR ¬y OR ¬z) & (¬x OR ¬y OR ¬z)

Figure 1.30: Super Mario Brothers

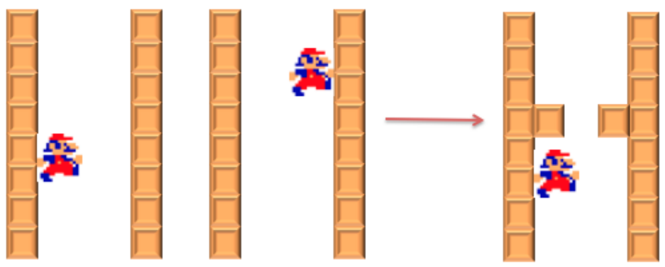


Figure 1.31: Making Wall Jumps Impossible

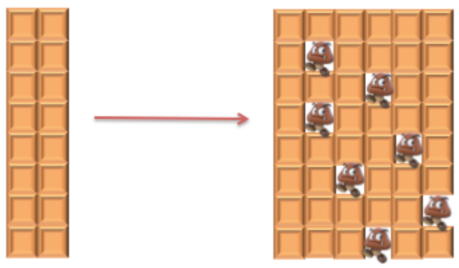


Figure 1.32: Making Jumping Through Walls Impossible

1.9 Further Results

There are NP-complete problems and NP-hard problems in many different fields. We will see this throughout this book; however, for now, we have two exercises from two different fields.

Exercise 1.52. Recall the problem 0-1 PROGRAMMING from Example 0.9. Show that 0-1 PROGRAMMING is NP-complete by a reduction from 3SAT.

Exercise 1.53. Show that the following problem is NP-hard: Given a nondeterministic finite automata M, does there exist a string that is not accepted by it?

Chapter 2

NP-Hardness via SAT and Planar SAT

2.1 Introduction

Many graph problems are NP-complete. In this chapter we show that many NP-complete graph problems, when restricted to planar graphs, are still NP-complete. Let XX be a graph problem (e.g., Hamiltonian Cycle) and Planar XX be its planar version (e.g, Hamiltonian Cycle restricted to planar graphs). One way to show that Planar XX is NP-complete is to show XX is NP-complete and then show XX $\le_p$ Planar XX by devising some crossover gadget to remove crossings. This has been used on some problems including 3-colorability. We will *not* take this approach. We will define Planar 3SAT which is an NP-complete variant of SAT, and then show Planar 3SAT $\le_p$ Planar XX directly. We will also use variants of Planar 3SAT. We note that Planar 3SAT is not a natural problem. It is a means to an end.

Why take this approach?

1. Rather than devise many crossover gadgets for many problems, we essentially devise only one, the one used to show Planar 3SAT is NP-complete.

2. Gurhar et al. [GKM+12], and independently Burke [Bur], showed that, for the Planar Hamiltonian Cycle problem, it is literally *impossible* to create crossover gadgets. This may be true for other problems as well.

We may also restrict the degree of a graph.

Definition 2.1. The ***degree of a graph*** is the max degree of all of its vertices. A graph is ***regular*** if all vertices have the same degree, which of course will be the degree of the graph.

For some of our reductions an input of length n is mapped to an output of length $O(n)$ or of length $O(n^2)$. This will be useful for Chapter 7.

Definition 2.2. Let $A \le_p B$ via f.

1. f has ***linear blowup*** if $|f(x)| \le O(|x|)$.

2. f has ***quadratic blowup*** if $|f(x)| \le O(|x|^2)$.

We will also do some reductions from 3SAT both for contrast and because sometimes these reductions are the same as the ones from Planar 3SAT.

2.2 How to Reduce from Planar 3SAT

Section 1.3 listed common gadgets for reductions from 3SAT and variations. In PLANAR 3SAT and its variations, some additional gadgets become common because we are usually reducing to a problem in two dimensions.

To produce a reduction in 2D, we usually need to define explicit coordinates. A useful starting point is that every n-vertex planar graph can be drawn with its vertices placed at grid points of an $n \times n$ grid, where edges are drawn as straight lines and do not cross [Sch90]. Such a drawing can be computed in $O(n)$ time.

In many situations, however, it is difficult to design the wires implementing the edges of the graph that connect two arbitrary grid points. Often it is simpler to produce wire gadgets that only work in certain directions (e.g., horizontal and vertical). In this case, we need edges to be drawn as polygonal lines instead of single line segments.

A useful starting point for this type of reduction is that every maximum-degree-4 n-vertex planar graph can be drawn on an $n \times n$ grid, with its vertices placed at grid points and its edges routed along grid edges [BK98]. Furthermore, each edge has at most two bends, and such a drawing can be computed in $O(n)$ time. Back to hardness reductions, in this case we need a ***turn gadget*** that shows how to bend a (semi)wire in 2D.

When embedding on a grid, wires often have a parity constraint where they must repeat a module of size k, so their size is always of a fixed length modulo k. This can be a problem when not all gadgets are aligned modulo k. In this case, we often need a ***shift gadget*** to adjust a wire's position modulo k, e.g., by ± 1. Applying a shift gadget enough times, we can adjust a wire's end positions modulo k to match whatever gadgets they need to attach to.

2.3 CLIQUE, INDEPENDENT SET, and VERTEX COVER

We start with some nonplanar problems that will be good examples throughout the book.

Clique (CLIQUE) and Independent Set (INDEPENDENT SET)
Instance: Graph $G = (V, E)$ and a $k \in \mathbb{N}$.
Question: For Clique (Independent Set): are there k points such that every pair (no pair) has an edge between them.

Vertex Cover (VERTEX COVER)
Instance: Graph $G = (V, E)$ and a $k \in \mathbb{N}$.
Question: Is there a $V' \subseteq V$ of size k such that every $e \in E$ has some $v \in V'$ as an endpoint.

Karp [Kar72] proved that CLIQUE, VERTEX COVER, and INDEPENDENT SET are NP-complete. For CLIQUE we give a different proof (folklore) that has properties we will use in Theorem 12.30. For INDEPENDENT SET and VERTEX COVER we give Karp's proofs.

Theorem 2.3.

1. *3SAT $\leq_p$ CLIQUE, so CLIQUE is NP-complete. This reduction has linear blowup.*

2. *CLIQUE $\leq_p$ INDEPENDENT SET, so INDEPENDENT SET is NP-complete. The reduction has linear blowup.*

3. *INDEPENDENT SET $\leq_p$ VERTEX COVER, so VERTEX COVER is NP-complete. The reduction has linear blowup.*

Proof.
1) We give an example of our reduction using a formula that has a two 3-clauses and one 2-clause in Figure 2.1. While this is not strictly a 3CNF formula, it is a better example for readability.

Here is the reduction.

1. Input $\varphi = C_1 \wedge \cdots \wedge C_k$ where each C_i is a 3-clause.

2. We create a graph G with $7k$ vertices as follows: For each clause we have 7 vertices. Label them with the 7 ways to set the 3 variables to make the clause satisfiable. For example, for the clause $x \vee y \vee \neg z$, we have 7 vertices
   - $(x = \text{TRUE}, y = \text{TRUE}, z = \text{TRUE})$
   - $(x = \text{TRUE}, y = \text{TRUE}, z = \text{FALSE})$
   - $(x = \text{TRUE}, y = \text{FALSE}, z = \text{TRUE})$
   - $(x = \text{TRUE}, y = \text{FALSE}, z = \text{FALSE})$
   - $(x = \text{FALSE}, y = \text{TRUE}, z = \text{TRUE})$
   - $(x = \text{FALSE}, y = \text{TRUE}, z = \text{FALSE})$
   - $(x = \text{FALSE}, y = \text{FALSE}, z = \text{FALSE})$

3. There are no edges between vertices associated to the same clause. We put an edge between vertices associated with different clauses if the assignments do not conflict. For example, vertex $(x = \text{TRUE}, y = \text{TRUE}, z = \text{TRUE})$ will have an edge to the vertex $(w = \text{FALSE}, x = \text{TRUE}, z = \text{TRUE})$ but not to the vertex $(w = \text{FALSE}, x = \text{FALSE}, z = \text{TRUE})$.

We leave it to the reader to show that the $\varphi \in$ 3SAT if and only if $(G, k) \in$ CLIQUE.

2) G has a clique of size k if and only if $\overline{G}$ has an independent set of size k.

3) G has an independent set of size k if and only if G has a vertex cover of size $n - k$. □

If we restrict G to be planar then what happens?

Theorem 2.4.

1. *The CLIQUE problem restricted to planar graphs is in P. This is an easy exercise.*

2. *The VERTEX COVER problem restricted to planar graphs of degree 3 is NP-complete. This is due to Garey et al. [GJS76]. We will prove it later (Theorem 2.14).*

3. *The INDEPENDENT SET problem restricted to planar graphs of degree 3 is NP-complete. This follows easily from Part 2.*

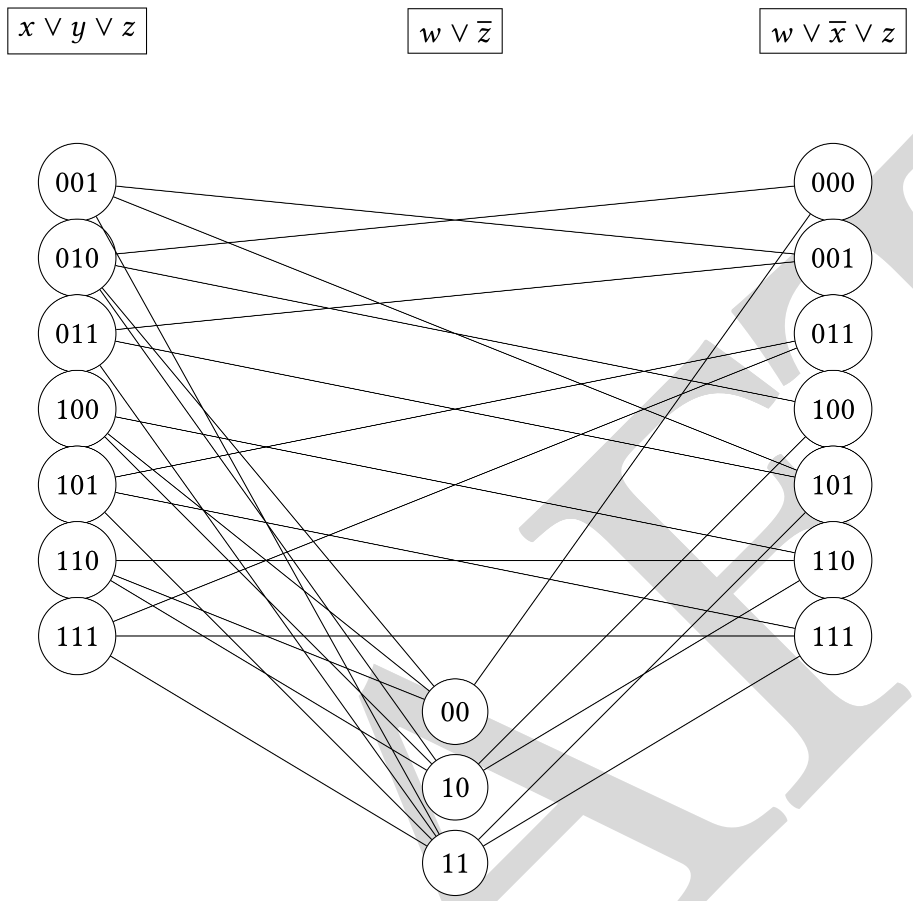


Figure 2.1: Reduction of 3SAT to Clique

Exercise 2.5. Prove Theorem 2.4.1.

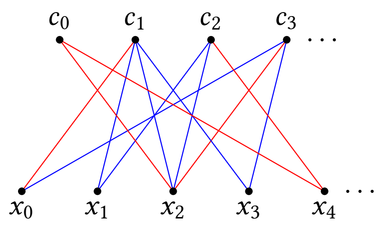
Exercise 2.6. Let Setpack be the following problem: Given $\{1, \ldots, n\}$, $X_1, \ldots, X_m \subseteq \{1, \ldots, n\}$, and k, are there k X_i's that are all disjoint? Show that Independent Set $\le$ Setpack, hence Setpack is NP-complete.

Exercise 2.7. Let Mastermind be the following problem: Given a position in the game Mastermind, is there a solution? Read Stuckman & Zhang's paper [SZ06] on the complexity of Mastermind. Rewrite their proof that Vertex Cover $\le$ Mastermind in your own words. (For more on the complexity of Mastermind, see the papers of Goodrich [Goo09], Viglietta [Vig12], Ben-Ari [BA18], and Doerr et al. [DDST16]. This is not a complete list of papers.)

2.4 Planar 3-SAT

A CNF formula can be viewed as a bipartite graph with (1) variables on one side and clauses on the other and (2) a red edge between x and C if x is in C, a blue edge between x and C if $\neg x$ is in C, and no edge between x and C if x is not in C. See Figure 2.2 for an example.

Definition 2.8. A formula is ***planar*** if the associated bipartite graph is planar.



$$c_0 = (\neg x_2 \wedge \neg x_4)$$
$$c_1 = (\neg x_0 \wedge x_1 \wedge x_2 \wedge x_3)$$
$$c_2 = (x_1 \wedge x_2 \wedge \neg x_4)$$
$$c_3 = (x_0 \wedge \neg x_2 \wedge x_3)$$
$$\cdots$$

Figure 2.2: Bipartite Graph Associated to a CNF Formula

Planar 3SAT
Instance: A planar formula φ given as a bipartite graph. Hopcroft & Tarjan [HT74] showed that determining whether a graph is planar is in polynomial time (actually linear time). Hence this can be checked. If the graph is not planar they output no.
Question: Is φ satisfiable?

Note: When we define variants of Planar 3SAT we will omit mentioning that planarity can be checked quickly.

David Lichtenstein [Lic82] showed that Planar 3SAT is NP-complete.

Theorem 2.9.

1. *3SAT $\le_p$ Planar 3SAT so Planar 3SAT is NP-complete. The reduction has quadratic blowup.*

2. *The problem remains NP-complete if we insist that all of the variable vertices are connected in a cycle. (This will be left as an exercise.)*

3. *The problem remains NP-complete if we insist that, for every variable vertex v, the edges to clauses where it appears positively have v as a left endpoint, and the edges to clauses where it appears negatively have v as a right endpoint. (This will be left as an exercise.)*

Proof. We show 3SAT $\le_p$ Planar 3SAT.

1. Input $\varphi = C_1 \wedge \cdots \wedge C_k$ where each C_i is a 3-clause.

2. Create the bipartite graph that represents φ. If the graph is planar then you are done, output that graph. More likely it is not.

3. Replace every crossing in the graph with a crossover gadget that we will describe now.

   Create a graph containing nodes representing variables and clauses as described above and connect it in the way shown in Figure 2.3 thus removing all crossings. In this gadget, the small green circles represent clauses and big blue circles are variables. The blue connections are positive while red are negative. Because the graph is bipartite, we can't disrupt planarity by the connections between variables and clauses.

   We leave it as an exercise that the truth value of a is the same as that of a_1 and the truth value of b is the same as that of b_1. We are almost done.

(There is a slight glitch in our gadget. Some clauses contain 4 literals and some contain 2 literals; however, we need every clause to have 3 literals. We can easily fix both situations. For a clause that only contains 2 variables, we can either create parallel edges from one variable to the clause or create a separate false variable to include in the constraint. For the 4 variable clause, we borrow the reduction from 4SAT to 3SAT to create a gadget that involves introducing an extra variable set to false which represents whether the left or right side of the original clause contained the satisfying literal.)

□

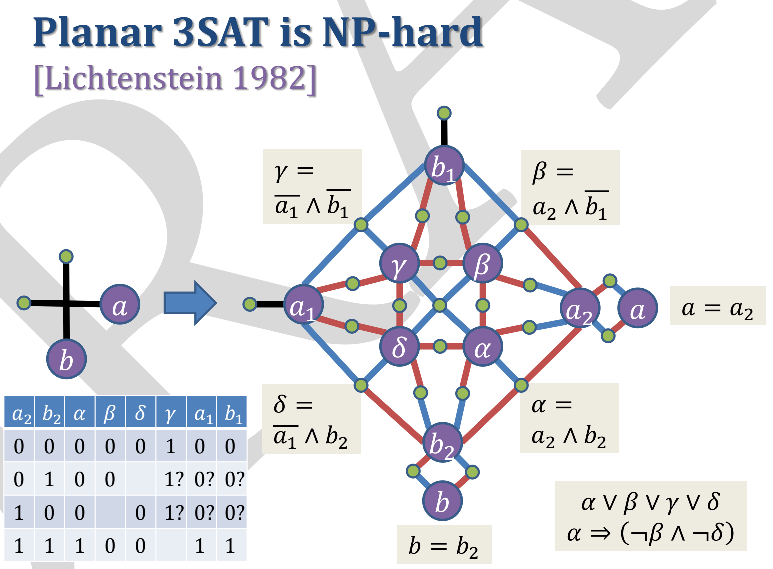
Exercise 2.10. This problem is in reference to the proof of Theorem 2.9.

1. Show that the gadget in Figure 2.3 works.

2. Show that the reduction in Part 1 has quadratic blowup.

3. Show that one can put a cycle through all of the variable vertices of the planar bipartite graph that is the result of the reduction. Figure 2.4 is a hint.

4. Show that one can modify the planar bipartite graph as specified in Theorem 2.9.3. Figure 2.5 is a hint.



Planar 3SAT is NP-hard
[Lichtenstein 1982]
$\gamma = \overline{a_1} \wedge b_1$
$\beta = a_2 \wedge \overline{b_1}$
a
b
b_1
γ
β
a_1
a_2
a
$a = a_2$
δ
α
$\delta = \overline{a_1} \wedge b_2$
$\alpha = a_2 \wedge b_2$
b_2
b
$b = b_2$
$\alpha \vee \beta \vee \gamma \vee \delta$
$\alpha \Rightarrow (\neg\beta \wedge \neg\delta)$

a_2	b_2	α	β	δ	γ	a_1	b_1
0	0	0	0	0	1	0	0
0	1	0	0		1?	0?	0?
1	0	0		0	1?	0?	0?
1	1	1	0	0		1	1

Figure 2.3: Planar 3SAT Crossover Gadget

2.5 Graph Coloring and Variants

Graph 2-Coloring is easily seen to be in polynomial time. Karp [Kar72] showed that if the number of colors is allowed to vary then the problem is NP-complete (it was one of his original 21 problems). What if the number of colors is fixed at 3?

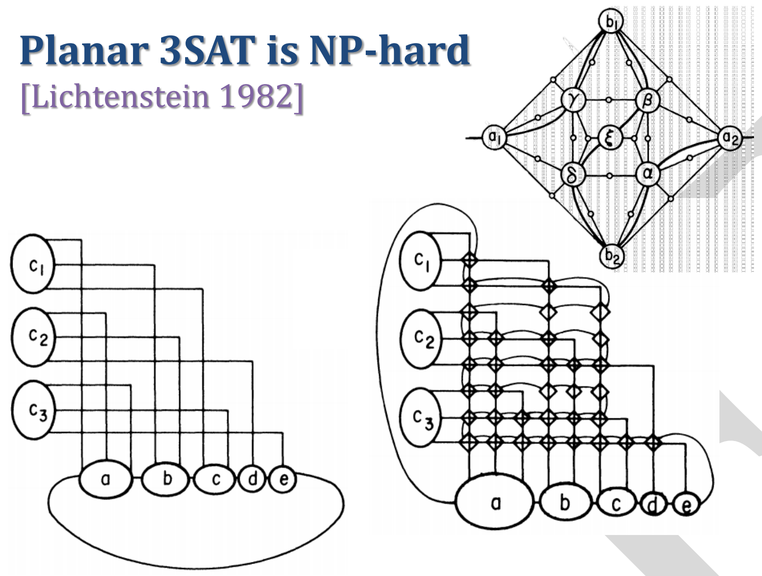


Figure 2.4: Planar 3SAT with a cycle through variable nodes. Top right: the path through the crossover gadget. Left: the initial arrangement of the bipartite graph, with the variables in a cycle. Bottom right: the arranged graph, with crossover gadgets inserted to preserve planarity.

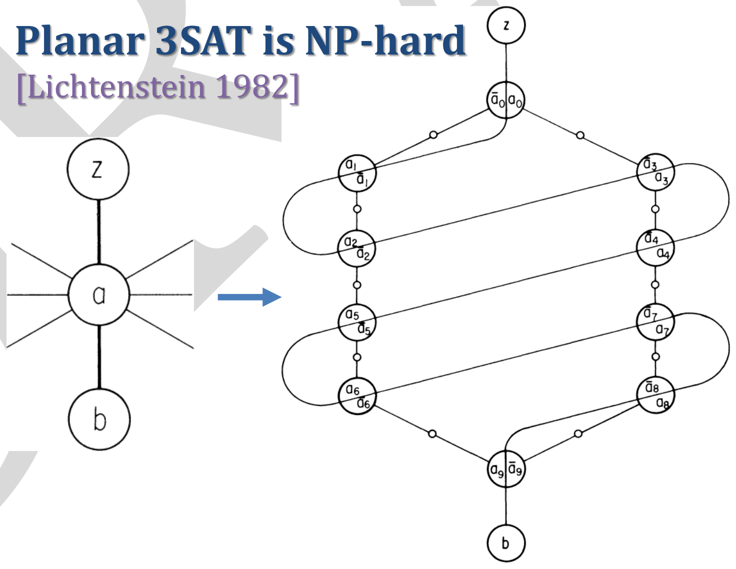


Figure 2.5: Planar 3SAT with Variables Divided.

3COL and Planar 3COL and Planar Degree-4 3COL
Instance: A graph $G = (V, E)$. For Planar 3COL the graph is planar. For Planar Degree-4 3COL the graph has degree 4.
Question: Does there exist a 3-coloring of G, which is an assignment of V to $\{1, 2, 3\}$ (the colors) such that all adjacent vertices have different colors.

Garey et al. [GJS76] showed that 3COL, Planar 3COL, and Planar Degree-4 3COL are all NP-complete. Their proof did not use Planar 3SAT. We present their proof. An alternative proof using Planar 3SAT is left as an exercise.

Theorem 2.11.

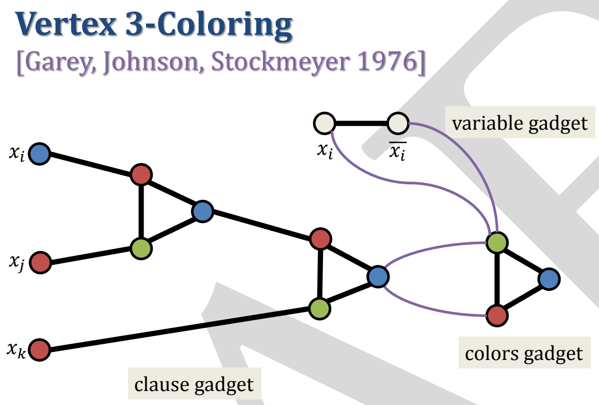
1. *3SAT $\le_p$ 3COL, so 3COL is NP-complete. The reduction has linear blowup.*
2. *Planar 3SAT $\le_p$ Planar 3COL, so Planar 3COL is NP-complete. The reduction has linear blowup. (This will be an exercise.)*
3. *3COL $\le_p$ Planar 3COL so (again) Planar 3COL is NP-complete. The reduction has quadratic blowup.*
4. *3COL $\le_p$ Planar Degree-4 3COL so Planar Degree-4 3COL is NP-complete. The reduction has quadratic blowup.*

Proof.
1) We show 3SAT $\le_p$ 3COL.

1. Input $\varphi = C_1 \wedge \cdots \wedge C_m$.
2. Create a graph G as follows.
   (a) There is a triangle which we think of as being colored TRUE, FALSE, and Green. In Figure 2.6 Blue is TRUE and Red is FALSE.
   (b) For every variable x there is a vertex x, a vertex $\overline{x}$, an edge between them, and an edge from Green to both of them. This forces one of them to be TRUE and the other to be FALSE.
   (c) For every clause there is a gadget like the one for $x_i \vee x_j \vee x_k$ in Figure 2.6. One can show that of the three vertices corresponding to the literals, in a 3-coloring, the following occurs: (1) each one is colored TRUE or FALSE, (2) at least one is colored TRUE.

We leave it to the reader to show that $\varphi \in$ 3SAT if and only if $G \in$ 3COL and that the reduction has linear blowup.

Exercise 2.12. Give an alternative proof that 3COL is NP-complete using the gadgets in Figures 2.9, 2.10, 2.11, and 2.12.



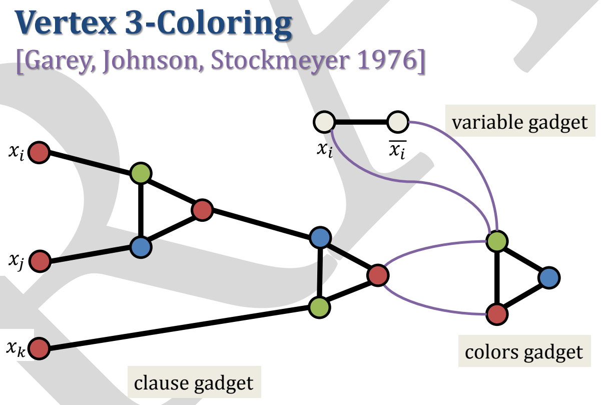


Figure 2.6: Gadgets for 3COL

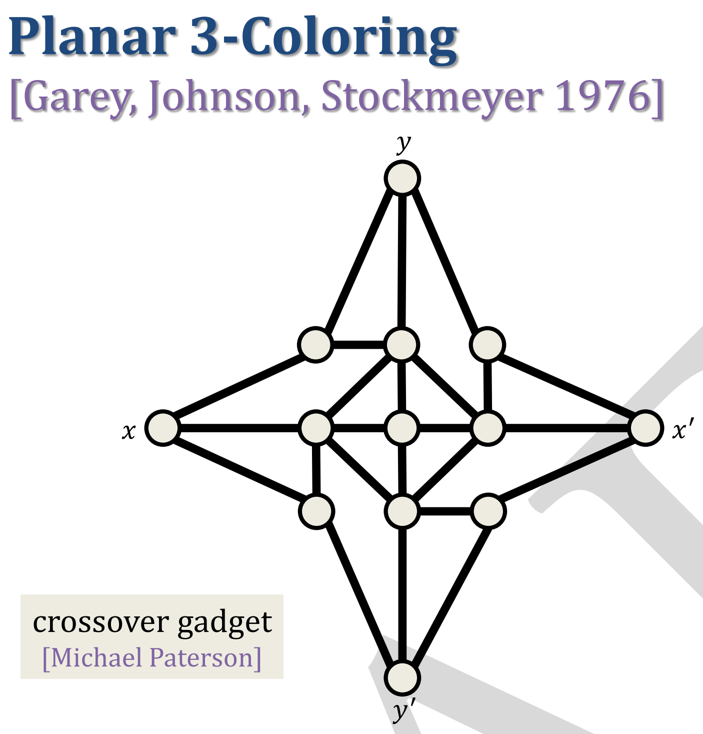


Figure 2.7: Crossover Gadget

2) The proof that PLANAR 3SAT $\le_p$ PLANAR 3COL is an exercise.

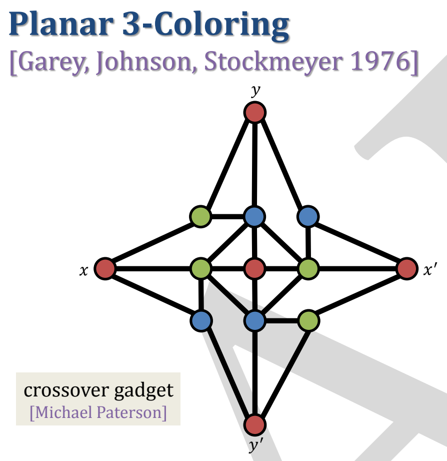
3) We show that 3COL $\le_p$ PLANAR 3COL. We show that every crossing can be replaced by a crossover gadget. If (x, x') crosses (y, y') then replace the crossing with the gadget in Figure 2.7.

Figure 2.8 shows the two distinct ways (up to permutation) in which the crossover gadget can be colored. Both cases result in $x = x'$ and $y = y'$.

There is one subtlety in the use of the crossover gadget. Say the edge from x to z included a crossing, and we wished to use the above crossover gadget. We would **NOT** identify x' with z, as we wish those vertices to be different colors. Instead, draw a segment connecting x' to z.

We call the new graph G'. We leave it to the reader to show that $G \in$ 3COL if and only if $G' \in$ PLANAR 3COL.

4) We show that PLANAR 3COL restricted to degree 4 graphs is NP-complete by reducing to PLANAR 3COL to it. We replace all vertices of degree ≥ 5 with the gadget in Figure 2.13. □



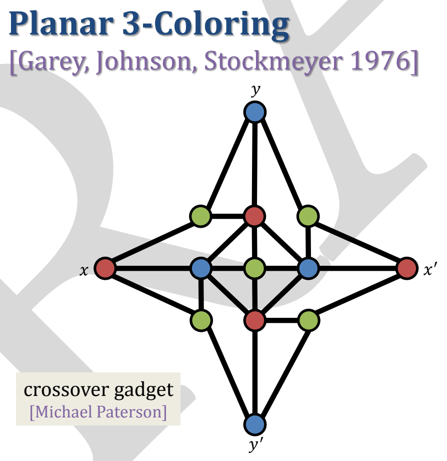


Figure 2.8: Two Ways to 3-Color the Crossover Gadget

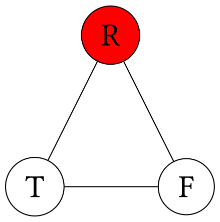


Figure 2.9: Alternative Gadget for 3COL: The Triangle

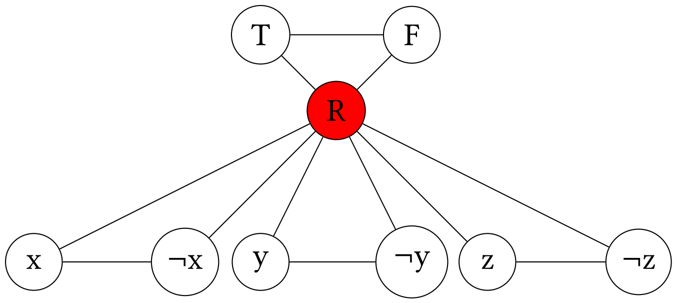


Figure 2.10: Alt Gadget for 3COL: Variables

Exercise 2.13. All of these questions are in reference to the proof of Theorem 2.11.

1. Show that the reductions given in parts 1, 3, and 4 work.
2. Do Part 2.
3. Show that all of the reductions have the blowup indicated.

Theorem 2.11 raises the question of which variants of graph coloring are in P. The following are known.

1. 3COL restricted to graphs of degree 3 is in P since, by Brook's Theorem [Bro41], such graphs are always 3-colorable.
2. 4-colorability of planar graph is in P since, by the celebrated 4-color Theorem of Appel et al. [AH77a, AHK77, AH77b] (and later a simpler proof by Robertson et al. [RSST97]) all planar graphs are 4-colorable.

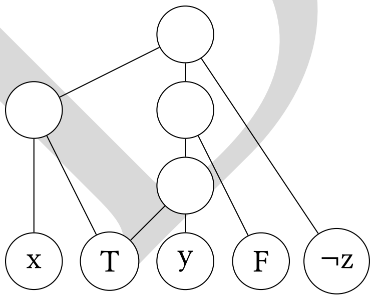


Figure 2.11: Alternative Gadgets: Clauses

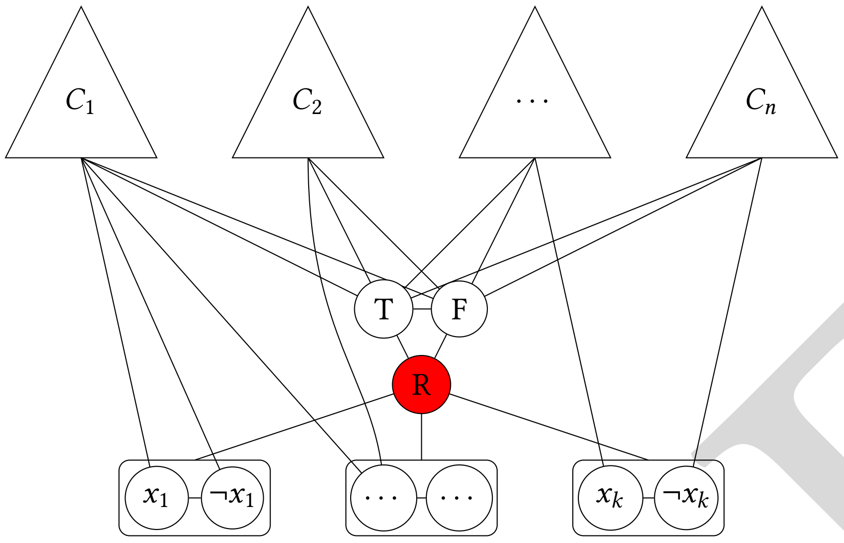


Figure 2.12: Alt Gadget: Putting it all Together

Planar 3-Coloring, Max Degree 4

[Garey, Johnson, Stockmeyer 1976]

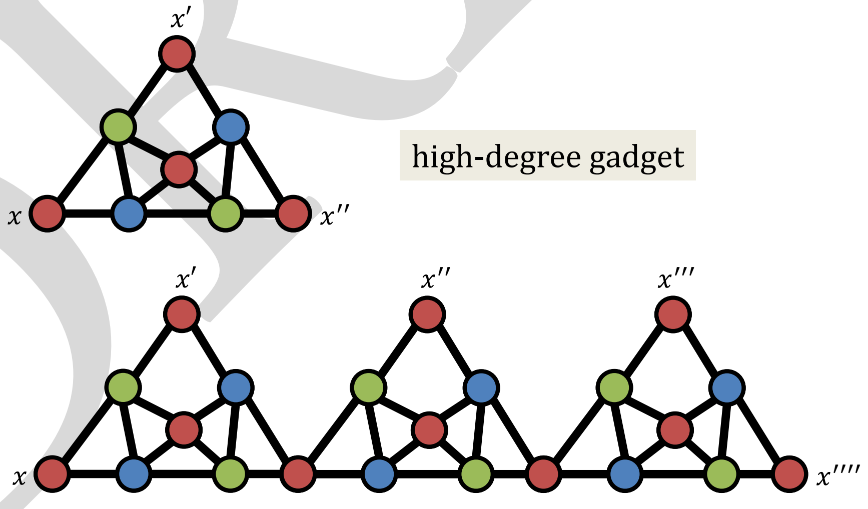


Figure 2.13: Gadget for Reducing Degree to 4

2.6 Vertex Cover and Variants

Vertex Cover and Variants (Vertex Cover, Vertex Cover-DEG3, Planar Vertex Cover, Planar VC-DEG3)
Instance: A Graph $G = (V, E)$ and a number k. If Vertex Cover-DEG3 then the graph has degree 3. If Planar Vertex Cover then the graph is planar. If Planar VC-DEG3 then the graph is planar and of degree 3.
Question: Is there a $V' \subseteq V$ of size k such that for every $e \in E$ has some $v \in V'$ as an endpoint.

Lichtenstein [Lic82] proved Planar VC-DEG3 is NP-complete by giving a reduction from 3SAT to Vertex Cover-DEG3 that, when restricted to planar formulas outputs planar graphs.

Theorem 2.14.

1. *3SAT $\le_p$ Vertex Cover-DEG3, so Vertex Cover-DEG3 is NP-complete. The reduction has linear blowup.*
2. *Planar 3SAT $\le_p$ Planar VC-DEG3, so Planar VC-DEG3 is NP-complete. The reduction has linear blowup.*
3. *3SAT $\le_p$ Planar VC-DEG3, so (again) Planar VC-DEG3 is NP-complete. The reduction has quadratic blowup.*

Proof.
1) We give a reduction of 3SAT to Vertex Cover. Figure 2.14 is an example of the reduction. It will be clear that the resulting graph has degree 3.

1. Input $\varphi = C_1 \wedge \cdots \wedge C_m$.
2. Form a graph $G = (V, E)$ as follows:
   (a) For every clause C_i there is a triangle with the three vertices labeled with the three literals in C_i. We call this a ***Clause-Triangle***. Note that any vertex cover of the graph will need at least 2 vertices from each triangle, so every vertex cover has at least $2m$ vertices from the clause-triangles.
   (b) Let x be a variable. Let m_x be the number of times x or $\overline{x}$ appears in φ. Form a cycle of length $2m_x$ where the vertices are alternatively labeled $x, \overline{x}, \ldots$. We call this a ***var-cycle***. (If $m_x = 1$ then just have a line graph with x and $\neg x$, but not a cycle.) Note the following: (1) there are enough vertices in the var-cycle for x for all of the clauses that x appears in, (2) any vertex cover of the graph will use at least m_x vertices from this var-cycle, and hence (3) every vertex cover has at least $\sum_x m_x = 3m$ vertices from the var-cycles.
   (c) For all (x, C) such that x is in C, draw an edge between one of the vertices labeled x in the var-cycle and one of the vertices labeled x in the Clause-triangle. Each var-cycle-vertex is used only once. Do the same for $(\neg x, C)$.
3. Output $(G, 5m)$.

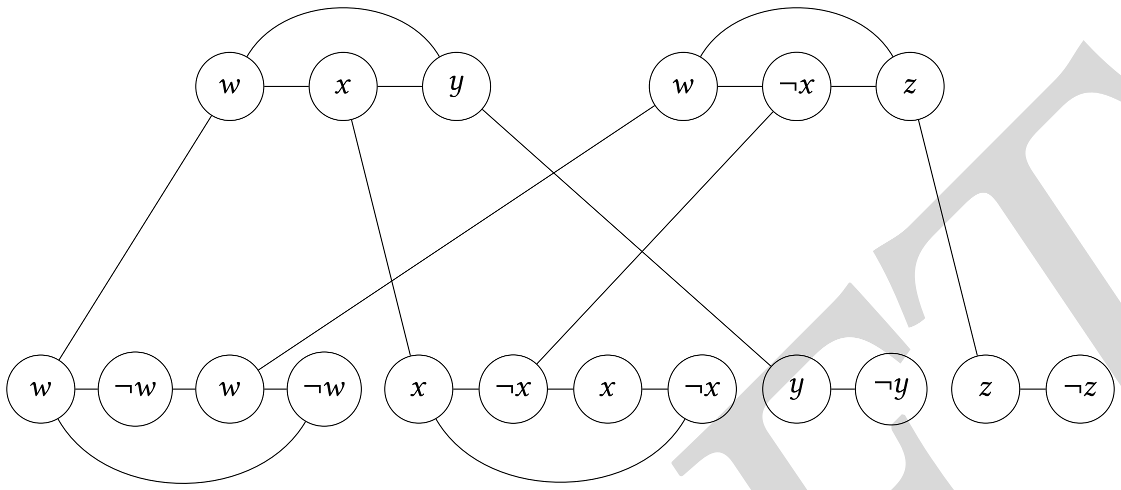


Figure 2.14: Graph from $(w \vee x \vee y) \wedge (w \vee \neg x \vee z)$

We leave the proof that $(G, 5m) \in$ Vertex Cover if and only if $\varphi \in$ 3SAT to the reader.

2) We show that Planar 3SAT $\leq_p$ Planar VC-DEG3. We show that for the reduction in Part 1, if φ is planar then G is planar. The only edges that might cross are the edges from the var-cycles to the clause-triangles. Let G' be the graph obtained by shrinking each var-cycles to one point and shrinking each clause-triangles to one point. We need G' to be planar. It is! G' is the bipartite graph associated to φ. Since φ is planar, G' is planar, so G is planar.

3) Combine
3SAT $\leq_p$ Planar 3SAT with quadratic blowup from Theorem 2.9
with Planar 3SAT $\leq_p$ Planar VC-DEG3 from Part 2.

□

Exercise 2.15. This exercise is about the proof of Theorem 2.14

1. Prove that the reductions in Theorem 2.14 have linear blowup.
2. Prove that the reductions in Theorem 2.14 work.

2.7 Dominating Set

Dominating Set and Planar Dominating Set (Dom Set and Planar Dominating Set)
Instance: A (planar) graph $G = (V, E)$ and a number k. k is the parameter.
Question: Is there a dominating set D of size k. (Every $v \in V$ is either in D or has an edge to some $u \in D$.)

Theorem 2.16.

1. *Vertex Cover $\leq_p$ Dom Set, so Dom Set is NP-complete. The reduction has linear blowup.*

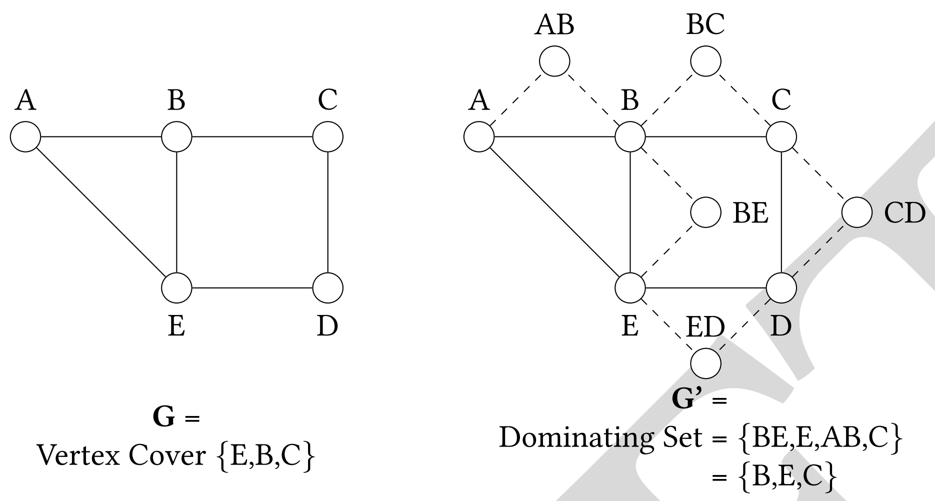


Figure 2.15: Proof that Dominating Set is NP-Complete

2. *Planar Vertex Cover $\le_p$ Planar Dominating Set, so Planar Dominating Set is NP-complete. The reduction has linear blowup.*

3. *3SAT $\le_p$ Planar Dominating Set, so (again) Planar Dominating Set is NP-complete. The reduction has quadratic blowup. (This follows from Part 2 and Theorem 2.14.3.)*

Proof sketch:
We give the reduction for Vertex Cover $\le_p$ Dom Set. Figure 2.15 will be an example of the reduction. It will be easy to see that if the original graph is planar then the resulting graph is planar.

1. Input graph $G = (V, E)$ and number k.

2. Create a graph $G' = (V', E')$ by starting with G, for every edge (u, v) in G create a new vertex uv, and then putting a new edge from uv to both u and v.

See Figure 2.15 for an example. ∎

Exercise 2.17. This exercise is about Theorem 2.16.

1. Prove that the construction works.

2. Prove that the reductions from parts 1 and 2 have linear blowup.

2.8 Planar Directed Hamiltonian Graphs and Variants

Definition 2.18. A ***directed graph*** is a graph where all of the edges have a direction. Formally (1) a graph is a pair $(V, \binom{V}{2})$, and (2) a directed graph is a pair $(V, V \times V)$. We will use a D in front of a graph problem to denote that we are looking at directed graphs.

HAM CYCLE and Variants
Instance: A graph.
Question: Is there a Hamiltonian cycle, which is a cycle that visits every vertex exactly once. One can define the problems of DIRECTED HAM CYCLE (input is a directed graph), HAM PATH (looking for a Hamiltonian Path), DIRECTED HAM PATH, and planar versions of all of these in the obvious way. For the HAM PATH problems the instance also has 2 vertices a, b and you are asking whether there is a Hamiltonian path from a to b.

David Lichtenstein [Lic82] showed that PLANAR DIRECTED HAM CYCLE is NP-complete.

Theorem 2.19.

1. *3SAT $\le_p$ DIRECTED HAM CYCLE hence DIRECTED HAM CYCLE is NP-complete. The reduction has linear blowup.*
2. *PLANAR 3SAT $\le_p$ PLANAR DIRECTED HAM CYCLE, hence PLANAR DIRECTED HAM CYCLE is NP-complete. The reduction has linear blowup.*
3. *3SAT $\le_p$ PLANAR DIRECTED HAM CYCLE, hence (again) PLANAR DIRECTED HAM CYCLE is NP-complete. The reduction has quadratic blowup. (This follows from Part 2 and Theorem 2.9.)*

Proof sketch: We give a reduction 3SAT $\le_p$ DIRECTED HAM CYCLE. The other parts we leave to the reader.

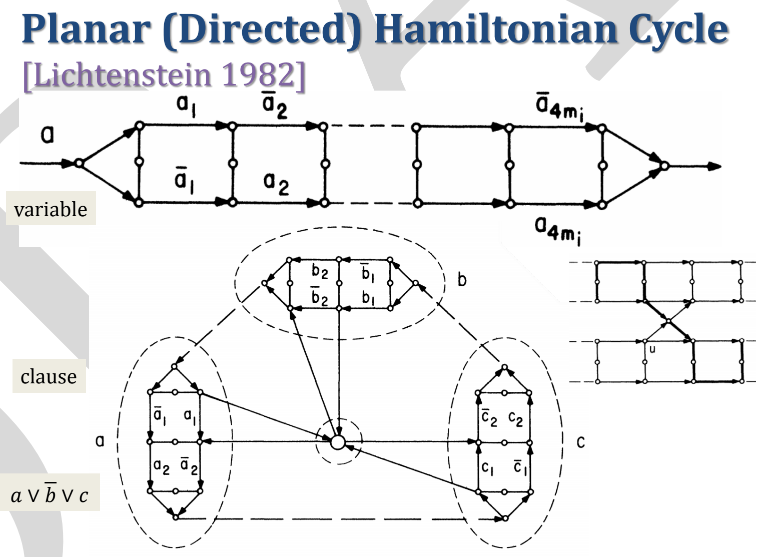


Figure 2.16: Hamiltonian Cycle Gadgets

1. Input a 3CNF formula φ.
2. For each variable a in φ form a graph as in Figure 2.16 (top picture). If m is the number of times a occurs in φ then the gadget has $4m$ rungs. Note that in any directed Hamiltonian cycle either the gadget is entered from the left by going up or going down. This will correspond to setting a to TRUE or FALSE.

3. Let C be a clause. The gadget for C will use the gadgets for the variables in C. If $C = a \vee \neg b \vee c$ then the gadget is portrayed in Figure 2.16 (left picture). From this we leave it to the reader to formally define the reduction.

∎

Exercise 2.20.

1. Complete the reduction in the proof of Theorem 2.19 and show that it works.
2. Show that the reduction given in the proof of Theorem 2.19 results in a graph that is linear in the size of the formula (note that we said *size of the formula* not *number of variables*).
3. Show that if the reduction given in the proof of Theorem 2.19 started with a planar formula then the output would be a planar graph.
4. Show that there is a linear reduction from 3SAT to (1) Directed Ham Path, (2) Ham Path, (3) Ham Cycle.
5. Show that there is a linear reduction from Planar 3SAT to planar versions of the problems in the last part.

2.9 Shakashaka

The following solitaire game was invented by Nikoli [Nik08].

Definition 2.21. The game of Shakashaka begins with a board like the one in Figure 2.17. The goal is to half-fill-in some of the white squares (with constraints we will discuss soon) so that the white squares form a disjoint set of rectangles, as in Figure YYY. Some of the black squares have numbers. A black square with number x will have exactly x half-filled white squares adjacent to it. If a black square has no number, there is no constraint on the number of half-filled white squares adjacent to it. (You can play this game, and others by Nikoli, online [Nik].)

Shakashaka *Instance:* A Shakashaka board. *Question:* Can the player win?

Erik Demaine et al. [DOUU14] showed that Shakashaka is NP-complete.

Theorem 2.22. *Shakashaka is NP-complete.*

Proof. We show Planar 3SAT $\le_p$ Shakashaka.

There are several gadgets needed in the reduction from Planar 3SAT. These gadgets include a wire, clause, and parity shift gadget. The role of the parity shift gadget is to provide 2 possible different configurations which shift the parity in different ways.

□

Shakashaka is NP-complete
[Demaine, Okamoto, Uehara, Uno 2013]

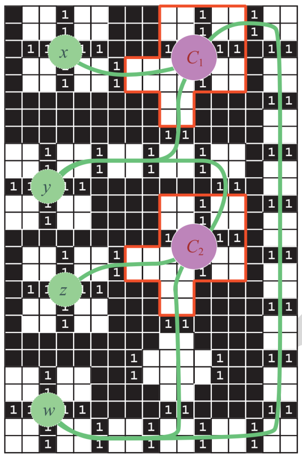


Figure 2.17: Shakashaka Puzzle Encoding a Planar 3SAT Instance Using Variable, Clause and Wire Gadgets

2.10 Planar Rectilinear 3SAT and Planar Rectilinear Monotone 3SAT

While reading these definitions see Figures 2.18 and 2.19.

Definition 2.23.

1. A ***Rectilinear Formula*** is one with the following representation. Each variable is a horizontal segment on the x-axis while each clause is a horizontal segment above the x-axis with vertical connections to the variables it includes on the x-axis. The connections can be either positive or negative.

2. A ***Planar Rectilinear Formula*** is a Rectilinear formula where the graph is planar.

Planar Rectilinear 3SAT
Instance: A Planar Rectilinear 3-CNF formula φ given as a graph.
Question: Is φ satisfiable.

We also look at a variant of Planar Rectilinear 3SAT.

Planar Rectilinear Monotone 3SAT
Instance: A Planar Rectilinear 3-CNF formula φ given as a graph such that (1) all the clauses have either all positive or all negative literals, and (2) the positive clauses are above the x-axis and the negative clauses are below the x-axis.
Question: Is φ satisfiable?

Knuth & Raghunathan [KR92] proved Planar Rectilinear 3SAT is NP-complete. de Berg & Khosravi [dBK12] proved that Planar Rectlinear Monotone 3SAT is NP-complete

Theorem 2.24.

1. *Planar Rectilinear 3SAT is NP-complete.*

2. *Planar Rectlinear Monotone 3SAT is NP-complete.*

Proof sketch: Use the gadgets in Figures 2.18 and 2.19. ▮

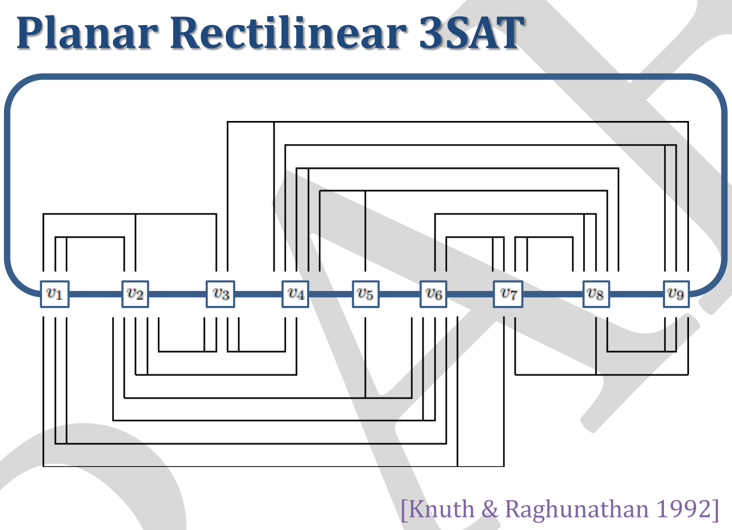


Figure 2.18: Planar Rectilinear 3SAT.

Surgeon General's warning: if all clauses are inside the variable cycle (both positive and negative edges) and the graph is planar, then the problem can be solved in polynomial time. The intuition behind solving this problem is by creating a tree of the nesting structure of the clauses (clauses underneath other clauses) and use dynamic programming on the tree.

A special case to the above is when all clauses are also connected in a path, then the problem is still in P because we can show that this implies that clauses are all in one side of the variable cycle.

This has various implications for hardness proofs. If we can get rid of the connections between clauses, then we don't need the crossover gadget because Super Mario Bros could just be reduced from planar 3SAT. In the case of Super Mario, we can't do this because we can't eliminate the connections between clauses.

Knuth & Raghunathan [KR92] invented Planar Rectilinear 3SAT and showed it was NP-complete. This result was used as a lemma to prove the following problem is NP-complete.

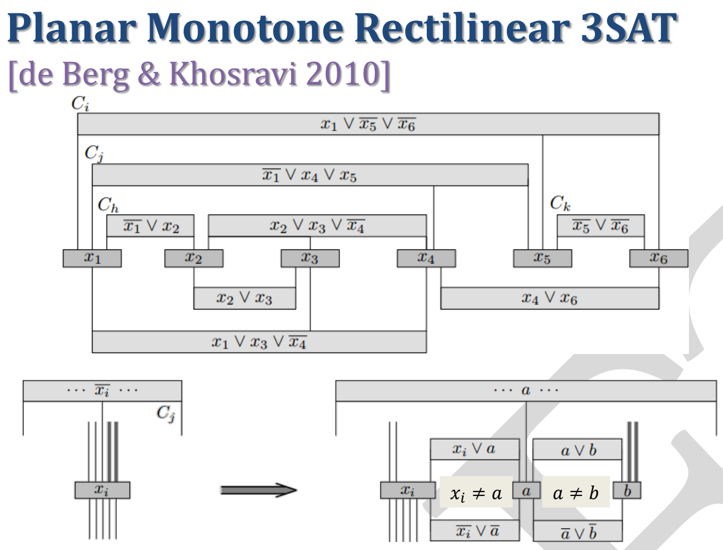

Planar Monotone Rectilinear 3SAT
[de Berg & Khosravi 2010]
C_i
$x_1 \vee \overline{x_5} \vee \overline{x_6}$
C_j
$\overline{x_1} \vee x_4 \vee x_5$
C_h
$\overline{x_1} \vee x_2$
$x_2 \vee x_3 \vee \overline{x_4}$
C_k
$\overline{x_5} \vee \overline{x_6}$
x_1
x_2
x_3
x_4
x_5
x_6
$x_2 \vee x_3$
$x_4 \vee x_6$
$x_1 \vee x_3 \vee \overline{x_4}$
$\cdots \overline{x_i} \cdots$
C_j
x_i
$\cdots a \cdots$
$x_i \vee a$
$a \vee b$
x_i
$x_i \neq a$
a
$a \neq b$
b
$\overline{x_i} \vee \overline{a}$
$\overline{a} \vee \overline{b}$

Figure 2.19: Planar Monotone Rectilinear 3SAT.

MetaFont Labeling
Instance: A set of lattice points in the plane $(p_1, \ldots, p_n)$.
Question: Is there a set of lattice points in the plane $(x_1, \ldots, x_n)$ such that the following hold.

1. For all $1 \le i \le n$, $|x_i - p_i| = 1$.
2. For all $1 \le i < j \le n$, $|x_i - p_j| > 1$.
3. For all $1 \le i < j \le n$, $|x_i - x_j| \ge 2$.

de Berg & Khosravi [dBK12] invented Planar Rectlinear Monotone 3SAT and showed it was NP-complete in order to show that a problem involving binary space partitions of the plane is NP-complete. We omit the formal definition of their problem.

2.11 Planar 1-in-3SAT

Exercise 2.25. Define Planar 1-in-3SAT.

Dyer & Frieze [DF86] defined Planar 1-in-3SAT and showed

Planar 3SAT $\le_p$ Planar 1-in-3SAT.

Hence Planar 1-in-3SAT is NP-complete. They then used this to show that Planar Exact Covering by 3-sets, and Planar 3-Dimensional Matching, are NP-complete.

Theorem 2.26. *Planar 3SAT $\le_p$ Planar 1-in-3SAT, so Planar 1-in-3SAT is NP-complete.*

Proof sketch:
Here is the reduction.

1. Input a planar 3CNF formula φ. We can assume each clause has exactly 3 literals.
2. For each clause $C = L_1 \vee L_2 \vee L_3$ we do the following. First note that the graph of φ is Figure 2.20 (left). Replace this part of the graph with the clauses and variables represented in Figure 2.20 (right).
3. The resulting formula is φ'.

We leave it to the reader to show that φ' is planar and that $\varphi \in$ Planar 3SAT if and only if $\varphi' \in$ Planar 1-in-3SAT. ∎

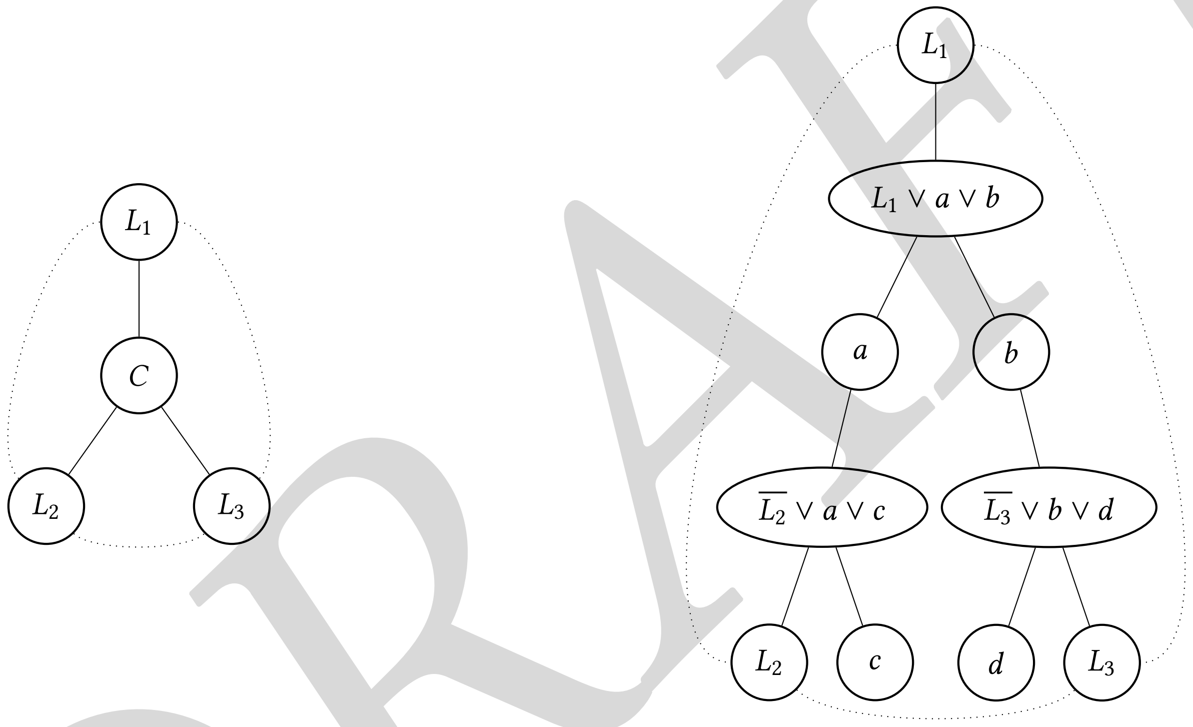


Figure 2.20: Planar 3SAT $\leq$ Planar 1-in-3SAT

Exercise 2.27. Show that the reduction given in the sketch of the proof of Theorem 2.26 works.

2.12 Triangulation

Mulzer & Roe [MR08] defined Positive Planar 1-in-3SAT and implicitly a rectilinear version and showed that it was NP-complete.

Planar 1-in-3SAT
Instance: A planar 3CNF formula where all of the literals are positive. The graph is not just planar, it is rectilinear.
Question: Is there a satisfying assignment where every clause has exactly one variable true?

Theorem 2.28. *Planar 3SAT is reducible to Rectilinear Positive Planar 1-in-3SAT. Hence Rectilinear Positive Planar 1-in-3SAT is NP-complete.*

Proof sketch: Figure 2.21 is the gadget used.

The construction is the same as planar rectilinear 3SAT except we no longer allow negative literals. We can reduce from planar rectilinear 3SAT, removing negations with equal and not all equal gadgets and expanding clauses to exactly three variables.

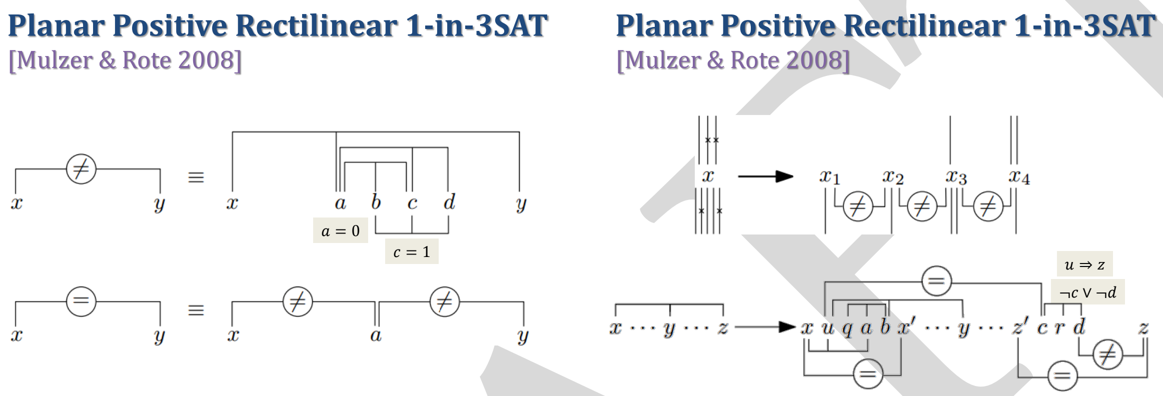


Figure 2.21: Planar Positive Rectilinear 1-in-3SAT Equals and Not-equals Gadgets and Transformation

▮

Exercise 2.29. Complete the proof of Theorem 2.28.

Mulzer & Roe [MR08] used Theorem 2.28 to show the following problem is NP-hard. We omit the proof.

Definition 2.30. Let S be a set of points in the plane. A ***triangulation*** of S is a graph which has S as the set of vertices such that (1) all of the edges are straight lines, (2) the graph is planar, and (3) if any more edges are added then the graph will no longer be planar. (This concept has also been defined in d-dimensions.)

Minimum-Weight Triangulation
Instance: A set S of n points in the plane.
Question: Find a triangulation of S that minimizes the sum of the lengths of the edges.

2.13 Planar NAE 3SAT

From the results in this section one might think that all planar SAT problems are NP-complete. Not so. We need some definitions before we get to our point.

MaxCut) and Planar MaxCut
Instance: For MaxCut a graph $G = (V, E)$ and a $k \in \mathbb{N}$. For Planar MaxCut the graph is planar.
Question: Is there a partition $V = V_1 \cup V_2$ such that the number of edges from V_1 to V_2 is at least k.

MaxCut is NP-complete. In Chapter 10 we will show that even approximating MaxCut is NP-hard. However, Hadlock [Had75] showed that Planar MaxCut is in P. Moret [Mor88] showed that Planar NAE-3SAT can be solved in polynomial time by reduction *to* Planar MaxCut.

The intuition for the method to solve the problem is to make colors alternate between black/white for the maximum cut. Exactly when the variables are not all equal, then we may specify a target with our cut.

2.14 Flattening Fixed-Angle Chains

Molecular geometry (also called ***stereochemistry***) studies 3D geometry of the atoms and the bonds between them that form a molecule. An atom is a vertex and a bond between atoms is an edge. A molecule is a graph but drawn in 3D instead of 2D.

Some molecules can be drawn in 2D. Which ones? We formalize this problem and (as usual) state that it is NP-complete.

Definition 2.31.

1. A ***linkage*** is a weighted graph. A ***configuration*** for all $(u, v) \in E$, $|C(u) - C(v)| = w(u, v)$. A configuration is ***non-crossing*** if two edges intersect only at a vertex. Note that if $d = 1$ then non-crossing is the same as planar.

2. A ***fixed-angle linkage*** is a linkage with an additional constraint: a function Θ that takes every (u, v_i, v_j) where (u, v_i) and (u, v_j) are edges, and returns the angle between them.

3. A ***chain of length n*** is a linkage whose a graph with vertices $V = \{v_1, \ldots, v_n\}$ and edges (v_1, v_2), (v_2, v_3), . . ., (v_{n-1}, v_n).

4. A ***flat state*** of a fixed-angle linkage is a non-crossing 2D configuration of the linkage.

5. FFAC (Flattening Fixed-Angle Chains) is the problem of, given a fixed-angle linkage which is a chain, does it have a flat state.

Erik Demaine & Sarah Eisenstat [DE11] showed that FFAC is NP-hard.

Theorem 2.32. *FFAC is strongly NP-hard.*

Proof sketch:

This problem is strongly NP-hard from a reduction from planar monotone rectilinear 3SAT. The problem is to decide whether a polygonal chain with fixed edge lengths and angles has a planar configuration without crossings.

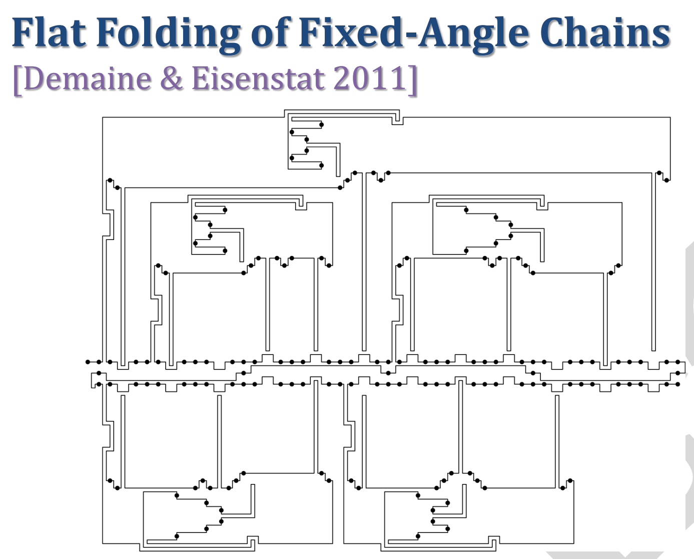

Flat Folding of Fixed-Angle Chains
[Demaine & Eisenstat 2011]

Figure 2.22: Fixed-angle Chain-Flattening Instance Encoding a Planar Monotone Rectilinear 3SAT Instance.

We can create a gadget such that all variables are in a chain. A clause has three possible positions, one each with a protrusion at a point corresponding to a variable, thus forcing the variable chain to take one position or the other. The positions determine the value of the variables within the clause.

∎

2.15 Further Results

We present a list of NP-hard (usually NP-complete) problems that were proven so using a known NP-complete planar problem.

- ***Corral*** is a paper-and-pencil puzzle involving a grid of squares where some of the squares contain natural numbers. Let CORRAL be the problem of, given an initial configuration of the Corral game, is there a way to win. Friedman [Fri02a] showed that PLANAR 3COL $\le$ CORRAL.

- ***PLANAR k-MEANS***: Given a finite set $S = \{p_1, p_2, \ldots, p_n\}$ of points with rational coordinates in the plane, an integer $k \ge 1$, and a bound $R \in \mathbb{Q}$ determine whether there exists k centers $\{c_1, \ldots, c_k\}$ in the plane such that

$$\sum_{i=1}^{n} \left(\min_{1 \le j \le k} [d(p_i, c_j)]^2 \right) \le R.$$

  ($d(p, c)$ is the Euclidean distance from p to c.) Mahajan et al. [MNV12] showed

  PLANAR 3SAT $\le$ PLANAR k-MEANS,

  so the problem is NP-hard, though it is not known to be in NP.

- ***Multi-Robot Path Planning Problems on Planar graphs***: Given a planar graph, robots (start vertices), and destinations (end vertices), and a number k, is there a set of paths along the graph such that no two paths lead to a collision with arrival time $\leq k$? Yu [Yu16] showed this problem is NP-hard using a reduction from Monotone PLANAR 3SAT. For another problem from robotics that is proven hard by a reduction from PLANAR 3SAT see Agarwal et al. [AAGH21].

- ***1-in-Degree Decomposition***: Given graph $G = (V, E)$ is there a partition $V = A \cup B$ such that every $v \in V$ has exactly one neighbor in B? Dehghan et al. [DSA18] showed (1) if G is restricted to graphs with no cycle of length $\equiv 2 \pmod 4$ then the problem is in P, (2) if G is restricted to r-partite graphs where $r \geq 3$ then the problem is NP-complete. They use Planar 1-in-3 SAT and a monotone version of it. They have other hardness results as well.

- ***Tracking paths problem***: Given $G = (V, E)$, a source $s \in V$, a destination $t \in V$, and a number $k \in \mathbb{N}$, does there exist $U \subseteq V$, $|U| \leq k$, such that the intersection of U with any $s - t$ path results in a unique sequence. Eppstein et al. [EGLM19] proved this problem is NP-complete even in the case of planar graphs using a reduction from PLANAR 3SAT.

- There is a myriad of other variants of PLANAR 3SAT. Tippenhauer's thesis [Tip16] looks at versions where the number of variables are bounded and the formulas are monotone. Filho's thesis [Fil19b] gives an extensive survey and introduces more clause variants. Linked SAT [Pil18] is a particularly powerful version where an added cycle through all variables and then all clauses is guaranteed to preserve planarity.

Chapter 3

NP-Hardness via Circuit SAT

3.1 Introduction

In this chapter, we describe some NP-hardness reductions from Circuit SAT. Recall the problem as defined in Section 1.2.1:

Circuit SAT
Instance: A Boolean circuit $C(x_1, x_2, \ldots, x_n)$ with n inputs.
Question: Does C ever output TRUE? That is, does there exist $(b_1, b_2, \ldots, b_n) \in \{\text{TRUE}, \text{FALSE}\}^n$ such that $C(b_1, b_2, \ldots, b_n) = \text{TRUE}$?

3.2 How to Reduce from Circuit SAT

In contrast to reductions from SAT as described in Section 1.3, Circuit SAT reductions typically fall into one pattern. Most obviously, we need a ***gate gadget*** to represent each type of gate. It suffices to have one gadget for each gate in any ***functionally complete*** set of gates, for example:

- {NAND},
- {NOR},
- {AND, NOT},
- {OR, NOT},
- $\{\rightarrow, F\}$ (where $x \rightarrow y$ evaluates to $\neg x \vee y$ and the F gate always outputs FALSE).
- Any set given by Post's Functional Completeness Theorem.

Next we need a ***wire gadget*** to connect the output of one gate to the input of another gate. Typically, each gate gadget only has one copy of its output so, like binary-logic reductions from 3SAT, we need a ***split gadget*** for duplicating a gate output (given by one wire) so that we can route it to every gate input where it is needed (via additional wires).

To represent the inputs, we need a ***terminator gadget*** that ends a wire without constraining the wire. Thus the wire is free to choose whether it carries a value of TRUE or FALSE, just like an

x_i input in the CIRCUIT SAT problem. To represent the desire of the final output to be TRUE, we need a ***TRUE terminator gadget*** that ends a wire and forces its value to be TRUE. All the gadgets can be satisfied if and only if the circuit is satisfiable.

Exercise 3.1. Read Erich Friedman's paper [Fri02c] on the NP-completeness of the Spiral Galaxies problem. The proof uses a reduction of CIRCUIT SAT. Rewrite the reduction in your own words.

3.2.1 PLANAR CIRCUIT SAT

PLANAR CIRCUIT SAT is the following problem. Note that this terminology is (surprisingly) not used often.

PLANAR CIRCUIT SAT
Instance: Boolean circuit $C(x_1, x_2, \ldots, x_n)$ represented by a planar directed acyclic graph. Each node in the graph is either a source (of indegree zero, representing an input), a NAND gate (of indegree 2 and any outdegree), or a sink (outdegree zero, representing the output). There must be exactly one sink.
Question: Does C ever output TRUE? That is, does there exist $(b_1, b_2, \ldots, b_n) \in \{\text{TRUE}, \text{FALSE}\}^n$ such that $C(b_1, b_2, \ldots, b_n) = \text{TRUE}$?

Equivalently, the question is whether each edge (wire) of the graph can be labeled with a value of either TRUE or FALSE such that the following hold:

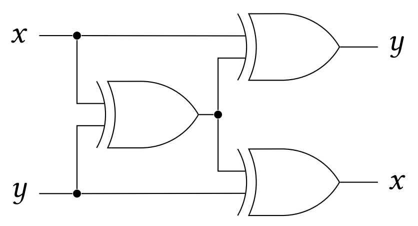
- The value of the edge into the sink is TRUE.
- The value of any edge out of a NAND gate is the NAND of the truth values of the two edges going into the gate.
- (The value of an edge out of a source can be anything.)

Theorem 3.2. *PLANAR CIRCUIT SAT is NP-complete. The problem is also NP-complete if we replace the word NAND with the word NOR everywhere.*

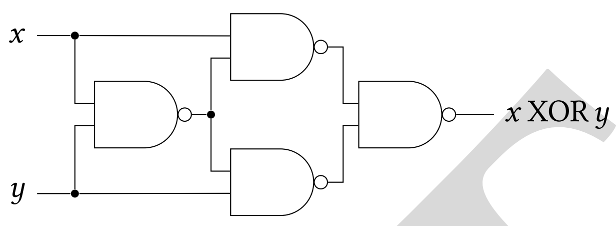
Proof. We prove NP-hardness by a reduction from CIRCUIT SAT, using the crossover gadgets in Figure 3.1. The base gadget (a) is built out of XOR gates. Each XOR gate can be replaced by a subcircuit of NAND gates (b) to prove PLANAR CIRCUIT SAT hard, and each NAND gate can be replaced by a subcircuit of NOR gates (c) to prove the NOR analog hard. Crucially, all of these subcircuits have no crossings. Hence we can remove all crossings by replacing each crossing with a crossover gadget of the appropriate type. □

3.3 Puzzles

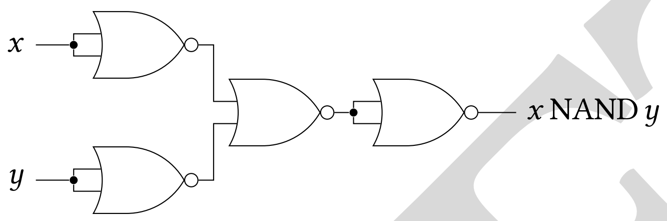
Next we will discuss the hardness of several puzzles.



(a) Crossover gadget built out of XORs.



(b) XOR gadget built out of NANDs.



(c) NAND gadget built out of NORs.

Figure 3.1: Crossover gadgets for Circuit SAT.

3.3.1 Light Up

An instance of the puzzle ***Light Up*** (also called ***Akari***) consists of a grid of white and black squares where some black squares have a number between 0 and 4 written on them. The goal of the puzzle is to place lights on the white squares of the board so that the following rules are satisfied:

- Each black square which has a number written on it must have that number of lights in a horizontally or vertically adjacent square.
- Each white square should be lit up by exactly one light. A light lights up a square if both the light and the square are in the same row or column and if there are no black squares between them.

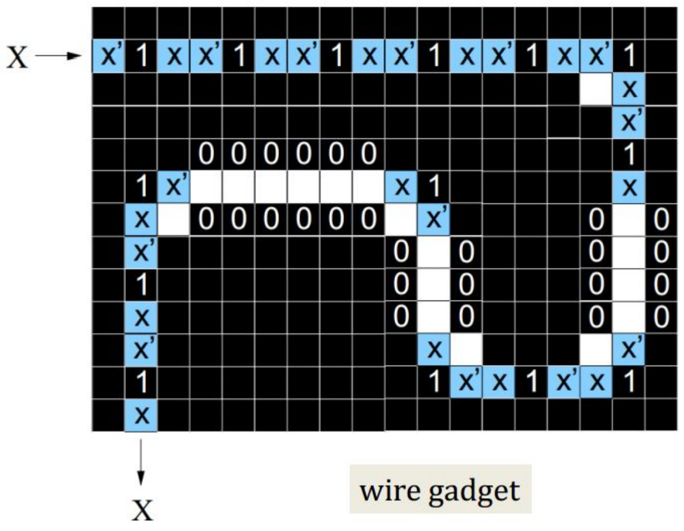
There have been entire puzzle books on this game, for example, that of Nikoli [Nik08]. You can play it online here:
`https://www.puzzle-light-up.com/`

Definition 3.3. Light Up is the problem of, given a board position in the game Light Up, can lights be placed on it to attain the goal.

Brandon McPhail [McP05] has shown that Light Up is NP-complete.

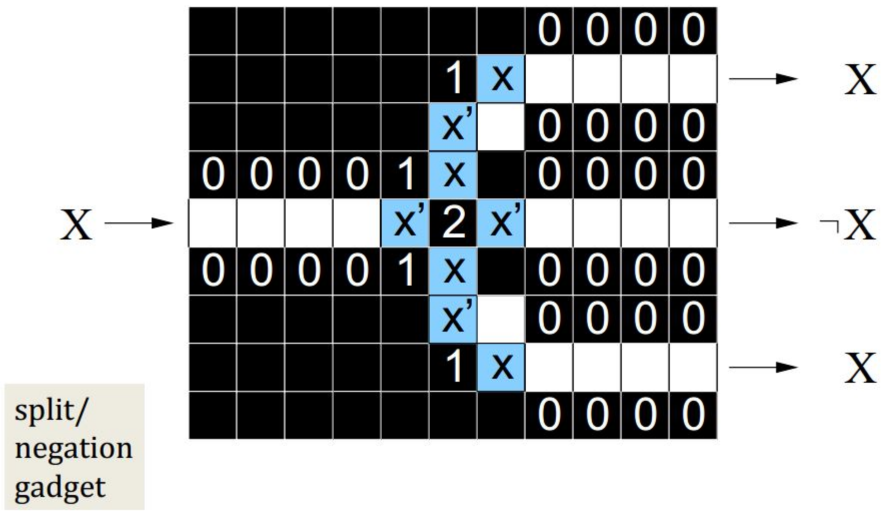
Theorem 3.4. *Light Up is NP-complete.*

Proof sketch: The hardness proof is a reduction from Planar Circuit SAT (NOR variation). Like most such proofs, the reduction starts with the wire gadget. A basic wire gadget consists of a repeating sequence of a black square with a 1 followed by two white squares. This forces exactly one of the two white squares to have a light in a way such that the choice of using the white square on the right or on the left propogates down the wire. Alternatively, by lining a corridor



with 0's, one can have a different wire because the light must be at one of the ends of the corridor. Making turns is also easy. All of these gadgets are depicted below.

The negation/split gadget below outputs a single negated copy and two regular copies of the incoming wire. By using a terminator gadget, you can use this gadget as either the split gadget or the not gadget.

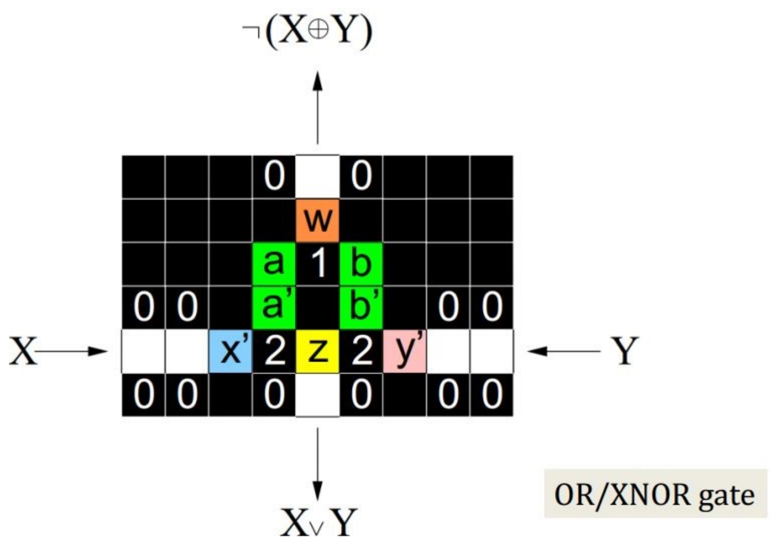


Similarly, the next gadget behaves as both an OR gate (and an XNOR gate). Showing that this is true requires simply going through all the cases.

Together, the OR and NOT gates create a NOR gate.

The above gadgets, together with a terminator gadget (which requires simply placing a 0 or 1 at the end of a wire to set its value), are sufficient to prove that LIGHT UP is NP-complete. ▮

Exercise 3.5. Formalize the proof of Theorem 3.4 and construct some (very small) examples of the reduction.



3.3.2 Minesweeper

Minesweeper is the following puzzle computer game, particularly famous for being included in most versions of Microsoft Windows:

1. The parameters are n, m (the board will be $n \times m$) and b (the number of bombs).

2. The player initially sees an $n \times m$ board of unlabeled spaces.

3. b of the spaces have hidden bombs. Some of the other spaces have numbers which indicate how many bombs are adjacent to that space. Here adjacent means up, down, left, right, or diagonal. Hence most spaces have 8 spaces adjacent to them.

4. The player will either left-click or right-click on a space.

   (a) If a player left clicks and the space has a bomb, the game is over and the player loses.

   (b) If a player left clicks and the space has a number, that number is revealed and the player may use that to try to figure out where bombs are.

   (c) If a player left clicks and the space has nothing then that space does not have a bomb, nor do any the adjacent spaces. Those adjacent spaces are also uncovered.

   (d) If a player right clicks then a flag is put on that space. The player does not know whether there was a bomb or not. A player is only allowed to do this b times.

5. If the player reveals all non-bomb space, then she wins. If a player gets blown up, then she loses.

6. Caveat: in some versions, the first space clicked on cannot have a bomb. There are various ways to implement this rule.

Given a grid where some spaces have numbers, is it ways possible to put bombs on the grid so that the numbers are consistent? No. Consider the following grid:

		1		
1		5		1
		1		

Figure 3.2: An Impossible Configuration

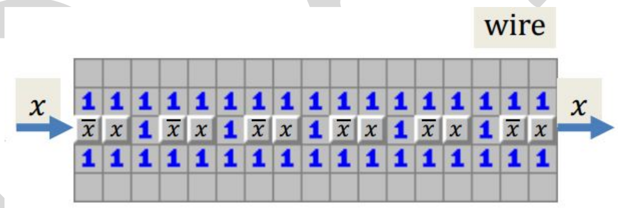
Exercise 3.6. Prove that there is no way to plant bombs such that Figure 3.2 is correct.

The Minesweeper Consistency problem asks whether a given Minesweeper board is possible. The motivation for this problem is that if we could solve the problem efficiently then placing a bomb in a location and then checking whether the board is consistent allows a player to check whether that location is safe to click. One of the problems with this approach to playing Minesweeper is that there might not be any safe moves. As a result, this isn't the best problem to pose, but it is the first one to be proved hard.

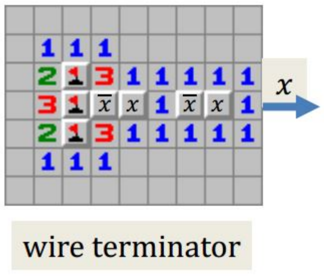
Brandon Kaye [Kay00] proved that the Minesweeper Consistency is NP-complete.

Theorem 3.7. *Minesweeper Consistency is NP-complete.*

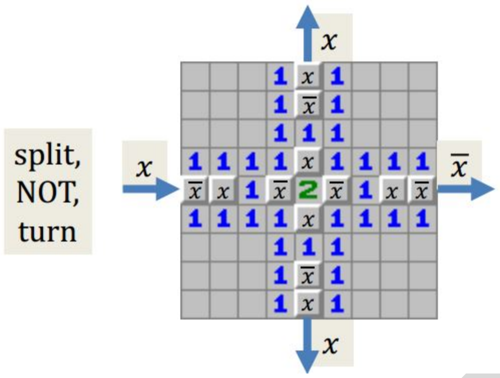
Proof. Reduction from Planar Circuit SAT (NAND variation). The wire gadget for Minesweeper looks a lot like the Light-Up wire. The bombs must all be placed in the right square or the left of the pairs of unspecified squares.



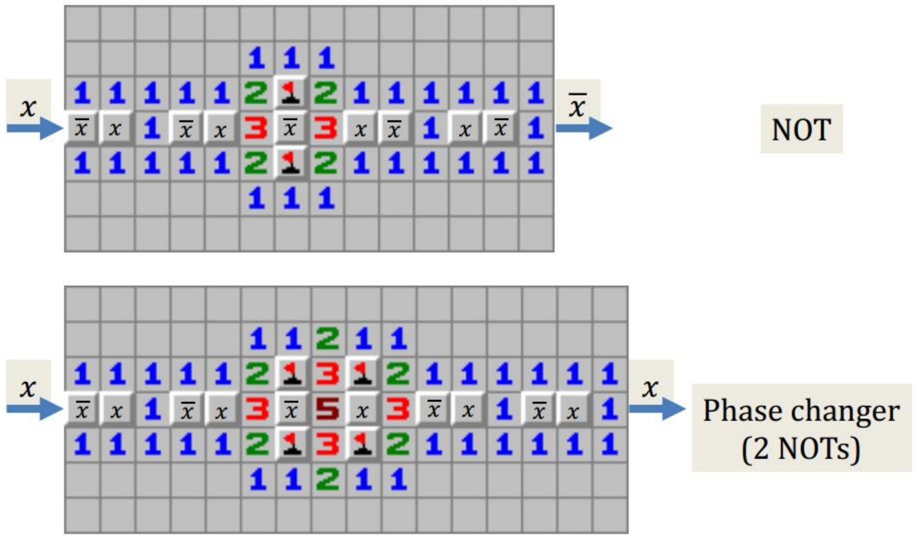
The terminator gadget for Minesweeper is more complicated than the Light-Up terminator because simply stopping a wire would require specifying that the last square has a bomb, which would not leave the wire unspecified.



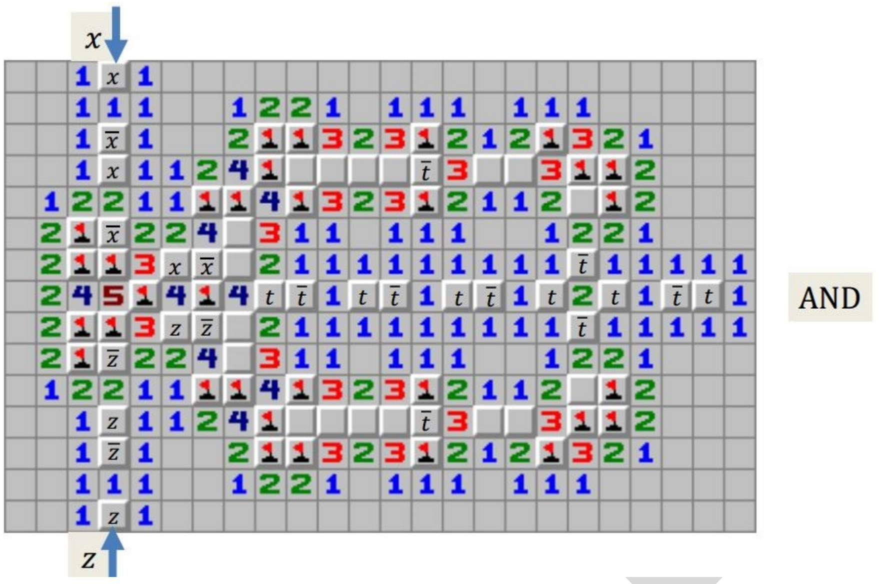
The following gadget is a split, NOT, and turn gadget all in one. Each individual gadget can be extracted from this design by adding terminators to the unnecessary wires.



The simpler NOT gate below can be used twice in order to fix the "mod-3-parity" issue (that wires are always a multiple of 3 length).



The AND gadget looks very complicated, but all one has to do to check it is check all 4 cases.



Together, the AND and NOT gates create a NAND.

Building a true terminator is easy (by just ending the wire with a 1), so the above gadgets are enough to prove the problem is NP-hard. □

While Theorem 3.7 is interesting, it's not quite the right question. A player wants to know what their next move should be. Also note that the number of bombs used in each gadget varies according to the truth values of the variables. In actual Minesweeper, the number of bombs is specified, so that is another problem.

Definition 3.8. A ***Minesweeper position*** is a grid, a number b of bombs, some of the grid squares are uncovered and have a number (the number of bombs around it) and some have a flag (indicating that there is a bomb there). Note that the player is promised that there is a consistent way to place bombs.

For most of the problems in this book there are no preconditions on the input. You can be given *any* formula or graph. For Minesweeper we will need to promise the player that the game is consistent. We take this opportunity to define promise problems.

Definition 3.9. A ***promise problem*** is two sets (A, B). A is the set we care about and B is the promise. A promise problem is in P if there is a polynomial-time algorithm M that does the following:

1. If $x \in B$ and $x \in A$ then $M(x) = 1$.
2. If $x \in B$ and $x \notin A$ then $M(x) = 0$.
3. If $x \notin B$ then $M(x) \in \{0, 1\}$

Definition 3.10. The ***Minesweeper Inference*** problem is the following: given a position in Minesweeper that you are promised is consistent, determine whether there is a square such that one can deduce with certainly either (a) there is a bomb there, or (b) there is no bomb there. Note that you may deduce that there is no such square.

Exercise 3.11. Show that, if Minesweeper Inference ∈ P, then there is a strategy for Minesweeper that makes the best move possible (note that a player can still lose).

To prove that Minesweeper is hard to play we want to prove that Minesweeper Inference is NP-complete. But there is one problem with that approach: Minesweeper Inference does not seem to be in NP. For other games we showed that playing them was NP-hard and the problem of membership in NP is open. Here we can obtain a finer classification. Allan Scott, Ulrike Stege, and Iris van Rooij [SSvR11] showed that Minesweeper Inference is coNP-complete.

Theorem 3.12. *Minesweeper Inference is coNP-complete.*

Proof. To show Minesweeper Inference ∈ coNP we show that its complement is in NP.

Let M be a game position (promised to be consistent) that is not in Minesweeper Inference. A witness to $M \notin$ Minesweeper Inference is the following: for every grid space where it is unknown if there is a bomb or not, give two consistent completions of the grid: one where there is a bomb, one where there is not.

We now show that Minesweeper Inference is coNP-hard. To prove that Minesweeper is coNP-hard we reduce from $\overline{\text{SAT}}$.

In this reduction, we make sure that all of our gadgets have the same number of bombs regardless of variable assignment.

The wire gadget stays the same, but many of the other gadgets get much more cluttered. This time we use an OR gate which, together with NOT, gives us the desired NOR gate.

The final output of the circuit is connected to a terminal gadget that leaves it undetermined rather then setting the output to true as we would in a SAT reduction. The reason is as follows: if the unsatisfiable instance is always false then there will always be a bomb in the end of the circuit; if some assignment exists that makes the output true then there exists some solution to the Minesweeper instance that has no bomb there. Thus, since it is hard to determine whether a formula is in UNSAT it is also hard to determine whether a bomb must exist in a particular location on the Minesweeper board. □

Chapter 4

NP-Hardness via Hamiltonian Cycle

4.1 Overview

We discuss the Hamiltonian cycle and path problems, with an emphasis on grid graphs, and use these problems to prove some NP-hardness results for games and lawn mowing. We will then informally discuss some Metatheorems about proving games NP-hard.

A ***Hamiltonian cycle*** (also ***tour***, ***circuit***) is a cycle visiting each vertex exactly once. Graphs are said to be ***Hamiltonian*** if they contain a Hamiltonian cycle. A ***Hamiltonian path*** is a path visiting each vertex exactly once. The decision problems ask whether a Hamiltonian cycle or path exists in a given graph.

Hamiltonicity is named after William Rowan Hamilton, an Irish mathematician, who studied Hamiltonian cycles on the dodecahedron. Hamilton commercialized his study as the Icosian Game (so named because the dodecahedron is the dual of the icosahedron).

Definition 4.1. Let GRID be the graph with vertices $\mathbb{N} \times \mathbb{N}$ and edges all pairs of the form either $((a, b), (a + 1, b))$ or $(a, b), (a, b + 1))$. A ***grid graph*** is any subgraph of GRID. For this book we only consider finite grid graphs.

In this chapter we do not give acronyms for problems since most problems are mentioned only once and the acronyms would be too long. For example ***Hamiltonian Cycle Restricted to Degree-3 Planar Graphs*** would be . . . well, you figure it out. But note that it's long!

4.2 Variants of Hamiltonian Cycle

We repeat the definition of Hamiltonian Cycle and some variants for completeness.

Ham Cycle and Variants
Instance: A graph.
Question: Is there a Hamiltonian cycle, which is a cycle that visits every vertex exactly once. One can define the problems of Directed Ham Cycle (input is a directed graph), Ham Path (looking for a Hamiltonian Path), Directed Ham Path, and planar versions of all of these in the obvious way. For the Ham Path problems the instance also has 2 vertices a, b and you are asking whether there is a Hamiltonian path from a to b.

The following NP-complete results are briefly stated here without proof.

- Ham Cycle is NP-complete. This was one of Karp's original 21 problems shown to be NP-complete [Kar72].
- Directed Ham Cycle, Ham Path, Directed Ham Path are all NP-complete.
- We proved that Planar Directed Ham Cycle is NP-complete in Theorem 2.19.
- Ham Cycle restricted to planar 3-regular 3-connected graphs with minimum face degree 5 is NP-complete. This was proven by Garey et al. [GJT76].
- Ham Cycle restricted to bipartite graphs is NP-complete. This was proven by Krishnamoorthy [Kri75].
- If G is a graph then ***the square of G*** is the graph with edges added between all vertices connected by a path of length 2. Ham Cycle on squares of graphs is NP-complete. This was proven by Underground [Und78].
- Ham Cycle where you are also given a Hamiltonian path in the graph is NP-complete. This was proven by Papadimitriou & Steiglitz [PS76]. In other words, knowing a path does not in general help when finding a cycle.

Here are some trivial instances of Hamiltonian Cycle; however, proving they are trivial is difficult.

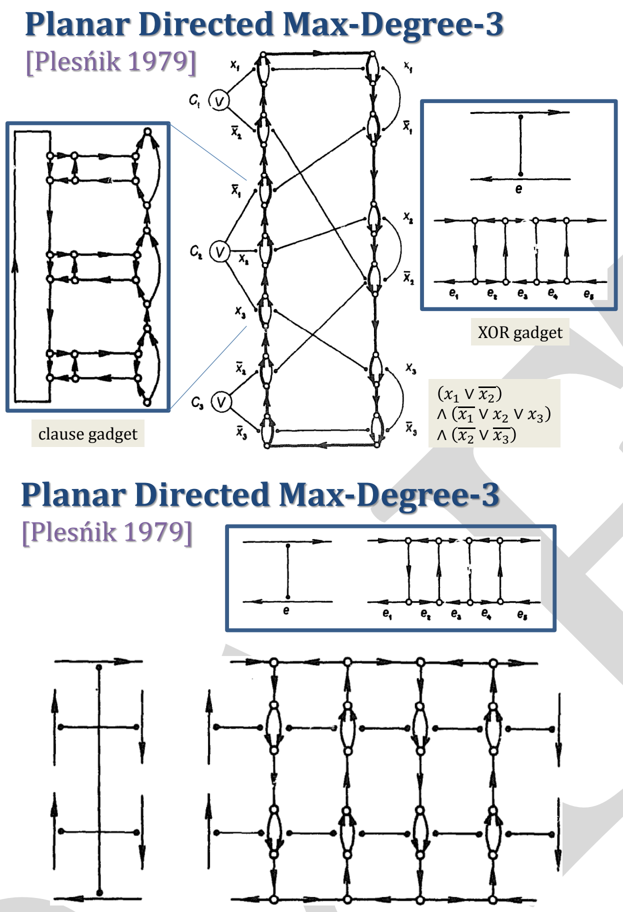
- Tutte [Tut56] proved that all planar 4-connected graphs are Hamiltonian. Hence the problem restricted to planar 4-connected graphs is trivial: just say yes.
- Karaganis [Kar68] showed that the cubes of a graph (the graph with edges added between all vertices connected by a path of length 3 or less) is always Hamiltonian. Hence the problem restricted to graphs that are cubes of other graphs is trivial: just say yes.

Note that Tutte proved his result in 1956, and Karaganis proved his result in 1968, before the notion of NP-complete was known.

4.3 Planar Graphs with Max-Degree-3: Directed and Undirected

Plesník [Ple79] showed the following:

Theorem 4.2. *The Hamiltonian cycle problem restricted to planar directed max-degree-3 graphs is NP-complete*



Planar Directed Max-Degree-3
[Plesńik 1979]
clause gadget
XOR gadget
e
e_1 e_2 e_3 e_4 e_5
C_1 C_2 C_3
$(x_1 \vee \overline{x_2})$
$\wedge (\overline{x_1} \vee x_2 \vee x_3)$
$\wedge (\overline{x_2} \vee \overline{x_3})$
Planar Directed Max-Degree-3
[Plesńik 1979]

Figure 4.1: Gadgets for Proving Ham Cycle on Planar Max-Degree-3 Graphs is NP-Complete

Proof. For all gadgets see Figure 4.1. This is a reduction from 3SAT. The reduction relies on an XOR gadget used to enforce that exactly one of two edges is in the cycle. Variables are then represented as pairs of doubled edges linked by a XOR gadget, forcing the pairs to have opposite membership, representing true and false assignments. The clause gadget contains a large cycle which can be in the Hamiltonian cycle only if one of the incoming literals (connected to the clause by XOR gadgets) is true. This reduction also requires a crossover gadget to allow the XOR gadgets connecting variables to clauses to cross; the crossover gadget itself uses the XOR gadgets, essentially exploding them. □

Itai et al. [IPS82] showed the following:

Theorem 4.3. *The Hamiltonian cycle problem restricted to planar undirected bipartite max-degree-3 graphs is NP-complete*

Proof. The proof is by reduction from the corresponding directed graph problem. (See Figure 4.1 for guidance.) Given a max-degree-3 directed graph G first check if any vertex has indegree 3 or outdegree 3. If so then G cannot have a Hamiltonian cycle and we are done (formally output a undirected bipartite max-degree 3 graph that does not have a Hamiltonian cycle). Henceforth we can assume that every vertex has either in-degree 1 or outdegree 1. Look at a vertex v. If

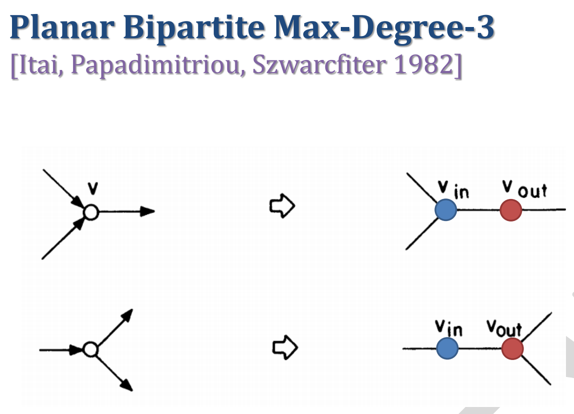

Planar Bipartite Max-Degree-3
[Itai, Papadimitriou, Szwarcfiter 1982]
v
v in
v out
v in
v out

Figure 4.2: Reduction from Planar Directed Degree-3 Ham to Planar Undirected Bipartite Degree-3 Ham.

it has in-degree 1 (out-degree 1) then the edge coming in to v (going out of v) must be in any Hamiltonian cycle of G. Hence we can map each vertex in the directed graph to a pair of vertices in the undirected graph. Any Hamiltonian cycle in the resulting graph will alternate between vertices from the original graph and vertices introduced to model forced edges, so the resulting graph is bipartite. □

4.4 Hamiltonicity in Grid Graphs and Applications

Definition 4.4.

1. ***Grid graphs*** are graphs with vertices on a (subset of a) lattice and edges between all pairs of vertices with unit distance. Unqualified, grid graphs are usually on the square lattice, but the triangular and hexagonal lattices can also be considered.

2. Faces of grid graphs of unit area are called ***pixels***, while faces with greater area (i.e., containing at least one lattice point that is not a vertex) are called ***holes***. ***Solid*** grid graphs have no holes.

3. A ***solid grid graph*** is one where every bounded region is 1×1. See Figure 4.3 for an example and a counterexample.

Umans & Lenhart [UL97] showed that Hamiltonian cycle on solid grid graphs is polynomial time.

By contrast, Itai et al. [IPS82] proved the following:

Theorem 4.5. *The Hamiltonian cycle problem restricted to grid graphs containing holes is NP-complete*

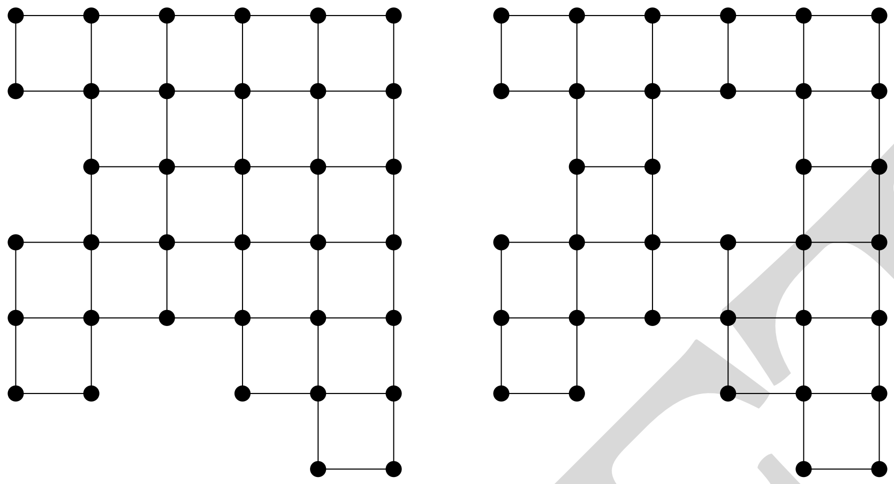


Figure 4.3: Left: A Solid Grid Graph; Right: a Non-Solid Grid Graph

Proof sketch:

The proof is by reduction from Hamiltonian cycle on planar undirected bipartite max-degree-3 graphs. The reduction first embeds the input bipartite graph on a grid graph using the grid graph's inherent 2-coloring. Any parity violations can be sidestepped by scaling the grid graph by a factor of 3, allowing "wiggles" to be added as required. (See Figure 4.4, both parts.)

The Hamiltonian cycle on the input graph need not use every edge, but the Hamiltonian cycle on the output grid graph needs to visit each vertex, so our edge gadget can be traversed in two ways, a "zigzag" corresponding to using the edge (a Hamiltonian path from one end of the gadget to the other) and a border traversal corresponding to not using the edge (a Hamiltonian path that returns to the same end of the gadget).

The vertex gadget is a 3-by-3 square of vertices, the corners of the gadget having the parity of the corresponding vertex in the input graph. This gadget has Hamiltonian paths from every corner to every other corner that traverse four marked edges. Edge gadgets connect to white vertices flush with one of the corners, but connect to black vertices offset by one (a ***pin joint***); this arrangement allows the edge to be used or not while preserving the parity of the cycle. (See Figure 4.5)

∎

Definition 4.6. Given a set of points in the plane, the ***Euclidean Traveling Salesman Problem (ETSP)*** asks for a tour through the points with Euclidean length less than k (or in the optimization problem, with minimum length).

Theorem 4.7. *The ETSP problem is NP-complete.*

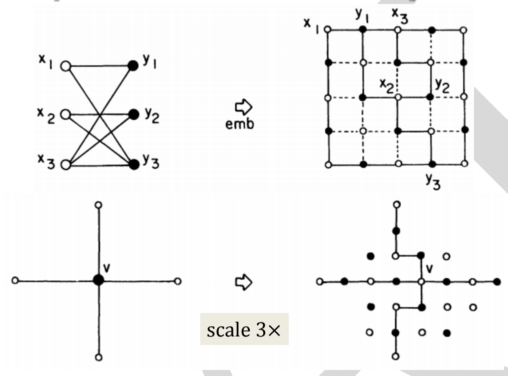
Proof. We showing that Hamiltonian Cycle on Grid Graphs is reducible to ETSP.

1. Input G, a grid graph. We can assume it is in the plane and the vertices are lattice points. Let n be the number of vertices in G.

2. The instance I of ETSP is just the points in the grid graph with the goal of having a Hamiltonian cycle of length n.

Planar Bipartite Graph Drawing

[Itai, Papadimitriou, Szwarcfiter 1982]



Hamiltonicity in Grid Graphs

[Itai, Papadimitriou, Szwarcfiter 1982]

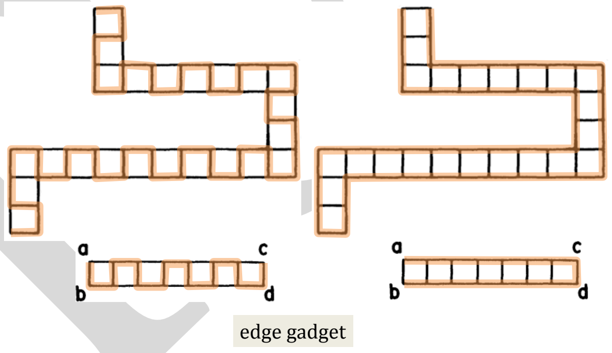
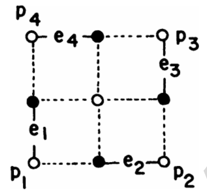


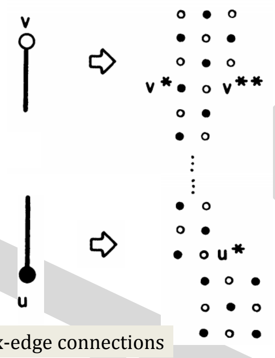
Figure 4.4: Planar Bipartite Graph Drawing

Hamiltonicity in Grid Graphs

[Itai, Papadimitriou, Szwarcfiter 1982]



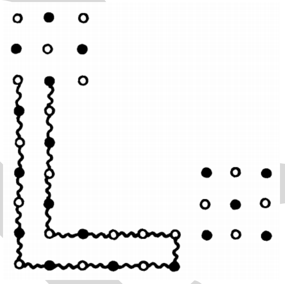
vertex gadget



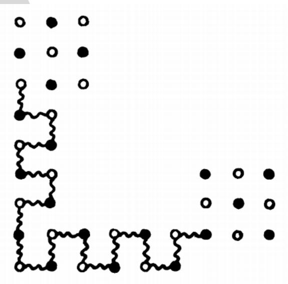
vertex-edge connections

Hamiltonicity in Grid Graphs

[Itai, Papadimitriou, Szwarcfiter 1982]



unused edge gadget



used edge gadget

Figure 4.5: Hamiltonian Grid Graph

Any tour that visits all the points in I must be a cycle in G since if the tour visits two non-adjacent points it will not have cost $\leq n$. □

Forišek [For10] proved the following:

Theorem 4.8. *Platform games whose objective is to collect all collectibles (e.g., coins) in a level before a timer expires are NP-hard.*

Proof sketch:

The proof is by reduction from Hamiltonian cycle on grid graphs. The reduction simply places coins at all the grid graph vertices and sets the timer such that moving between any nonadjacent pair of points (i.e., not taking the minimum tour) will result in the timer expiring. ▮

4.5 Max-degree-3 Grid Graphs and Applications

Papadimitriou & Vazirani [PV84] proved the following:

Theorem 4.9. *Hamiltonian cycle restricted to max-degree-3 grid graphs is NP-complete.*

Proof. This is by a similar reduction for unconstrained grid graphs, with the gadgets modified to avoid degree-4 vertices. The edge gadget gains holes at corners, but has the same topology with two configurations. Vertex gadgets are now "dumbbells". Degree-2 vertices have their edge gadgets connected to opposite ends of the dumbbell; degree-3 vertices have the forced edge connected to both ends of the dumbbell via a fork gadget. (This fork gadget is required to preserve parity.) □

The minimum spanning tree problem (MST) is in polynomial time by either Kruskal or Prim's algorithm. However, there are variants that are NP-complete.

Theorem 4.10. *MST where we require the tree have degree 2 is NP-complete.*

Proof. This is just the TSP problem. □

What about degree 3? Papadimitriou & Vazirani [PV84] proved the following:

Theorem 4.11. *MST restricted to Euclidean grid graphs, where we require the tree have degree 3, is NP-complete*

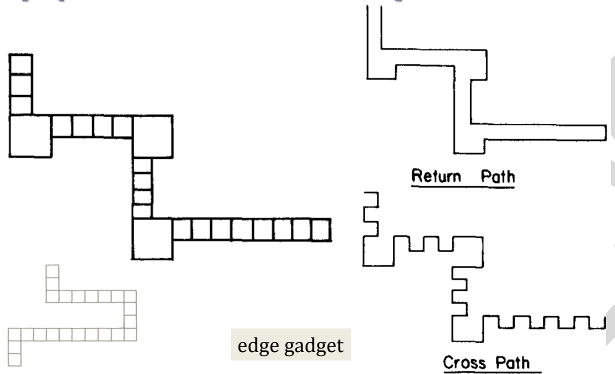
Proof. Euclidean degree-3 minimum spanning tree is still hard by reduction from Hamiltonian path on max-degree-3 grid graphs. The reduction adds new vertices very close to each existing vertex in the grid graph, forcing the edge between that vertex and the new vertex to be in the minimum spanning tree. The remainder of the tree is then finding a Hamiltonian path in the grid graph. □

4.6 Grid Graph Hamiltonicity Taxonomy

Superthin grid graphs have no pixels; all faces are holes or the outside face. ***Thin*** grid graphs have all vertices on the boundary. ***Polygonal*** grid graphs have no shared edges between holes or the outside face; polygonal graphs could be considered anti-superthin. With these definitions we taxonomize Hamiltonicity over various kinds of graphs in Figure 4.8.

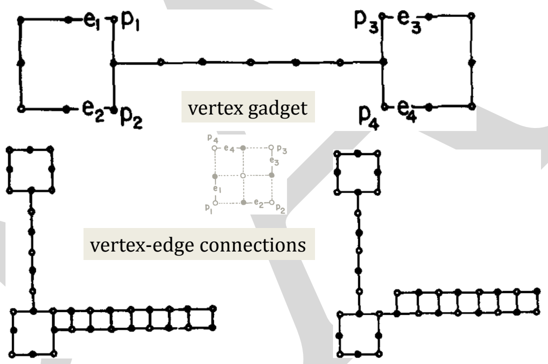
Max-Degree-3 Grid Graphs

[Papadimitriou & Vazirani 1984]



Max-Degree-3 Grid Graphs

[Papadimitriou & Vazirani 1984]



Max-Degree-3 Grid Graphs

[Papadimitriou & Vazirani 1984]

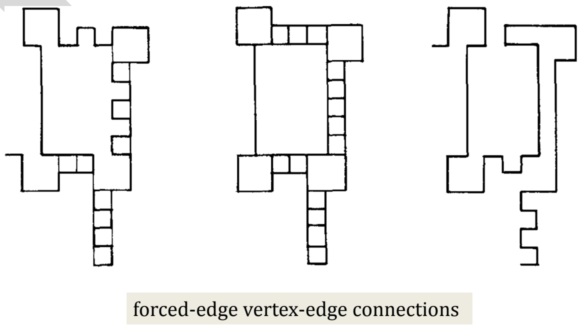


Figure 4.6: Max Degree 3 Grid Graphs

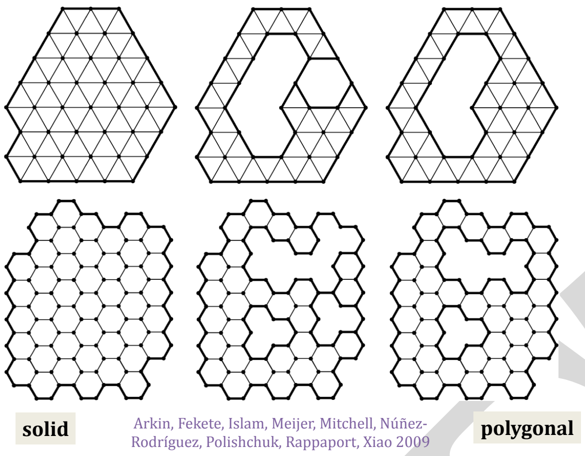

solid
Arkin, Fekete, Islam, Meijer, Mitchell, Núñez-Rodríguez, Polishchuk, Rappaport, Xiao 2009
polygonal

Figure 4.7: Polygonal Tiling

	triangular	square	hexagonal
solid	poly	poly	open
superthin	NP-hard	NP-hard	open
max-degree-4	NP-hard	-	-
max-degree-3	NP-hard	NP-hard	NP-hard
polygonal	poly	open	NP-hard

Figure 4.8: Taxonomy the Complexity of Hamiltonicity

4.7 Games are Hard Using Hamiltonicity

4.7.1 Settlers of Catan Longest Road Card

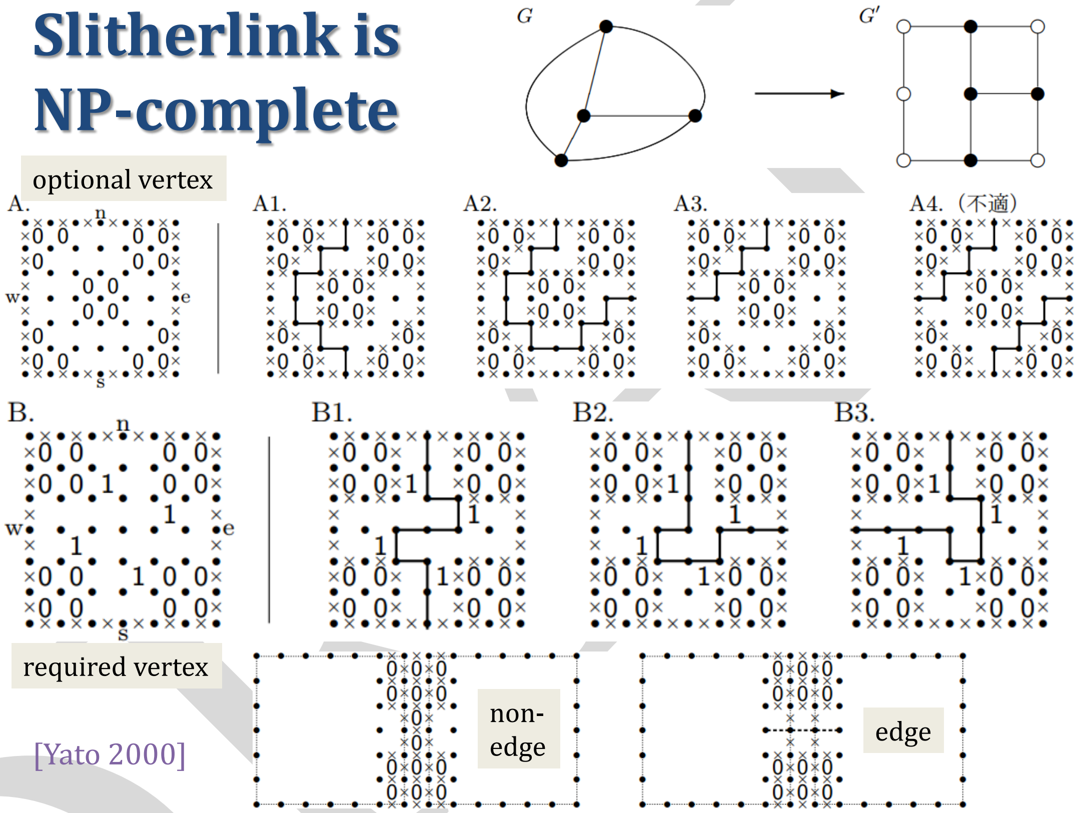
The Hamiltonian graph problem restricted to hexagonal grid graphs is NP-complete [AFI+09, DR17]. We use that to prove the following.[1]

Definition 4.12. Let G be a game and $i \in \mathbb{N}$. The ***Mate-in-i problem for G*** is the problem of, given a game position, can the player who is about to move get a win within i moves. Note that Mate-in-0 means the player about to move has already won; however, checking this might still be hard.

Theorem 4.13. *Mate-in-1 and mate-in-0 are NP-complete for Settlers of Catan.*

Proof sketch: In Settlers of Catan the player with ***The Longest Road card*** gets two victory points. If the opponent has a road of length $n - 1$ then mate-in-1/0 requires finding/having a Hamiltonian path in the hexagonal grid (with the opponents' roads forming obstacles). Thus we can do a reduction from Hamiltonian cycle in hexagonal grid graphs to either mate-in-1 or mate-in-0. ▮

Slitherlink is NP-complete



[Yato 2000]

Figure 4.9: Slitherlink is NP-Complete

Hashiwokakero is NP-Complete

[Andersson 2009]

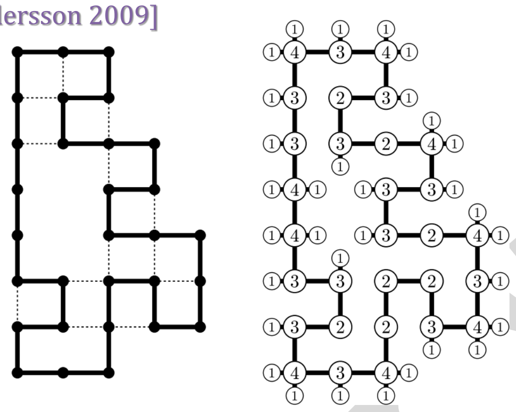


Figure 4.10: Hashiwokakero is NP-Complete

4.7.2 Slitherlink

Slitherlink is a Nikoli game played on a grid graph in which some pixels are labeled with the numbers 0 through 4. The objective of the game is to find a cycle (not necessarily Hamiltonian) along the grid lines such that the numbered pixels are bordered by that number of edges in the cycle.

> Slitherlink
> *Instance:* An Instance of the Slitherlink Game.
> *Question:* Can the player win?

Yato [Yat00] proved the following:

Theorem 4.14. *Slitherlink is NP-complete.*

Proof sketch: This is proven by reduction from Hamiltonian cycle on grid graphs. There are two vertex gadgets, one for vertices that are in the input graph (which must be in the cycle) and one for vertices in the grid graph that were not in the original graph (which are optional). ∎

4.7.3 Hashiwokakero

Hashiwokakero is a Nikoli game in which the goal is to add orthogonal noncrossing (multi-)edges to connect a given set of vertices with specified degree. Andersson [And09] proved the following:

Theorem 4.15. *Hashiwokakero is NP-complete.*

Proof sketch: This is proven by reduction from Hamiltonian cycle in grid graphs. Each vertex of the grid graph is mapped to a vertex with specified degree $2 + b$ where b is the number of directions in which it is not connected to another vertex. Those directions are filled with vertices of specified degree 1, which must fill up the b part of their neighboring vertex. The leftover 2 edges per vertex that must be placed are exactly the Hamiltonian cycle in the grid graph. ∎

[1]This result is from unpublished work by Kyle Burke, Erik Demaine, Gabe van Eycke, and Neil McKay (2011).

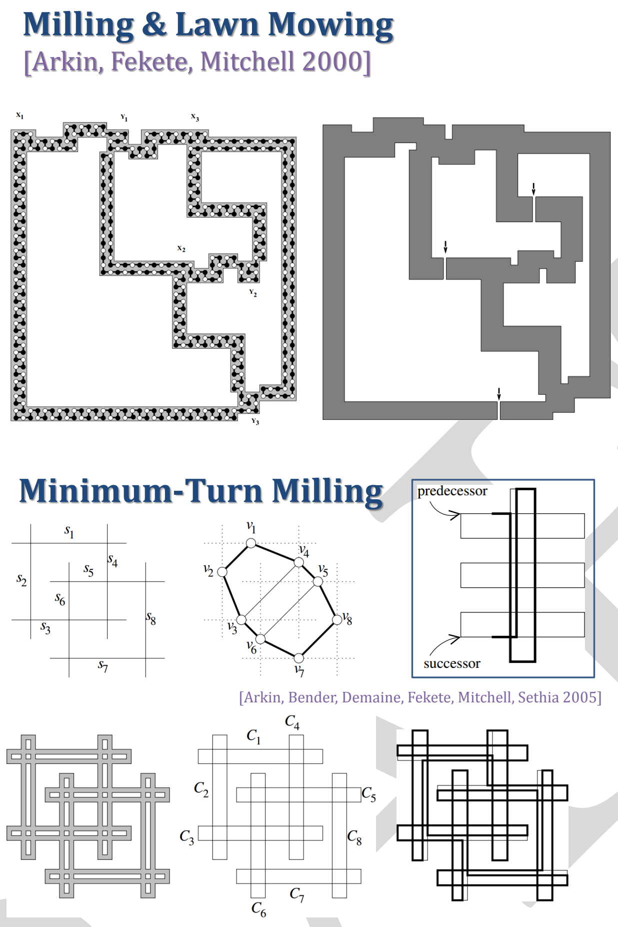

Milling & Lawn Mowing
[Arkin, Fekete, Mitchell 2000]
Minimum-Turn Milling
predecessor
successor
[Arkin, Bender, Demaine, Fekete, Mitchell, Sethia 2005]

Figure 4.11: Gadget For Showing Hashiwokakero is NP-Complete

4.7.4 Lawn Mowing and Milling

Lawn mowing and milling problems both involve cutting a specified region with a tool (we are not going to define them rigorously). In lawn mowing problems the tool path can go outside the region, while in milling it must stay inside. The goal is generally to find the shortest path. Milling problems arise in actual physical milling, while lawn mowing problems arise in laser and waterjet cutting and in each layer of 3D printing by deposition. Arkin et al. [AFM00] proved the following:

Theorem 4.16. *Milling and lawn mowing are NP-hard for grid polygons and a unit square too.*

Proof sketch: The proof is by reduction from Hamiltonicity in grid graphs. Minimum-turn milling (also infinite-acceleration milling) is NP-complete by reduction from Hamiltonicity in unit orthogonal segment intersection graphs; this problem arises in physical milling where the cost of milling a straight line is insignificant compared to the cost of stopping to turn. ∎

4.8 Metatheorems on Reductions Using Hamiltonian Cycle

In Viglietta's paper [Vig14] there are some metatheorems that will be used as general techniques for proving hardness. We will cover 2 main metatheorems (and some different versions of them): one for NP-hardness and one for PSPACE-hardness. They can be applied to a lot of games. We will use the term metatheorem in somewhat vague sense and not a formal theorem, because it's hard to state all the assumptions for all games. It will give a general set up for the proof.

Viglietta defines an "avatar" in his proofs, but for our case, we will use the term "player." The player is the character in the game that we can control and move around to complete objectives. One basic assumption we make about the player is that we can choose, at any time, to change the player's direction of movement.

4.8.1 Metatheorem 1

This metatheorem enunciates that if a game with a player traversing a 2D environment with a start location and:

- Location traversal (with or without a starting location or an exit location)
- Single-use paths (each path can only be traversed once)

then the game is NP-hard. Location traversal means that the player has to visit some locations in the board in order to win the level. The planarity of the problem will allow us not to worry about crossovers.

The claim of this metatheorem is that any game with such characteristic can be reduced from Planar Max-Degree 3 Hamiltonian Path. The reduction should turn each vertex in the graph into a location that must be traversed and each edge should become a single-use path. Since each vertex has degree 3 and each of the edges are single-use, once we traverse two edges to visit the vertex and leave, those edges will disappear, leaving just one edge. Now, the vertex is unreachable because if we use the third edge, the player will be trapped in that location, as there is no fourth edge out of the vertex.

Thus, we conclude that there will be a way to clear the game if and only if there is a Hamiltonian path.

We've seen reductions like this before, but with a time limit; this is another way to get the same kind a proofs.

Using this metatheorem, we can prove NP-hardness for many games.

Boulderdash

In this game, the player has a starting position and can walk around without being affected by gravity and dig in adjacent cells if they are made of earth. There are boulders are influenced by gravity and if they fall on the avatar, the player dies. The goal is to collect all the diamonds and get to the end location. You only need two gadgets: location traversal (shown in Figure 4.12 left) and single-use path (shown in Figure 4.12 center).

Figure 4.12 right demonstrates the single-use path gadget after it has been traversed from left to right: we push the first boulder into the pit in order to clear the obstacle; we then push the lower of the two stacked boulders over to the other pit, then push the last boulder into the final pit as part of our traversal. During this time, the higher of the two stack boulders falls down and blocks our path, since there is no pit to push it into.

(One small note is that the boulder can “rest” on the player without killing him, so pushing the lower boulder to the right and causing the player to stand under the higher boulder is permissible.)

The conclusion is that Boulderdash is NP-hard.

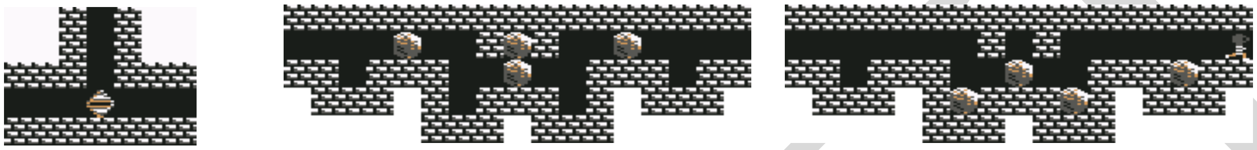


Figure 4.12: (Left) Location Traversal, (center) Single-use Path Gadgets for Boulder Dash. (Right) Single-use Gadget After Traversing From Left to Right.

Lode Runner

In this game the player have to collect all the coins and avoid enemies. You can dig a hole on the ground and the monsters can fall in this hole. Eventually, the hole will refill (after some specific time) releasing the monster. The avatar cannot jump and this property is exploited in the Single-use path gadget. Figure 4.13 shows the gadgets required for metatheorem 1. The conclusion is that Lode Runner is NP-hard.

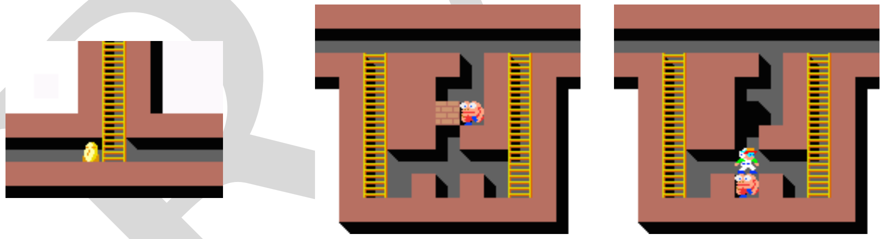


Figure 4.13: (Left) Location traversal gadget. (Center) Single-use path gadget and (right) it being used.

Zelda II

In this game you are a character called Link and this is a platform game. The location traversal is done by placing keys on certain locations. The keys are used to open doors. At the end of the level there will be exactly the same number of doors as the number of keys in the level, so in order to clear the level, we must collect all keys. The single-use paths are bridges that disappear when Link walks over them. An animation illustrating these two concepts can be found in the slides for the class in the course website.

The conclusion is that Zelda II is NP-hard.

4.8.2 Metatheorem 2

This is a slight variation of Metatheorem 1. It starts from the same kind of set up (player with a starting location), but requires tokens in order to traverse a path instead of single-use paths. The player collects tokens that appear in determined locations in the level, and uses the tokens to pay a toll in order to traverse some path. The idea is to simulate a single-use path with this kind of mechanism:

- We place a token at each vertex, and
- We transform every edge into a toll road that requires one token.

Thus, to get from one vertex to the next, the player must pay the token that was just acquired; if a vertex is visited more than once, there will be no token at that vertex, and the player is unable to cross over any edge.

The reduction from Hamiltonicity follows in the same way as in Metatheorem 1.

Pac-Man

We can use Metatheorem 2 to prove Pac-Man NP-hard. In order to win in Pac-Man, you have to collect all the dots, which will also function as our tokens. Figure 4.14 shows the degree-3 vertex with the token (dot) and the toll road (path with ghosts). Eating a token causes the ghosts to change state, so that Pac-Man can eat the ghosts; after some time, though, the ghosts revert to becoming harmful to Pac-Man. Also, whenever a ghost changes its state, it will change the direction of movement.

The ghosts that are "eaten" revive after a while and appear inside a "cage," which they then leave and continue moving around in their original paths.

Thus, the only way to traverse an edge is to consume the token and the ghost along the chosen exit edge. For the sake of argument, we will assume that the ghosts change state long enough for Pac-Man to exist safely, but short enough so that if Pac-Man even comes back to the same area of the map, the ghosts will have reverted states or respawned. Due to this, each edge can be traversed if and only if Pac-Man consumes the token; once the token is consumed, Pac-Man cannot return to this "vertex" again, since he won't be able to pay the toll anymore.

Therefore, Pac-Man is also NP-hard.

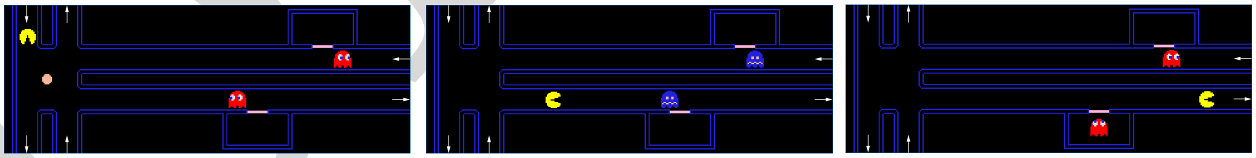


Figure 4.14: Token+Toll Road Gadget for Pac-Man.

Chapter 5

NP-Hardness via a Miscellany of Graph Problems

5.1 Introduction

In this chapter we prove several graph problems NP-complete and then use them to prove several non-graph problems NP-complete.

We remind readers what the degree of a graph is.

Definition 5.1. If G is a graph then ***the degree of G*** is the highest degree of a vertex.

Exercise 5.2. If $G = (V, E)$ is a graph and $v \in V$ then (1) $N(v)$ is the set of neighbors of v, and (2) the set of vertices that v ***dominates*** is $\{v\} \cup N(v)$.

Given a graph $G = (V, E)$, we say that G has a *k-strong coloring* if vertices of G can be colored by at most k colors such that no two vertices sharing the same edge have the same color and every vertex in the graph dominates an entire color class.

1. Show that, for all $k \geq 4$, determining whether a graph G has a k-strong coloring is NP-complete.

2. An *apex graph* is a graph that can be made planar by the removal of a single vertex. Show that the problem in the first part remains NP-complete if the input is restricted to *apex* graphs.

5.2 Vertex Cover

We look at variants of Vertex Cover, some in P and some NP-complete.

The following problems are in P:

1. **Exact vertex cover**, where each edge must be incident to exactly one vertex.

2. **Edge cover**, where we choose k edges to cover all vertices in a graph.

Planar Connected Vertex Cover

[Garey & Johnson 1977]

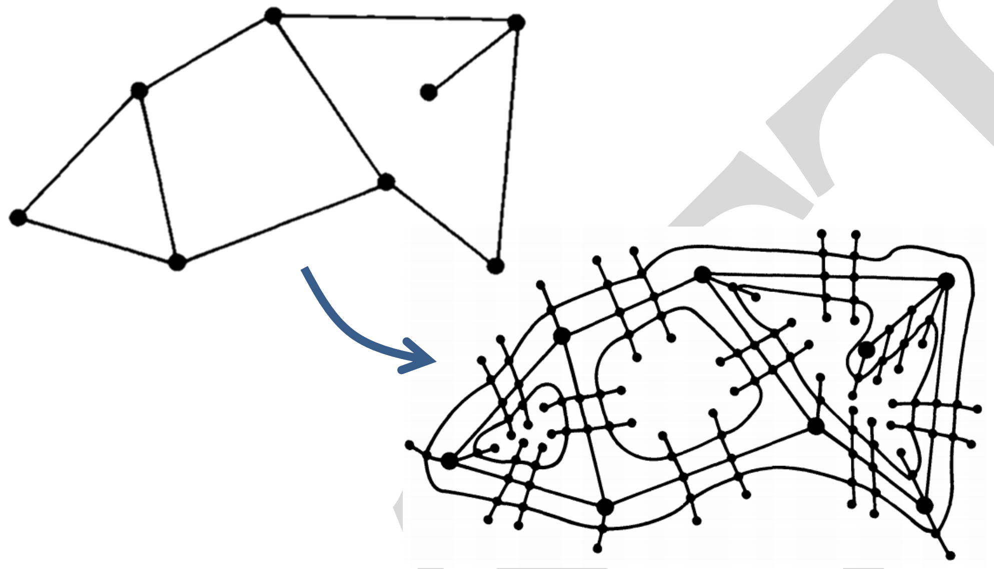


Figure 5.1: Reduction of Vertex Cover to Induced Subgraph Vertex Cover

By Theorem 2.4.2 Vertex Cover is NP-complete even when restricted to planar graphs of degree 3.

> Induced Subgraph Vertex Cover
> *Instance:* A graph G and a number k.
> *Question:* Is there a vertex cover of G of size k such that the vertices in the vertex cover induce a connected subgraph.

Garey & Johnson [GJ77] proved the following.

Theorem 5.3. *The induced Subgraph Vertex Cover problem is NP-complete, even when restricted to planar graphs of degree 4.*

Proof sketch:

We reduce Planar-VC restricted to graphs with degree 3 to Planar Induced Subgraph VC restricted to graphs of degree 4.

In Figure 5.1 we transform our given planar graph G by adding a closed loop for each face in the graph. For each edge in G that separates two loops l_1 and l_2, we add several vertices that "connect" the two closed loops together. The construction adds exactly $5 \cdot |E|$ edges to the graph, and increases each of the original vertex's degrees up at most 4.

Note that there always exists an optimal vertex cover where we never choose any leaves in the graph. This is because choosing the node adjacent to a leaf in our cover is always at least as good as choosing the leaf itself. Thus, to obtain a connected vertex cover, we must choose exactly one of the two nodes in each subdivided edge to connect each of the closed loops together. (It is never more useful to choose both, since we could simply choose a vertex of our original graph G and be guaranteed to cover at least as many nodes.) After we have done so, these additions induce a connected graph.

▮

5.3 Shortest Common Subsequence

Shortest Common Subsequence (SCS)
Instance: An alphabet Σ, $S \subseteq \Sigma^*$, and $k \in \mathbb{N}$.
Question: Is there a string y, $|y| \le k$, such that, for all $x \in S$, x is a subsequence of y.

Maier [Mai78] proved the following.

Theorem 5.4. *SCS is NP-complete.*

Proof. We show that Vertex Cover $\le_p$ SCS.

1. Input (G, V) and $k \in \mathbb{N}$. We will let $\Sigma = V \cup E \cup \{*\}$.

   We assume $V = \{1, \ldots, n\}$. We assume there are m edges. Let $c = \max\{m, n\}$.

   Let $\widehat{V}$ be the string of vertices in order, so just

   $$123 \cdots n.$$

   Let $\widehat{E}$ be the string of edges in lex order, but each edge appears twice in a row. For example, if the only edges were $(1, 2)$ and $(2, 3)$ this would be

   $$(1, 2)(1, 2)(2, 3)(2, 3).$$

   Let $\widehat{A}$ be the string of $4c$ $*$'s. If $c = 2$ this would be

   $$* * * * * * * *$$

2. We create an instance of SCS.

   (a) $\Sigma = \{1, \ldots, n\}$.

   (b) The set S has the strings

   $$\widehat{A}\widehat{E}\widehat{E}\widehat{V}.$$

   and, for every $(i, j) \in E$, the string

   $$(i, j)(i, j)i\widehat{A}j(i, j)(i, j).$$

We now need to prove that G has a Vertex Cover of size $\leq k$ if and only if there is a supersequence of length $\leq 8c + 6m + 2n + k$. This is rather difficult so we direct the reader to either try to prove it themselves or go to the original paper. □

Exercise 5.5. Maier [Mai78] also prove that SCS restricted to $|\Sigma| = 5$ is NP-complete. Räihä & Ukkonen [RU81] showed that SCS restricted to $|\Sigma| = 2$ is NP-complete. Prove or look up these results.

The Shortest Common Subsequence Problems is interesting in its own right. The Restricted Shortest Common Subsequence problem is only interesting since it will enable us to prove the hardness of variants of the Flood-it game (we will do that in the next section).

Restricted Shortest Common Subsequence (RSCS)
Instance: An alphabet Σ, $S \subseteq \Sigma^*$, and $k \in \mathbb{N}$. No string in S has two of the same characters in a row.
Question: Is there a string y, $|y| \leq k$, such that, for all $x \in S$, x is a subsequence of y.

Exercise 5.6. Show that RSCS is NP-complete.

5.4 The Flood-It Game

We will study a variant of Flood-it, played on a tree, in Chapter 8. Our context will be parameterized complexity.

We describe the 1-player game Flood-it on a square grid.

Definition 5.7.

1. A set of squares is ***connected*** if you can get from any square in the set to any other square in the set by going through walls. Note that two squares that meet at a corner are not connected.

2. Assume an $n \times n$ grid is c-colored with no restriction. A ***mono-region*** is a connected set of squares that are the same color.

Definition 5.8. The game ***Flood-It (on a grid)*** is defined as follows.

1. The parameters are $n, c, g \in \mathbb{N}$.

2. The starting position is an $n \times n$ grid that is c-colored. There are no restrictions on the coloring. The goal is to, through a sequence of $\leq g$ moves (to be defined soon), make the entire grid monochromatic. (See Figure 5.2 for an example of what a position looks like.)

3. A move consists of a player picking one of the c colors.

4. The move causes the following to happen. Take the mono-region that contains the top left square. Change all of the squares in it to color c. Note that if there are squares colored c next to the squares that changed to c then the mono-region containing the top left square is now larger.

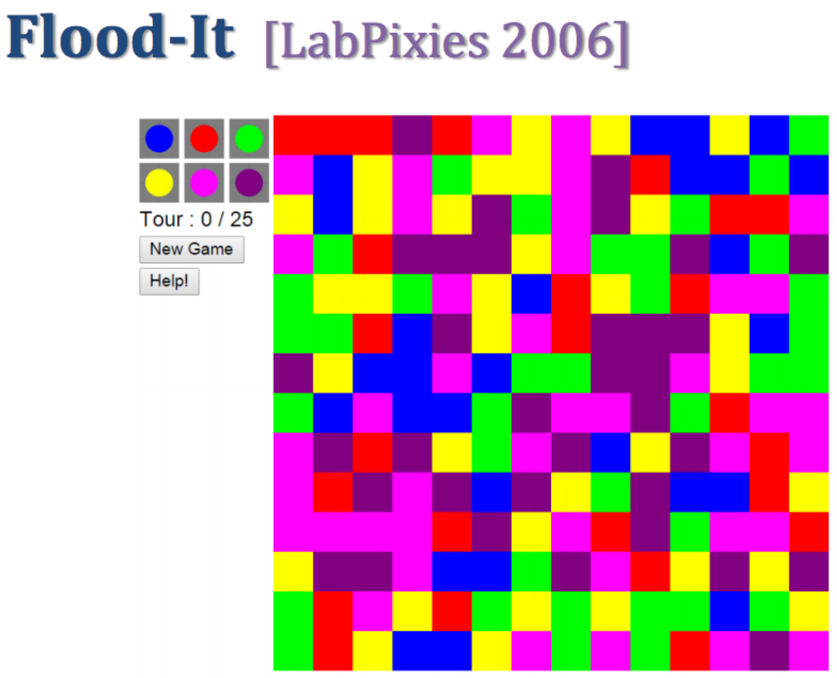


Figure 5.2: Example of a Position in Flood-It

5. The game ends when all of the squares are the same color. If this is accomplished within g moves then the player wins. Else he loses.

There are several sites where one can play this game. Here is one:

https://unixpapa.com/floodit/

> Flood-it on Grids (FLOOD-IT ON GRAPHS)
> *Instance:* Parameters $n, c, g \in \mathbb{N}$ and an $n \times n$ grid that is c-colored.
> *Question:* Can the player win the game?
> (The G in FLOOD-IT ON GRAPHS is for Grid. In a later chapter we will define the Flood-it game on trees. That will be denoted FLOOD-IT ON TREES.)

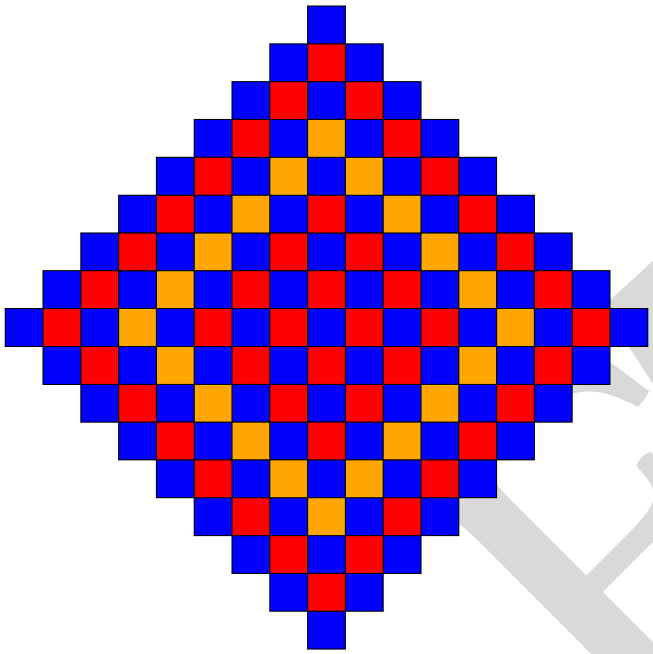
Clifford et al. [CJMS12] proved the following.

Theorem 5.9. *FLOOD-IT ON GRAPHS is NP-complete.*

Proof sketch: We show that RSCS $\le_p$ FLOOD-IT ON GRAPHS.
We first define the gadget needed.

Definition 5.10. Let Σ be a finite alphabet. Let $w = w_1 \cdots w_n$ where $w_i \in \Sigma$. Then DIA(w) is a diamond-shaped subset of a grid such that the border is colored w_1, the next level is colored w_2, etc. See Figure 5.3.

We give a reduction of a slightly different type. We will take a set of strings S and produce an instance of the game G (but without g) such that the length of the shortest common subsequence is exactly the number of moves needed to make the grid monochromatic.
Here is the reduction.



Color 1 = 🟧 Color 2 = 🟥 Color 3 = 🟦

Figure 5.3: DIA(323132323)

1. Input is alphabet Σ and a set $S \subseteq \Sigma^*$. We will view Σ as a set of colors. Let d be a color that is not in Σ.

2. For each $w \in S$ form $\text{DIA}(w)$.

3. Create a game of Flood-it on Graphs which is formed by first creating an $n \times n$ grid (we will determine n later) and the placing all $\text{DIA}(w)$ into the grid such that they do not overlap. Take n large enough so that this can be accomplished.

We leave it to the reader to show that the reduction works. ▮

Exercise 5.11. Show that the reduction in Theorem 5.9 works.

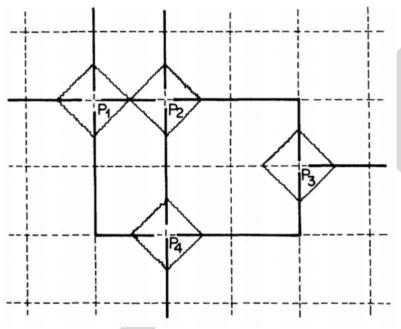
Clifford et al. [CJMS12] have shown many more versions of Flood-it on Graphs NP-complete. Fellows et al. [FRdSS18] have upper and lower bounds on the complexity of versions of Flood-it on Trees (we will look at this in Chapter 8). For a collection of papers on Flood-it see the website
`http://www.cs.umd.edu/gasarch/TOPICS/floodit/floodit.html`

5.5 The Steiner Tree Problem and Its Variants

Steiner Tree
Instance: A graph weighted graph $G = (V, E)$ with non-negative edge weights and $T \subseteq V$. The set T is called the ***terminals***.
Question: Is there a tree that contains T (it may also contain other vertices) of cost $\le k$.

Rectilinear Steiner Tree

[Garey & Johnson 1977]



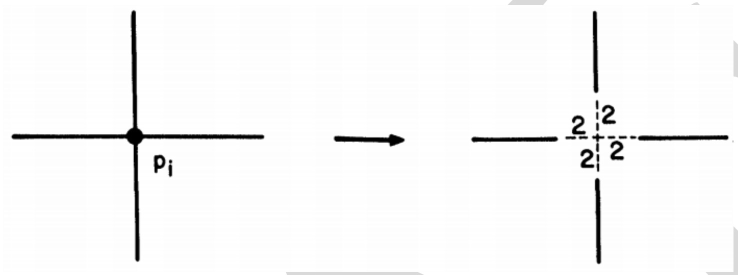


Figure 5.4: Transformation for Rectilinear Euclidean Steiner Tree

Euclidean Steiner Tree
Instance: n points in the plane with integer coordinates, and a number k.
Question: Can you build roads which have total length $\leq k$ that connect all the points. Note that we are allowed to add arbitrary vertices to shorten the length of road. For example, if we are given 4 points at $(-1,-1)$, $(-1,1)$, $(1,-1)$, $(1,1)$, then we can add a vertex at $(0,0)$ and connect each of our four given vertices to the new vertex to form the minimum Steiner tree.

Rectlinear Euclidean Steiner Tree
Instance: Given n points in the plane and a number k.
Question: Can you build roads that are all parallel to the x or y-axis which have total length $\leq k$ that connect all the points.
Note: This problem is interesting since it applies to printed circuit boards.

Garey & Johnson [GJ77] proved the following.

Theorem 5.12. *Rectlinear Euclidean Steiner Tree is NP-complete.*

Proof sketch:

We reduce the induced subgraph vertex cover problem for planar graphs of degree 4 to the rectilinear Euclidean Steiner tree problem.

1. Input $G = (V, E)$ and k where G is planar and of degree ≤ 4.

2. We draw our given graph rectilinearly on the grid, and scale it by $4n^2$. We add auxiliary points at all integer points along the edges, except within radius 1 of the vertices. (The vertex transformation is shown in the Figure 5.4.) It is fairly easy to see that each of the points comprising the "edges" of the graph must be connected to its adjacent points. Also, every edge must connect to a vertex, so we must add at least $2|E|$ edges. We must also connect the other end of the edges in a spanning tree of G, which means we must add at least $2(|V| - 1)$ edges.

▮

5.6 Two More Push Games

In Section 1.7 we proved many push games are NP-hard by using reductions to variants of SAT. In this section we state a two more results on push games. They are proved NP-hard using reductions to the same graph problem. We will not prove these statements; however, we will say what graph problem is used.

Push-1X and Push-1G
Instance: The initial configuration of a push problem.
Question: Push-1X: Can the object get to a specified location by pushing blocks in a grid, subject to the stipulation that the object never visit the same location twice?
Question: Push-1G: Can the object get to a specified location by pushing blocks in a grid, subject to the stipulation that the robot can only push one block, and blocks that are directly over one or more open squares fall immediately downward as far as possible (the G stands for "Gravity")?

Theorem 5.13.

1. *(Demaine et al. [DDHO01]) Push-1X is NP-complete.*

2. *(Friedman [Fri02b]) Push-1G is NP-complete.*

Proof sketch: Both proofs are reductions from the push problem to Planar Degree-4 3COL.
▮

5.7 Graph Orientation

Horiyama et al. [HIN+12] defined the following problem and showed it was NP-complete. This problem is not interesting in itself, but is a means to an end. We will be using it in the next section to show a packing problem is NP-complete.

Graph Orientation
[Horiyama, Ito, Nakatsuka, Suzuki, Uehara 2012]

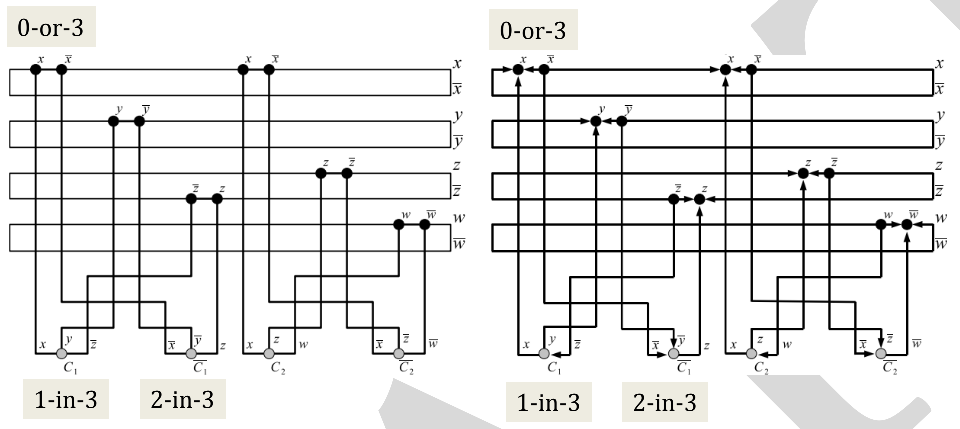


Figure 5.5: Reduction of 1-IN-3SAT to Graph Orientation

GRAPH ORIENTATION
Instance: An undirected graph $G = (V, E)$ and a partition of V into sets V_ℓ, V_c, V_n (this will later be Literal, Clause, Negated Clause).
Question: Is an orientation of the edges such that the following hold?

- Each $v \in V_\ell$ has indegree 0 or 3.
- Each $v \in V_c$ has indegree 1.
- Each $v \in V_n$ has outdegree 1.

Theorem 5.14. *GRAPH ORIENTATION is NP-complete even when restricted to graphs of degree 3.*

Proof sketch: We show GRAPH ORIENTATION is NP-complete by reducing from 1-IN-3SAT. We just give an example from which the general construction will be clear. Let

$$\varphi = (x \vee y \vee \overline{z}) \wedge (x \vee z \vee w).$$

Then the undirected graph in Figure 5.5 on the right is the instance of GRAPH ORIENTATION that is created. The directed graph on the left represents a truth assignment that satisfies exactly 1 literal per clause. ∎

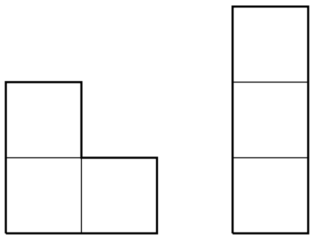


Figure 5.6: The 2 Trominoes

5.7.1 Packing Trominoes into a Polygon

Definition 5.15.

1. An ***L*-tromino** is the shape in Figure 5.6 on the left.

2. An ***I*-tromino** is the shape in Figure 5.6 on the right.

Packing L-Trominoes, Packing I-Trominoes
Instance: A set of L-trominoes (I-trominoes) and a polygon (which may have holes in it). The input is really a number and we think of having that number of L-trominoes (I-trominoes).
Question: Can the L-trominoes (I-trominoes) tile the polygon, filling every space and with no overlap.

Theorem 5.16.

1. *Packing L-Trominoes is NP-complete.*

2. *Packing I-Trominoes is NP-complete.*

Proof sketch:
1) We reduce graph orientation to packing L-trominoes. We construct the necessary gadgets (edge and crossover gadgets, 0-or-3 gadget, 1-in-3 and 2-in-3 gadgets) as below. The 0-or-3 gadget comes in the form of a double 0-or-3 gadget, which is the only one required for the reduction from graph orientation (since they are only used in pairs to set the variables). The orientation of the trominoes represents the directedness of the edge. For example, it is easy to check that exactly one of the grid squares outside the large square will be filled. An exact packing will only be possible if we can satisfy the corresponding graph orientation problem.
2) We follow the same techniques as for packing L-trominoes, and construct the necessary gadgets as in Figure 5.8. The bend in the edge gadget is to ensure that only two parities exist (if we just used I shapes in a line, we could have a missing block on one end and a protrusion of 2 blocks on the other end). Other than that, the construction is similar to that of the L-tromino case.
∎

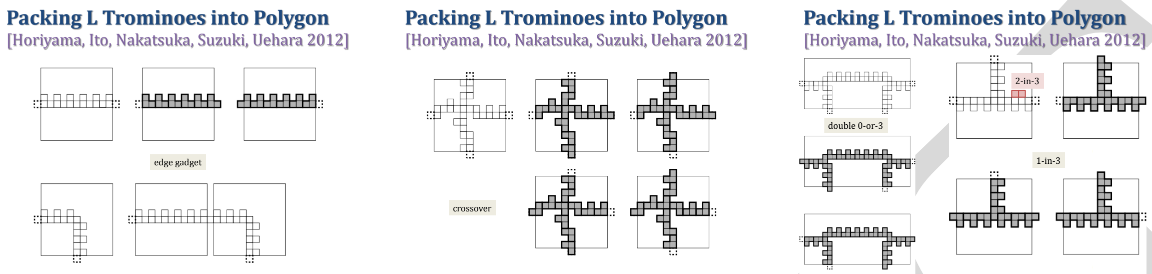

Packing L Trominoes into Polygon
[Horiyama, Ito, Nakatsuka, Suzuki, Uehara 2012]
edge gadget
Packing L Trominoes into Polygon
[Horiyama, Ito, Nakatsuka, Suzuki, Uehara 2012]
crossover
Packing L Trominoes into Polygon
[Horiyama, Ito, Nakatsuka, Suzuki, Uehara 2012]
double 0-or-3
2-in-3
1-in-3

Figure 5.7: Gadgets for Reduction of Graph Orientation to L-trominoes Packing

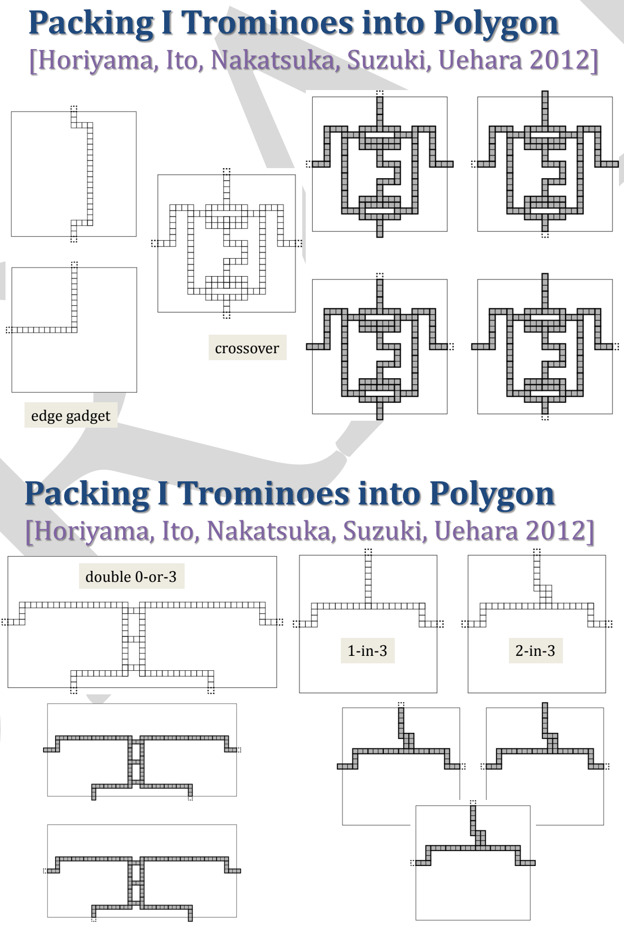

Packing I Trominoes into Polygon
[Horiyama, Ito, Nakatsuka, Suzuki, Uehara 2012]
crossover
edge gadget
Packing I Trominoes into Polygon
[Horiyama, Ito, Nakatsuka, Suzuki, Uehara 2012]
double 0-or-3
1-in-3
2-in-3

Figure 5.8: Gadget For Reduction of Graph Orientation to Packing I-Trominoes

5.8 Linear Layout

In this section we define a large set of layout problems and state results. There are many problems that can be described as linear layout problems. We will use these results in Section 5.8.1.

Definition 5.17. Let $G = (V, E)$ be a graph. A ***linear layout of G*** is a bijection from V to $\{1, \ldots, |V|\}$. We impose many conditions on these functions:

Linear Layout Problems

Instance: A graph $G = (V, E)$.

Question: Find a linear layout of G that optimizes some metric. We describe several such problems.

Bandwidth.- Let the points be laid out on a number line so that the distance between consecutive points is a unit length. Minimize the maximum length of any edge between two points. This problem is motivated by the concept of bandwidth in linear algebra. In linear algebra, matrices whose nonzero values are all in a thin band around the main diagonal are much easier to manipulate and, if they represent a system of linear equations, easier to solve. Permuting the columns and rows of an adjacency matrix is analogous to permuting the points on a linear layout. Bandwidth problems are hard even for very constrained graphs, such as trees of maximum degree 3 and caterpillars.

Minimum Linear Arrangement (MinLA). The goal in this problem is to minimize the *sum* of the edge lengths (as opposed to the maximum edge length, as in bandwidth). This problem is motivated by VLSI chip design.

Cut Width. Let the ***size*** of a cut in a linear layout be the number of edges with one endpoint at i or less and the other endpoint at $i + 1$ or more, for some i. Minimize the maximum size of any cut. Note that attempting to minimize the sum of the cuts is identical to Minimum Linear Arrangement.

Modified Cut (abbreviated Mod Cut).

Vertex Sep. This problem is similar to Cut Width, except that we consider the ***vertex separation*** of a cut, which is the number of vertices that have at least one edge crossing the cut. We aim to minimize the maximum vertex separation of any cut.

Sum Cut. This problem uses the same setup as vertex separation. The only difference is that we aim to minimize the sum of the cuts instead.

Edge Bis. Minimize the number of edges that cross the middle cut.

Betweeness. This problem is not posed in a graph-based way. Instead, you are given a list of rules of the form "y is between x and z", meaning that either $x < y < z$ or $z < y < x$.

The following table is from Diaz et al. [DPS02] survey of layout problems, except the entry on Betweeness.

Exercise 5.18. The problems presented in Definition 5.17 are all functions. For each metric (e.g., Vertex Bisection) do the following.

Problem	Type of Graph	Paper
BANDWIDTH	General	[Pap76]
BANDWIDTH	Trees with $\Delta \leq 3$	[GGJK78]
BANDWIDTH	Caterpillars with hair length ≤ 3	[Mon86]
BANDWIDTH	Caterpillars with ≤ 1 hair per backbone vertex	[Mon86]
BANDWIDTH	Cyclic caterpillars with hair length 1	[DPPS01]
MINLA	General	[GJS76]
MINLA	Bipartite Graphs	[ES75]
CUT WIDTH	General	[Gav77]
CUT WIDTH	Max Degree 3	[MPS85]
CUT WIDTH	Planar and $\Delta \leq 3$	[MS88]
CUT WIDTH	Grid Graphs & Unit Disk Graphs	[DPPS01]
MOD CUT	Planar Graphs with $\Delta \leq 3$	[MS88]
VERTEX SEP	General	[Len81]
VERTEX SEP	Planar Graphs with $\Delta \leq 3$	[MS88]
VERTEX SEP	Chordal Graphs	[Gus93]
VERTEX SEP	Bipartite Graphs	[GMKS95]
VERTEX SEP	Grid and Unit Disk Graphs	[DPPS01]
SUM CUT	General	[DGPT91]
SUM CUT	co-bipartite	[YJ94],[Gol97],[YLLW98]
EDGE BIS	General	[GJS76]
EDGE BIS	$\Delta \leq 3$	[McG78]
EDGE BIS	Δ bounded	[McG78]
EDGE BIS	d-regular graphs	[BCLS87]
BETWEENESS	Not a Graph Problem	[Opa79]

Figure 5.9: NP-Complete Layout Problems

1. Come up with a set version of the function problem.

2. Show that if the set version is in P then the function version is in FP. For example, if the set version of Vertex Bijection is in P then the problem of *finding* the linear arrangement that minimizes the number of vertices in the left half that have edges to the right half is in FP.

5.8.1 Bipartite and General Crossing Number

Recall that a graph is planar if it can be drawn in the plane with 0 pairs of edges crossing. That way of stating it leads to the following definition.

Crossing Numb and Bip Crossing Numb
Instance: A graph G.
Question: Crossing Numb. What is the least c such that the G can be drawn in the plane with c pairs of edges crossing? The number c is called the ***crossing number of G***.
Question: Bip Crossing Numb is this problem restricted to bipartite graphs, and requiring that the left and right vertices are still on the left and on the right.

Note that both Crossing Numb and Bip Crossing Numb are functions not sets. This section will do reductions between functions; however, it would be a routine matter to translate all of the concepts and proofs into sets.

Garey & Johnson [GJ83] showed that determining the crossing number of a graph is NP-hard. We will follow their treatment. They showed the following.

Theorem 5.19.

1. *MinLA $\le_p$ Bip Crossing Numb.*

2. *Bip Crossing Numb $\le_p$ Crossing Numb.*

Proof sketch:
1) Minimum Linear Arrangement $\le_p$ Bip Crossing Numb.

1. $G = (V, E)$ that we want to find the Minimum Linear Arrangement of.

2. Create a graph as follows.

   (a) For each vertex v in G there is a vertex v'. Let V' be the set of all the v'-vertices. There will be no edges between elements of V. There will be no edges between elements of V'. The only edges will be between V and V' and hence we are creating a bipartite graph.

   (b) For all v in G there is an edge between v and v'.

   (c) For all edges (u, v) in G there is an edge (u, v').

   (d) For every edge between V and V' we add a large bundle of more edges. $()|V| + |E|)^2$ edges will do. We do this so that the ordering of the vertices of V and of their analogs in V' will be the same.

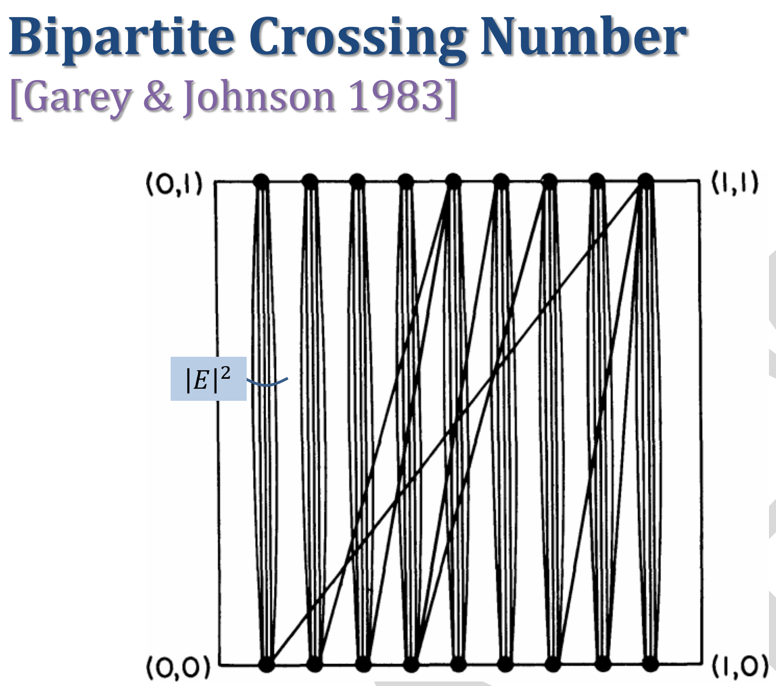


Figure 5.10: Bipartite Graph From Reduction of Minimum Linear Arrangement

See Figure 5.10 for what the final bipartite graph looks like.

Exercise 5.20. 1. Show that in an optimal (in terms of crossing number) drawing of the graph with V on one rail and V' on a parallel rail, the permutations of V and V' are the same.

2. Show that the an optimal (in terms of crossing number) drawing of the graph gives a permutation which minimizes the sum of the edge lengths as a linear arrangement of G (and hence solves Minimum Linear Arrangement).

2) Clearly, a bipartite graph is an example of a general graph, so all we have to do is impose some additional structure on the bipartite graph to mimic the 'two rails' condition from the bipartite crossing number problem. To do so, add two 'bounding' vertices X and Y (the top and bottom vertices in the following diagram). Connect X to the first part of the bipartite graph using large bundles, and connect Y to the other part. Then connect X and Y to each other using large bundles twice. Large bundles should be significantly larger than B (B^4 is probably safe, but smaller numbers may work as well). Then, the drawing of the bipartite graph is forced as shown in Figure 5.11.

∎

5.9 Rubik's Cubes

Demaine et al. [DDE+11] studied the complexity of the $n \times n$ Rubik's cube puzzle. We omit formal definitions.

Crossing Number is NP-Complete
[Garey & Johnson 1983]

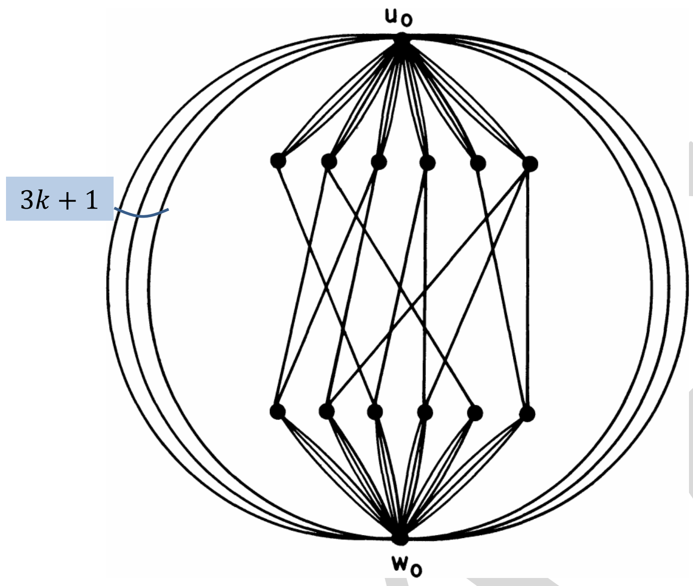


Figure 5.11: Reduction of Bip Crossing Numb to Crossing Numb

It is well known that both the $n \times n \times$ and $n \times n \times 1$ Rubik's cute can be solved with $O(n^3)$ moves. They showed the following.

Theorem 5.21.

1. *For both the $n \times n \times n$ Rubik's cube and the $n \times n \times 1$ Rubik's Square, from any position, there is a solution with $O(\frac{n^3}{\log n})$ movements.*

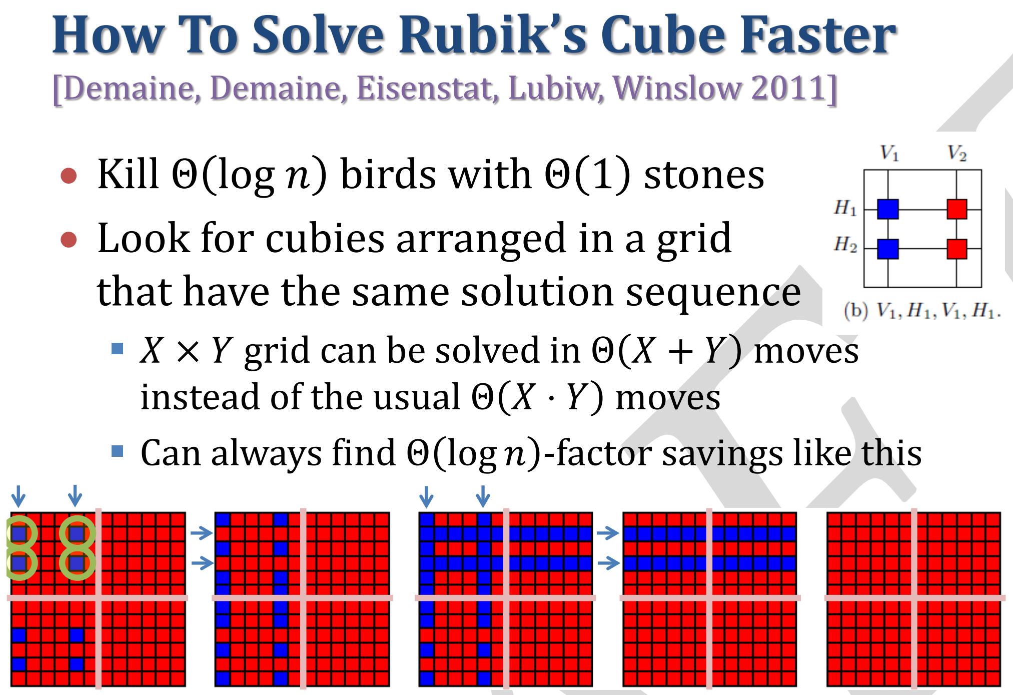
2. *For both problems mentioned in part 1 there are cases that require $\Omega(\frac{n^3}{\log n})$ moves.*

Proof sketch: The explanation applies to both the Rubik's cube and the Rubik's square.

On a high level, this is done by identifying $\Theta(\log n)$ ***cubies*** ($1 \times 1 \times 1$ cubes on the boundary) that can be solved with the same solution sequence. In the above figure, the four circled cubies can be solved simultaneously by a vertical flip on those two columns, followed by a horizontal flip on those two rows, then another vertical and horizontal flip. They showed that they can always find such a set of cubies to solve as a batch, thus providing a $\Theta(\log n)$ factor of savings. See Figure 5.12 for a sketch.

The lower bound is a counting argument. ▮

Drucker and Erickson in 2010, on the StackExchange forum at http://cstheory.stackexchange.com/questions/783, asked if the problem of finding the optimal solution for a given position is NP-complete. Demaine et al. did not quite prove that, but they did prove the following.


How To Solve Rubik's Cube Faster

[Demaine, Demaine, Eisenstat, Lubiw, Winslow 2011]

- Kill $\Theta(\log n)$ birds with $\Theta(1)$ stones
- Look for cubies arranged in a grid that have the same solution sequence
  - $X \times Y$ grid can be solved in $\Theta(X + Y)$ moves instead of the usual $\Theta(X \cdot Y)$ moves
  - Can always find $\Theta(\log n)$-factor savings like this

Figure 5.12: Rubik's Cube

Theorem 5.22. *The following problem is NP-complete: Given a position of the $n \times n \times 1$ Rubik's square, with some cubes whose positions you don't care about, and a number k, can you solve the problem (and note that this will not be a full solution) in $\leq k$ moves?*

Proof sketch:

The reduction is from betweenness, a problem we encountered in Definition 5.17. That problem is not quite enough, so the reduction also uses 1-IN-3SAT.

▮

5.10 Further Results

We present a list of results which can be viewed either as further reading (we provide references) or exercises.

Definition 5.23. Let $G = (V, E)$ be a graph. A ***biclique*** is a set of two disjoint sets $A, B \subseteq V$ such that, for all $a \in A$ and $b \in B$, $(a, b) \in E$. An ***induced biclique*** is a biclique where there are no edges between vertices of A or vertices of B.

MAX EDGE BICLIQUE and Variants
Instance: A bipartite graph $G = (A, B, E)$ and number $k \in \mathbb{N}$.
Question: Is there a biclique with $|A| = |B| = k$.
Question: Is there a biclique with $|A| + |B| \geq k$.
Question: Is there a biclique with $|A| \times |B| \geq k$.

Theorem 5.24.

1. *(This is stated by Garey & Johnson [GJ79], where it is called* ***Balanced Complete Bipartite Subgraph****, and proven by Johnson [Joh87].) The question of finding a biclique with $|A| = |B| = k$ is NP-complete.*

2. *(This is folklore.) The question of finding a biclique with $|A| = |B| \leq k$ is P.*
   Hint: *Use matching.*

3. *(Peeters [Pee03]) The question of finding a biclique with $|A| \times |B| \geq k$ is NP-complete.*

> EDGE DOM SET
> *Instance:* A graph $G = (V, E)$ and number $k \in \mathbb{N}$.
> *Question:* Is there a set $E' \subseteq E$, $|E'| = k$, such that every edge $e \notin E'$ is adjacent to an edge in E'.

Theorem 5.25.

1. *EDGE DOM SET is NP-complete.*

2. *(Yannakakis & Gavril [YG80]) EDGE DOM SET is NP-complete even when restricted to bipartite graphs of degree 3. (Yannakakis & Gavril [YG80]) EDGE DOM SET is NP-complete even when restricted to planar graphs of degree 3.*

Chapter 6

NP-Hardness via 3-Partition

6.1 Introduction

In this chapter, we first examine several classes of NP-hardness and polynomial-time algorithms which arise from differences in how integers are encoded in problem input.

We then look at the 3-partition problem, which is very useful for proving the strongest notion of NP-hardness. Finally, we use reduction from 3-partition to prove NP-hardness for several problems. Four of them are about packing-puzzles. These four are equivalent to each other.

6.2 Types of NP-Hardness

Consider a number problem, that is, a problem whose input includes one or more integers. The complexity of the problem may depend on how the integers are represented. We restate Definition 0.19 of weakly NP-hard and strongly NP-hard.

Definition 6.1. Let A be a problem that has numbers in the input.

1. A is **weakly NP-hard** if the problem is NP-hard when the numbers are given in binary. A might be in P if the numbers are given in unary.

2. A is **strongly NP-hard** if the problem is NP-hard when the numbers are given in unary. (Such problems are also weakly NP-hard but we do not call them that.)

We also need to differentiate algorithms for the two cases.

Definition 6.2. Let A be a problem with numbers. Assume $(a_1, \ldots, a_n)$ is most or all of the input. Assume a_1 is the max element.

1. An algorithm for A is **pseudo polynomial** if it is polynomial in n and a_1 (so the input can be viewed as being in unary). This often arises in dynamic programming, where the table grows proportionally with the integer values. Examples: knapsack, partition.

2. An algorithm for A is **weakly polynomial** if it is polynomial in n and $\log(a_1)$ (so the input can be viewed as being in binary). This is the typical meaning of polynomial time.

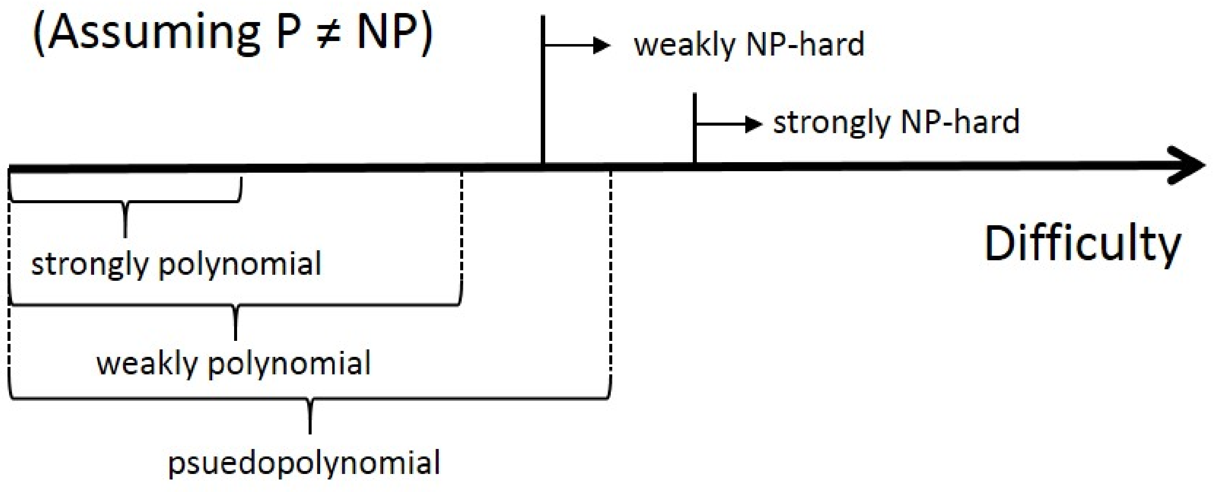


Figure 6.1: Hierarchy

3. An algorithm for A is **strongly polynomial** if it is polynomial in n alone.

What we care about most is the distinction between pseudo polynomial and weakly polynomial. Assuming P ≠ NP:

- If a problem is weakly NP-hard, there is no weakly polynomial algorithm (but there might be a pseudo polynomial algorithm).
- If a problem is strongly NP-hard, there is no pseudo polynomial algorithm (and therefore no weakly polynomial algorithm).

Figure 6.1 summarizes the relationships between these hardness types and algorithm strengths:

In practice, the integers involved in a problem are often only polynomially large in n, so a pseudo polynomial algorithm may be efficient. Thus there is practical significance in showing strong vs. weak NP-hardness, since only the former rules out pseudo polynomial algorithms.

6.3 Partition Problems and Scheduling

Most of the proofs in this chapter will be reductions from 3-Partition. But before that, we'll introduce Partition (This is sometimes also called 2-Partition, though we will *never* call it that since we will be defining 3-Partition which is very different. The '2' and '3' have nothing to do with each other)

Notation 6.3. If A is a multiset of numbers then $\sum_{i=1}^{n} a_i$ is the sum of all the numbers in A, counting multiplicities. For example, if $A = \{1, 1, 2\}$ then $\sum_{i=1}^{n} a_i = 4$.

6.3.1 Partition

Partition
Instance: Multiset of integers $A = \{a_1, a_2, \ldots, a_n\}$.
Question: Can A be partitioned into 2 sets A_1 and A_2 that have the same sum? Formally: $\sum A_1 = \sum A_2$. (The a_i's are in binary.)

Subset Sum is a generalization of this problem. It is also weakly NP-hard:

Subset Sum
Instance: n integers, $A = \{a_1, \ldots, a_n\}$ and a target sum t.
Question: Is there a $S \subset A$ such that $\sum S = t$? (The a_i's are in binary.)

Garey & Johnson [GJ79] (page 60) showed the following:

Theorem 6.4. *Partition is NP-complete, hence Subset Sum is NP-complete. Since the numbers are in binary, they are both weakly NP-complete.*

Is Partition strongly NP-complete? Unlikely, as Exercise 6.5 shows that Partition with numbers in unary is in P.

Exercise 6.5.

1. Show that there is an algorithm for Partition that is polynomial in $n, \max\{a_i\}$.
   Hint: Use dynamic programming.

2. Show that Partition is NP-complete.

Alfonsín [Alf98] looked at variants of Subset Sum. We look at one of them. The basic idea is that instead of being able to use a_i just once, you can use it $\le r_i$ times where r_i is part of the input.

Subset Sum With Repetition
Instance: Positive integers $a_1, \ldots, a_n$ and $r_1, \ldots, r_n$ and a target t.
Question: Does there exist $0 \le x_i \le r_i$ such that $\sum_{i=1}^{n} x_i a_i = t$.

Exercise 6.6.

1. Show that Subset Sum With Repetition is NP-complete.

2. Show that Subset Sum restricted to the case where the elements are superincreasing (every element is greater than or equal to the sum of all of the previous elements) is in P.

3. Show that Subset Sum With Repetition restricted to the case where the elements are superincreasing is NP-complete.

4. Show that Subset Sum With Repetition restricted to the case where the elements are superincreasing and $\forall i : r_i \in \{1, 2\}$ is NP-complete.

6.3.2 3-Partition

3-Partition is a very useful strongly NP-hard problem:

> 3-Partition
> *Instance:* A multiset of integers $A = \{a_1, a_2, \ldots, a_n\}$ (n is divisible by 3) such that, $\sum_{i=1}^{n} a_i = tn/3$ and, for all i, $\frac{t}{4} < a_i < \frac{t}{2}$. We often write $t = (\sum_{i=1}^{n} a_i)/(n/3)$.
> *Question:* Can A be partitioned into $n/3$ sets $A_1, \ldots, A_{n/3}$ such that they all have equal sums? Formally, for all $1 \le i \le n/3$, $\sum A_i = t$? Each A_i has size exactly 3 (see Exercise 6.8.) The a_i's are in unary.

Garey & Johnson [GJ79] (page 96) showed the following:

Theorem 6.7. *3-Partition is NP-complete. Since the numbers are in unary it is strongly NP-complete.*

Exercise 6.8.

1. Show that in any solution of 3-Partition all of the A_i have exactly 3 elements.
2. Prove Theorem 6.7.
3. Theorem 6.7 restricts the a_i's by $\frac{t}{4} < a_i < \frac{t}{2}$. Show that 3-Partition remains NP-complete if use the restriction $\frac{7}{24} < a_i < \frac{10}{24}$. Show that, for all $\delta > 0$, 3-Partition remains NP-complete if we use restriction $\frac{1}{3} - \delta < a_i < \frac{1}{3} + \delta$.

Exercise 6.9. Give a direct reduction from 3-Partition to Partition.
Hint: First reduce directly from 3-Partition to Subset-Sum, then modify the proof to work with Partition.

We define a sequence of strongly NP-complete problems that will be useful in proving other problems strongly NP-complete.

First one is Numerical-3D-Matching. The adjective "Numerical" is because it has numbers in it (duh), in contrast to 3D-Matching, which we will study soon, that has no numbers in it. Numerical-3D-Matching is a closely related specialization of 3-Partition and is also strongly NP-hard.

> Numerical-3D-Matching
> *Instance:* Multisets $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}, C = \{c_1, \ldots, c_n\}$ of integers.
> *Question:* Is there a partition of $A \cup B \cup C$ into n sets $D_1, \ldots, D_n$ such that (1) each set has 3 elements, one from A, one from B, and one from C, and (2) each set has the same sum? The a_i's are in unary.

Garey & Johnson [GJ79] noted (Page 224) that their proof of Theorem 6.7 can be easily modified to show the following:

Theorem 6.10. *Numerical-3D-Matching is NP-complete. Since the a_i's are in unary the problem is strongly NP-complete.*

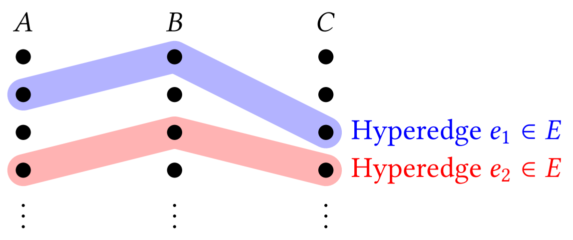


Figure 6.2: 3D-MATCHING

To explain the terminology NUMERICAL-3D-MATCHING, we recall two related problems: Three-Dimensional Matching and Exact Cover by 3-sets. We studied both of these problems in Chapter 2 and proved that there were NP-complete even in the planar version.

3D-MATCHING
Instance: a 3-hypergraph with vertices in A, B, C, disjoint, where $|A| = |B| = |C| = n$, and hyperedges $E \subseteq A \times B \times C$. (If you want to relate this problem to aliens that have 3 sexes, see the definition in Section 6.3.2.)
Question: Are there n disjoint edges that cover all of the vertices? (See Figure 6.2.)

EXACT COVERING BY 3SETS
Instance: A set X where $|X| \equiv 0 \pmod 3$ and a collection of 3-sets $E_1, \ldots, E_m$ of X. (So the input is a 3-ary hypergraph where the vertex set has size a multiple of 3.)
Question: Does there exist a set of $|X|/3$ E_i's that every elements of X occurs in exactly one of the E_i's.

Exercise 6.11. Show that 3D-MATCHING is a generalization of NUMERICAL-3D-MATCHING, and hence strongly NP-complete. Show that EXACT COVERING BY 3SETS is a generalization of 3D-MATCHING and hence strongly NP-complete.

Exercise 6.12. For each of the following problems, either (I) show that the problem is in P by giving a polynomial-time algorithm or (II) show that the problem is NP-hard by reducing one of the following to it: (a) 3-Partition, (b) 3-Dimensional Matching, or (c) Numerical 3-Dimensional Matching.

1. Given a set of numbers $A = \{a_1, \ldots, a_{2n}\}$ that sum to $t \cdot n$, find a partition of A into n sets $S_1, \ldots, S_n$ of size 2 such that each set sums to t.

2. Given a set of numbers $A = \{a_1, \ldots, a_{2n}\}$ that sum to $t \cdot n$, find a partition of A into n sets $S_1, \ldots, S_n$ of any size such that each set sums to t.

3. Given a set of numbers $A = \{a_1, \ldots, a_{2n}\}$ and a sequence of target numbers $\langle t_1, \ldots, t_n \rangle$, find a partition of A into n sets $S_1, \ldots, S_n$ of size 2 such that for each $i \in \{1, \ldots, n\}$, the sum of the elements in S_i is t_i.

Exercise 6.13. Given a graph $G = (V, E)$, we say the graph G is *beautiful* if we can color the vertices of G with either blue or red such that each vertex has **exactly one** blue neighbor. The ***Beautiful Problem*** is to, given a graph G, determine whether G is beautiful.

1. Show that the Beautiful problem is NP-complete.
   Hint: Use Exact Covering by 3Sets.

2. What happens if you restrict the Beautiful problem to planar graphs?

Exercise 6.14. Give an easy proof (not going through the Cook-Levin Theorem whose proof uses Turing machines) that Numerical-3D-Matching reduces to 3-Partition
Hint: Let N be a large number. Let a, b, c be numbers you find. Add aN to all the elements of A, bN to all the elements of B, and cN to all the elements of C.

Exercise 6.15. The ***connected bisection problem*** (CBS) is as follows. The input is a graph $G = (V, E)$ with n vertices. Determine whether V can be partitioned into two sets, each of size $n/2$ such that each part induces a connected subgraph. Show that 3D-Matching $\leq_p$ CBS, and hence CBS is NP-complete.

Now, let's use these problems to perform some hardness reductions. We'll begin with a trivial reduction.

Multiprocessor Scheduling
Instance: n jobs, with completion times $a_1, \ldots, a_n$, and p processors, each processor sequential and identical with each job running to completion on a single processor.
Question: Can all jobs be finished in time $\leq t$?

Theorem 6.16. *Multiprocessor Scheduling is strongly NP-complete.*

Proof. We show 3-Partition $\leq_p$ Multiprocessor Scheduling.

1. Input $(a_1, \ldots, a_n)$, an instance of 3-Partition.

2. Output $(a_1, \ldots, a_n)$, $p = n/3$, $t = \sum_{i=1}^{n} a_i/p$.

The proof that this reduction works is trivial. □

6.3.3 1-Planar

Definition 6.17. A ***1-planar graph*** is a graph that can be drawn in the plane with each edge crossing at most one other edge.

1-Planar
Instance: A graph G.
Question: If G 1-planar?

Grigoriev & Bodlaender [GB07] showed the following:

Theorem 6.18. *3-Partition $\leq_p$ 1-Planar, hence 1-Planar is strongly NP-complete.*

Proof sketch: The key idea in this construction is the creation of an uncrossable edge. An uncrossable edge is a subgraph (in fact, K_6) with the property that any edge passing through it must cross an edge that already has a crossing, so crossing this subgraph isn't allowed. Once we've created an uncrossable edge, we can use it as a black box to create graphs that only allow crossings in particular places.

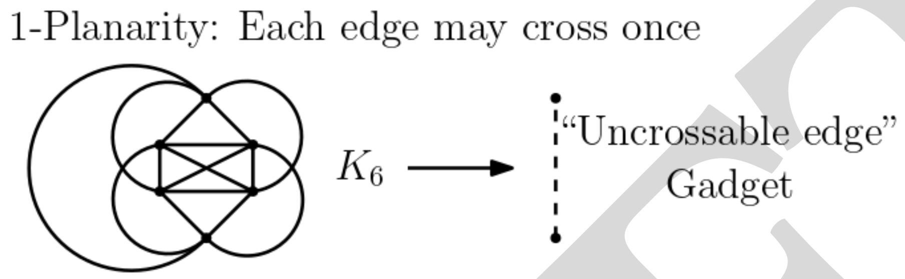

1-Planarity: Each edge may cross once
K_6
"Uncrossable edge" Gadget

Figure 6.3: The "Uncrossable Edge" Gadget

Lesson: It's often useful to create black box gadgets that you use as your notation—once you've established them, the rest of the construction becomes a lot easier.

We build two wheels out of uncrossable edges, one corresponding to the set A and one corresponding subsets of A. Using uncrossable edges, we can make each subset separate and force each a_i to belong to only one subset. Lots of cases to check, but uncrossable edges make the picture a lot clearer.

▮

6.4 Packing Problems

Packing puzzles have been around for a while and are fun! The idea is to pack given shapes (e.g., rectangles) into a bigger given shape (e.g., rectangles). Eternity [Wikc] was a complicated packing puzzle that launched in 1999 with an offer of £1,000,000 for solving it. It sold around 500,000 copies, at £35 each. It was solved by two Cambridge mathematicians (working together) Alex Selby & Oliver Riordan. Hence, one reason to study this topic is in case more money is offered for harder puzzles. But alas, we will show that such problems are generally hard.

RECT-RECT PACKING
Instance: n rectangles, target rectangle $RECT$.
Question: Can the rectangles be packed into $RECT$ with no overlap? Rotation and translations are allowed. They do not need to cover of $RECT$. (See Figure 6.4.)

Unlike many problems in this book, it is *not obvious* that RECT-RECT PACKING is in NP! This is because it is complicated to encode rotations efficiently. Is it in NP? This is an open question.

Demaine & Demaine [DD07] have shown that RECT-RECT PACKING is strongly NP-hard. In fact, they show that a certain restriction on it is NP-hard.

Theorem 6.19. *RECT-RECT PACKING where (1) the packings should be exact – no gaps, and (2) no rotations are allowed, is strongly NP-hard.*

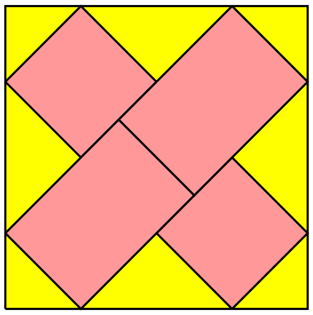


Figure 6.4: An Example of Rectangle Packing

Proof. We show that 3-Partition is reducible to this restricted version of Rect-Rect Packing.

1. Input $(a_1, \ldots, a_n)$, an instance of 3-Partition. Let $t = \sum_{i=1}^{n} a_i/(n/3)$. Recall that, for all i, $\frac{t}{4} < a_i < \frac{t}{2}$.

2. For $1 \le i \le n$ we have an $a_i \times 1$ rectangle (1 is the height).

3. Let the target rectangle be $\frac{n}{3} \times t$.

See Figure 6.5 for an example.

We view the target rectangle as being divided into $\frac{n}{3}$ mini-rectangles of height 1.

Assume that $(a_1, \ldots, a_n)$ is in 3-Partition. By renumbering we can assume that the partition is $\{a_1, a_2, a_3\}$, $\{a_4, a_5, a_6\}$, …, $\{a_{n-2}, a_{n-1}, a_n\}$. Note that $a_1 + a_2 + a_3 = t$, hence the $a_1 \times 1$, $a_2 \times 1$, and $a_3 \times 1$ rectangles can exactly pack the first target mini rectangle. Proceed like this for all $\frac{n}{3}$ triples.

Assume the target rectangle can be packed without gaps or rotations. Look at the first target mini-rectangle. Since $\frac{t}{4} < a_i < \frac{t}{2}$, the first target rectangle must be covered by exactly 3 of the $a_i \times 1$ rectangles. Renumber and assume the first mini-rectangle is covered by $a_1 \times 1$, $a_2 \times 1$, $a_3 \times 1$. Note that $a_1 + a_2 + a_3 = t$. Proceed similarly for the 2nd, 3rd, …, $\frac{n}{3}$ mini-rectangles. This yields a solution to 3-Partition. □

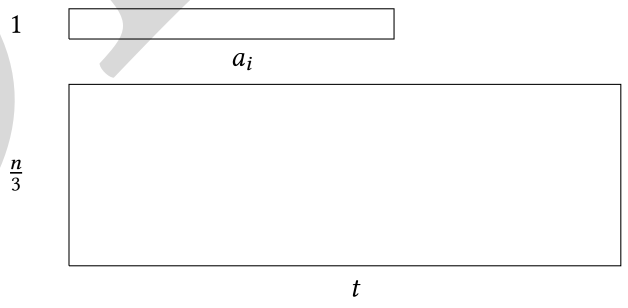


Figure 6.5: 3-Partition $\le$ restricted Rect-Rect Packing

In a similar vein: What about packing squares into a rectangle? packing squares into squares?

SQ-Rect Packing
Instance: n squares target rectangle $RECT$.
Question: Can the squares be packed into $RECT$ without overlap? No rotations are allowed. The squares do not need to cover the entire area.

Li & Cheng [LC89] showed the following.

Theorem 6.20. *SQ-Rect Packing is strongly NP-complete.*

Proof. We show 3-Partition is reducible to SQ-Rect Packing.

1. Input $A = \{a_1, \ldots, a_n\}$. Let $t = (\sum_{i=1}^{n} a_i)/(n/3)$. Note that the goal is to partition A into 3-sets that add up to t.

2. Let B be a (large) number to be named later.

3. The squares are of sides $a_1 + B, \ldots, a_n + B$. (See Figure 6.6.)

4. Let the target rectangle have height $(B + t)\frac{n}{3}$ and width $3B + t$. We think of rectangle as $\frac{n}{3}$ mini-rectangles of height $B + t$ and width $3B + t$. See Figure 6.7.

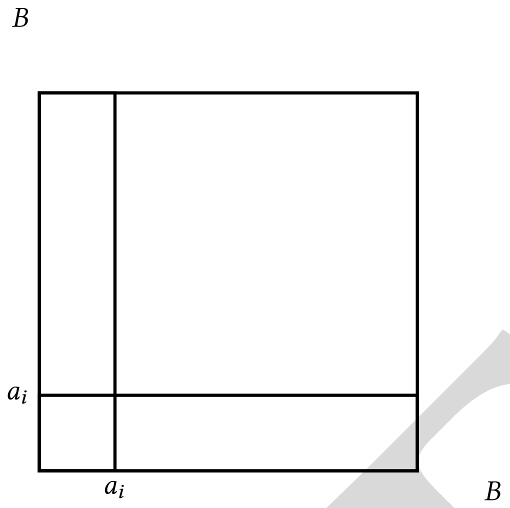


Figure 6.6: The Squares In SQ-Rect Packing Proof

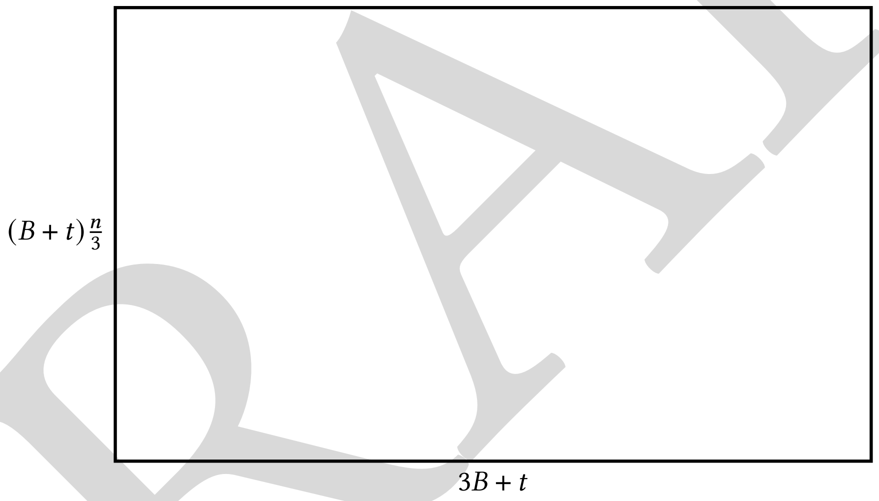


Figure 6.7: The Target Rectangle in SQ-Rect Packing Proof

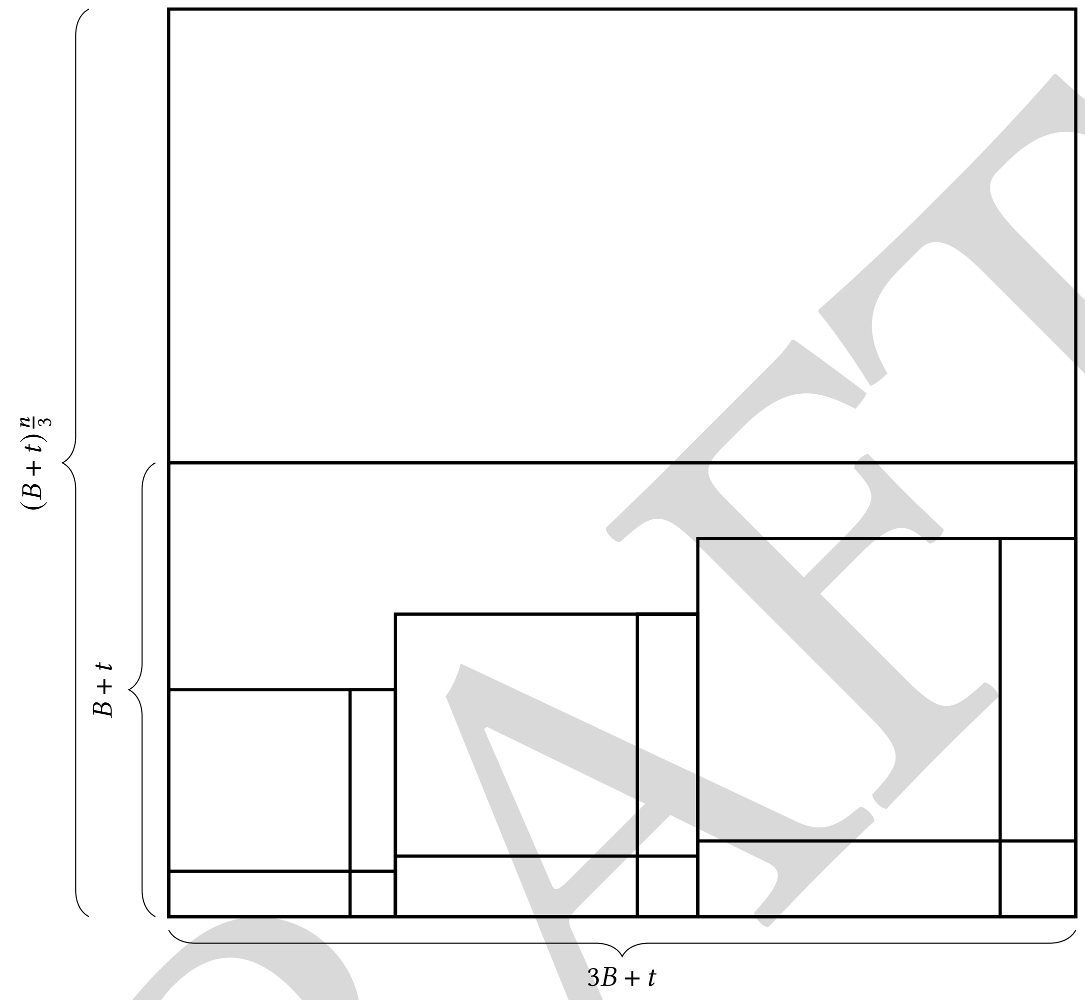


Figure 6.8: Packing Squares into 1st Mini-Rect in SQ-Rect Packing Proof

We show this reduction works.

Assume that $(a_1, \ldots, a_n)$ is in 3-Partition. By renumbering we can assume that the partition is $\{a_1, a_2, a_3\}, \{a_4, a_5, a_6\}, \ldots, \{a_{n-2}, a_{n-1}, a_n\}$. Note that $a_1+a_2+a_3 = t$ so $(a_1+B)+(a_2+B)+(a_3+B) = 3B+t$, the width of each mini-rectangle. As you can see in Figure 6.8 we can pack the squares of sides a_1+B, a_2+B, a_3+B into the first mini-rectangle. We do this for all $\frac{n}{3}$ parts to get a packing of the squares into the big rectangle.

Assume that there is a way to pack the squares into the big rectangle. Let a_1 be the min of the a_i's. Look at the first mini-rectangle. It has area $(B+t)(3B+t)$. If it has 4 squares in it then the area would be at least $4(B+a_1)^2$. We do not want this to happen, hence we have our first constraint on B:

$$(B+t)(3B+t) < 4(B+a_1)^2.$$

By renumbering assume that the first mini-rectangle is covered by the a_1+B, a_2+B, and a_3+B squares (the fact that a_1 is the min will not come into this reasoning). We can assume that they all have a side on the left boundary of the first mini-rectangle. Since the height of the mini-rectangle is $3B+t$, we have that $a_1+a_2+a_3 = t$. This reasoning holds for all of the mini-rectangles.

Warning The (say) first mini-rectangle may have part of a fourth rectangle in it using some of the left over space towards the top of the mini-rectangle. But this will be minor and the space saved will not help fit anything else in in a way we did not intend. Details are left to the reader.

□

SQ-SQ PACKING
Instance: n squares and a target square SQ.
Question: Can the squares be packed into SQ without overlap? No rotations are allowed. The squares do not need to cover the entire area.

Leung et al. [LTW⁺90] showed the following:

Theorem 6.21. *SQ-SQ PACKING is strongly NP-complete.*

Proof sketch:

We show 3-PARTITION is reducible to SQ-SQ PACKING.
We first show 3-PARTITION is reducible to SQ-RECT PACKING in a particular way.

1. Input $A = \{a_1, \ldots, a_n\}$. Let $t = (\sum_{i=1}^{n} a_i)/(n/3)$. Note that the goal is to partition A into 3-sets that add up to t. Normally we assume $\frac{t}{4} < a_i < \frac{t}{2}$; however, using Exercise 6.8 we will assume the a_i's are all within δ of $\frac{t}{3}$, where we pick δ later.

2. We will first create an instance of SQ-RECT PACKING. For each $1 \le i \le n$ we have an $a_i \times a_i$ square. The target rectangle has height $\frac{n}{3} \times \frac{t}{3} + \frac{\delta n}{3}$ and width t. See 6.9 and 6.10 for the square tiles and the rectangle.

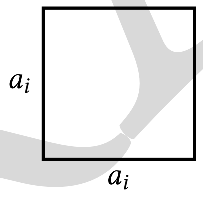


Figure 6.9: The Square Tiles

To show that this is a reduction from 3-PARTITION to SQ-RECT PACKING we consider the target rectangle to be broken into mini-rectangles of size $\frac{n}{3} + \frac{3\epsilon}{n}$. We set δ to $\frac{3\epsilon}{n}$. The argument now proceeds in a manner similar to the arguments in the proofs of Theorems 6.19 and 6.20. See 6.11.

We will now stack these rectangles to form a square. We can assume that $t < \frac{n}{3} \times \frac{t}{3} + \epsilon$. Hence we want to stack $\frac{n}{3} \times \frac{1}{3} + \frac{\epsilon}{t}$ of the rectangles on top of each each other, have $\frac{n}{3} \times \frac{1}{3} + \frac{\epsilon}{t}$ copies of each $a_i \times a_i$ square. To make sure they don't interfere with each other we will put a device between each pair of rectangles (and that will have to be added to the length and width) so that packings do not interfere with each other.

∎

Exercise 6.22. Fill in the details of the proof of Theorem 6.21.

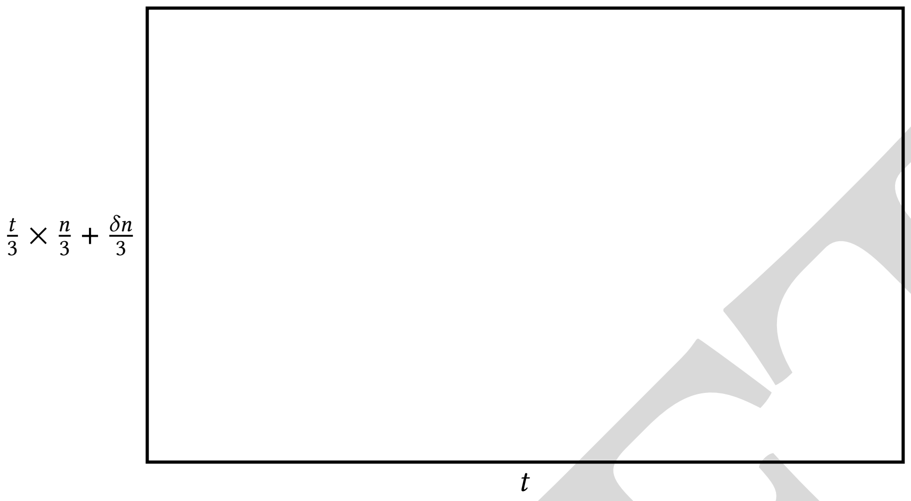


Figure 6.10: The Rectangle to Pack the Squares Into

Chou [Cho16] studied the problems of (1) packing triangles into a rectangle (Tri-Rect Packing) and (2) packing triangles into a triangle (Tri-Tri Packing). The triangles cannot be rotated.

Exercise 6.23.

1. Show that Tri-Rect Packing, restricted to right triangles by, is NP-hard.
   Hint: Use a reduction from 3-Partition.
2. Show that Tri-Tri Packing, restricted to triangles being right or equilateral, is NP-hard.
   Hint: Use a reduction from 3-Partition.
3. Show that Tri-Tri Packing, restricted to triangles being equilateral, is NP-hard.
   Hint: Use a reduction from 4-Partition.

6.5 Puzzles

We will show that four puzzle problems are equivalent and are all strongly NP-complete. All the results in this section are by Demaine & Demaine [DD07].

6.5.1 Edge Matching

The following puzzle goes back to the 1890's you are given triangles with half-frogs on them, and the goal is to pack pack these triangles with half frogs on them into a larger triangle such that the cranial and caudal ends of each frog match up (front, hind). Note that for this puzzle we want adjacent edges to be different, but in a very definite way. We will mostly look at edge matching puzzles where you want the adjacent edges to have the same color.

Eternity II [Wikb] was a complicated edge matching puzzle that launched in 2007 with an offer of £2,000,000 for the first solution before December 31, 2010 (it was made by the same people who made Eternity). We have not been able to find out how much it sold for or how many sold; however (1) it is available on E-bay for around \$40.00, and (2) it seems to have never generated

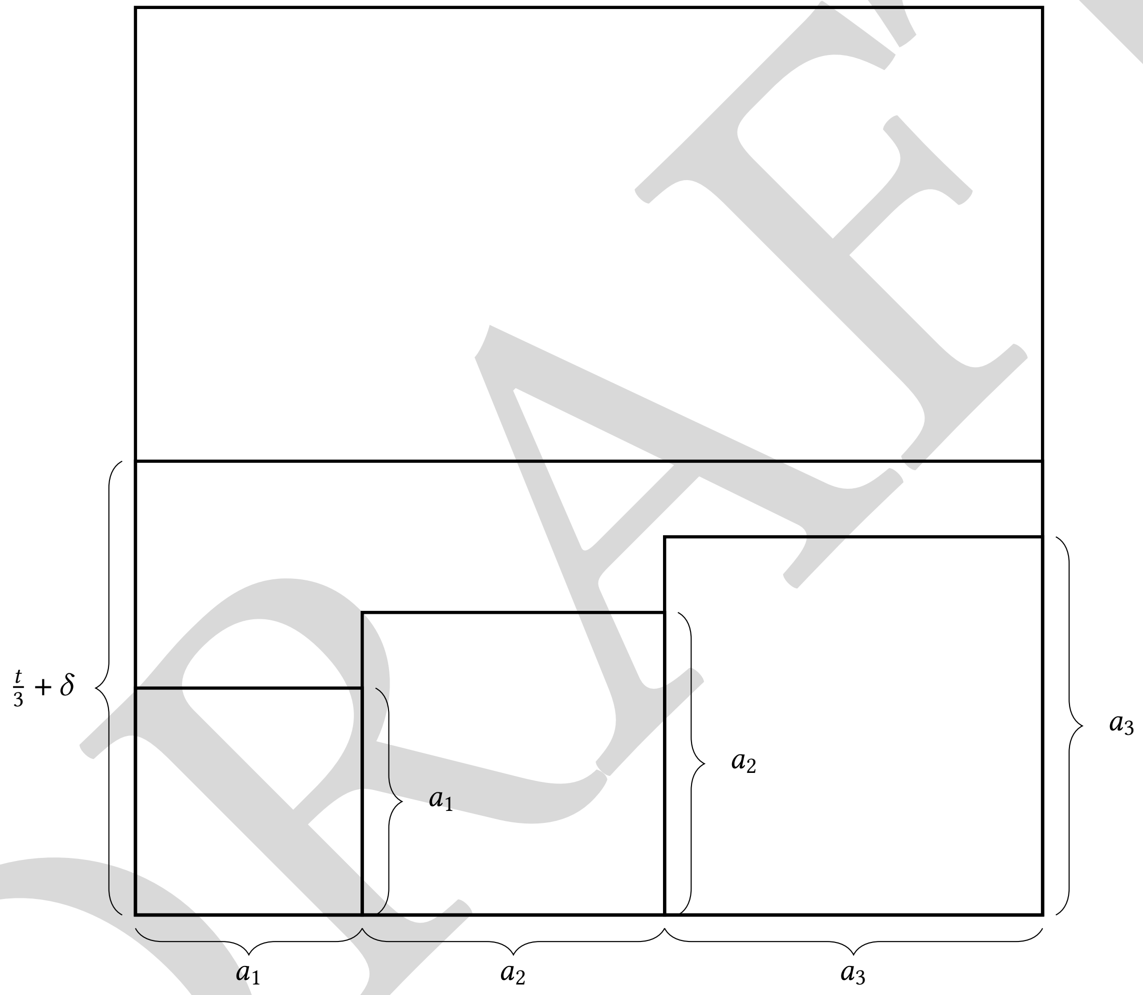


Figure 6.11: Packing the Squares into the Mini-Rectangles

the same buzz as Eternity. Eternity II was never solved (though of course the sellers know how to solve it), however, Louis Verhaard had a partial solution where he placed 467 of the 480 pieces. For this he received \$10,000. So a good reason to study these puzzles is that they may be lucrative. On the other hand, we will show that solving them is hard.

EDGE MATCHING
Instance: A set of unit squares with the edges colored, and a target rectangle RECT.
Question: Is there a packing of the squares into RECT, which fills the entire space, such that all tiles sharing an edge have matching colors. The colors are unary numbers, hence the problem will be shown strongly NP-complete. (These tiles are called ***Wang Tiles*** and were introduced by Hao Wang to study problems in logic.)

Since all the squares are of unit size (so the same size) we cannot immediately reduce from 3-PARTITION. Instead, we will first construct gadgets composed of tiles that are forced to be joined together in a particular way.

Theorem 6.24. *3-PARTITION $\leq_p$ EDGE MATCHING, hence EDGE MATCHING is strongly NP-complete.*

Proof. We first use unique, ***unmatchable*** colors to force tiles to the outer wall of B, thus creating a ***frame***. With the borders occupied, all remaining edges *must* be matched with another tile. Thus, we may use uniquely colored pairs of edges to force tiles together, creating any composite shape we like.

Here is the reduction.

1. Input $a_1, \ldots, a_n$. Let $t = \sum_{i=1}^{n} a_i$. We can assume that, for all i, $\frac{t}{4} < a_i < \frac{t}{2}$.

2. For each i create a set of a_i colored tiles that must be joined together as in the right part of Figure 6.12.

3. Create a set of colored tiles that must go together to form the frame in the left part of Figure 6.12. Color the frame's inner horizontal edges and the pieces' outer horizontal edges a single color, and a use a second color for the vertical edges. This forces all pieces to be horizontally aligned, but does not otherwise restrict the arrangement of gadgets in the frame. The frame will have an empty rectangle of height $\frac{n}{3}$ and height $t/(n/3)$.

4. The frame gives the dimensions of RECT.

Clearly if $(a_1, \ldots, a_n) \in$ 3-PARTITION then the edge puzzle problem has a solution.

We prove the converse. If there is a solution to the edge puzzle problem, all we need to show is that each row has exactly 3 of the a_i-shapes. This holds because $\frac{t}{4} < a_i < \frac{t}{2}$.

The left part of Figure 6.12 shows the frame. Unique colors a-x force the frame tiles to go on the outer edge of B, while the light and dark green edges put them on a particular side. The red/pink/orange edges force a single piece gadget together. The dark/light blue edges prevent the piece gadgets from rotating.

□

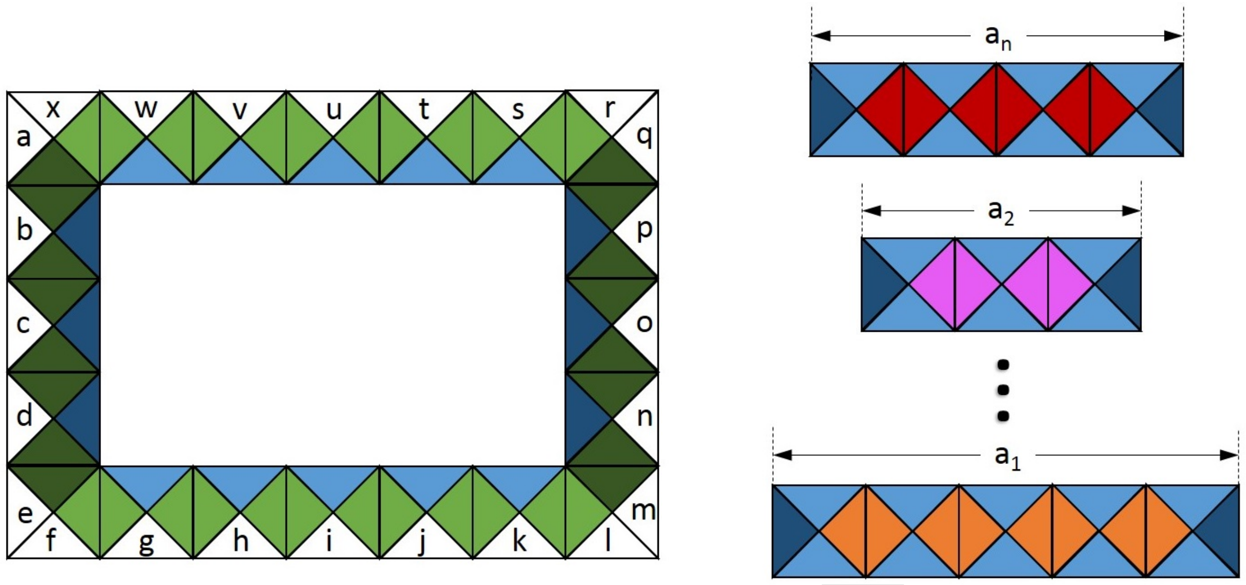


Figure 6.12: The Frame and the a_i-Gadgets

In the above reduction we create an instance of EDGE MATCHING with $\Theta(\sum_i a_i)$ tiles. Thus, for the reduction to remain polynomial we need that each a_i is polynomial in the input size n. The strong NP-hardness of 3-PARTITION guarantees it is still NP-hard under this restriction. If we tried to reduce from PARTITION it would not work because a_i could be exponential.

6.5.2 SIGNED EDGE MATCHING PUZZLE

SIGNED EDGE MATCHING PUZZLE (SIGNED EDGE MATCHING)
Instance: A set of unit squares with the edges colored, a set of pairs of colors, and a pairing of all the colors (we use a and A as a pair). We will call two colors that are a pair ***complementary***.
Question: Is there a packing of the squares into RECT such that all edges between two squares have complementary colors? The colors are unary numbers which is why when we show it is NP-complete it will be of interest that it is strongly NP-complete.

In SIGNED EDGE MATCHING we again must pack unit tiles with colored edges into a rectangle RECT. However, now colors come in pairs: a&A, b&B, etc. A color may no longer be matched with itself, but instead must be matched with a buddy color.

Theorem 6.25. *EDGE MATCHING $\le_p$ SIGNED EDGE MATCHING, hence SIGNED EDGE MATCHING is strongly NP-complete.*

Proof. Here is the reduction.

1. Input a set of colored unit tiles and a rectangle RECT. We refer to these unit tiles as ***unsigned tiles***.

2. For each unsigned tile, create a 2×2 ***supertile*** of signed tiles as shown in Figure 6.13. These will be the tiles for our instance of Signed Edge Matching.

3. The pairs of colors are $\{a, A\}$, $\{b, B\}$, etc.

4. Let RECT' be RECT with each side doubled. RECT' will be the rectangle in our instance of Signed Edge Matching.

It is easy to see that given a solution to the unsigned puzzle, we may give a solution to the signed puzzle by arranging the supertiles in like manner. To show the converse, first observe that because of the unique color-pairs used internally, all tiles must form into their intended supertiles (or partial supertiles, if on the border of B). Furthermore these supertiles must be aligned along a grid with spacing 2 (otherwise unfillable space is created). If the grid matches up with the borders, then all tiles belong to proper supertiles, and by replacing each supertile with the corresponding unsigned tile, we get a valid solution to the unsigned puzzle.

If the grid does not match up with the borders, so that there are partial supertiles on the borders, then we may fix this it to by shifting all tiles up and/or left by 1 tile (Figure 6.14). All tiles now outside of B may be recombined with their proper supertiles (on the bottom and/or right rows) to form a proper grid. The external coloring of the supertiles implies that since the partial supertiles matched on the bottom and/or left, the completed supertiles must also match. Again, this easily translates to an unsigned solution, and the proof is complete. □

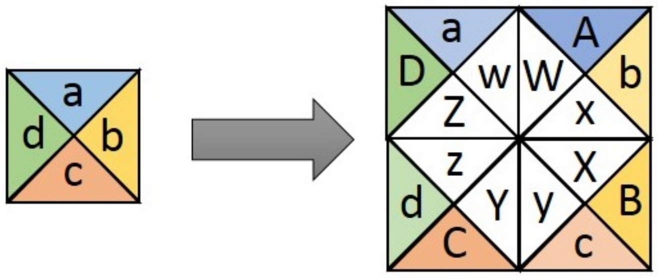


Figure 6.13: Creating a 'Supertile' of Signed Tiles from an Unsigned Tile. The internal colors are unique to this supertile

6.5.3 Jigsaw

Consider a classic jigsaw puzzle, except with no guiding picture, and ambiguous mates (i.e. there may be more than one piece that fits any particular tab or pocket). Such a jigsaw puzzle is very similar to a signed color-matching puzzle. In the jigsaw puzzle, instead of pairs of colors, each side of a piece has either a tab or a pocket. A tab only matches a pocket if it its shape is the exact inverse of the other. Also, some of the pieces have flat edges on 1 or 2 sides, which must be placed at the border.

Exercise 6.26.

1. Define Jigsaw rigorously.

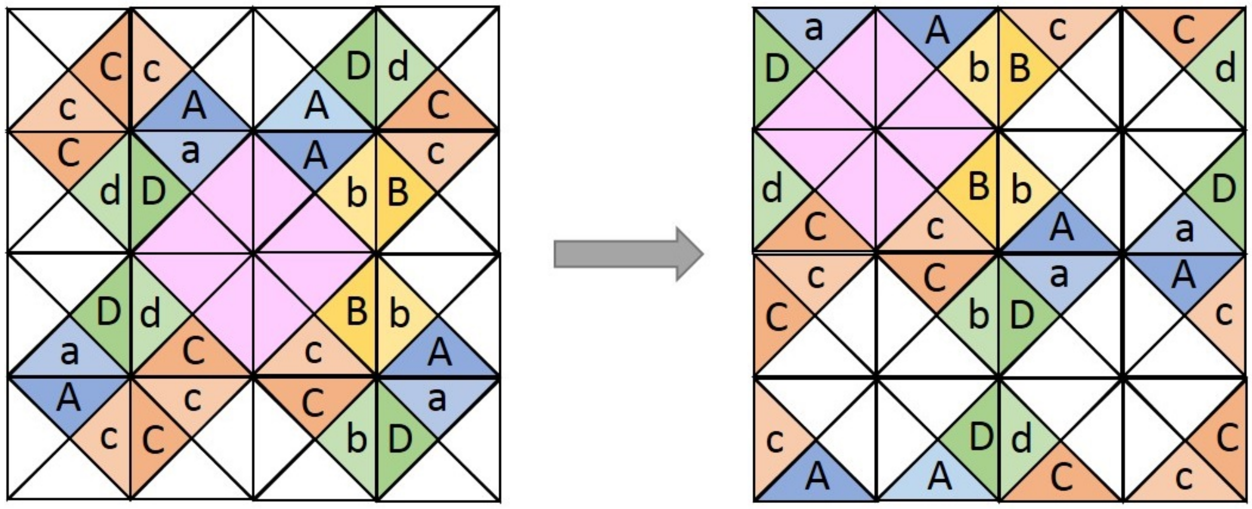


Figure 6.14: Aligning a mis-aligned solution by shifting left and up by 1. In this case, the supertile fragments directly across from each other line up. In general they may need to be rearranged, but are still guaranteed to fit.

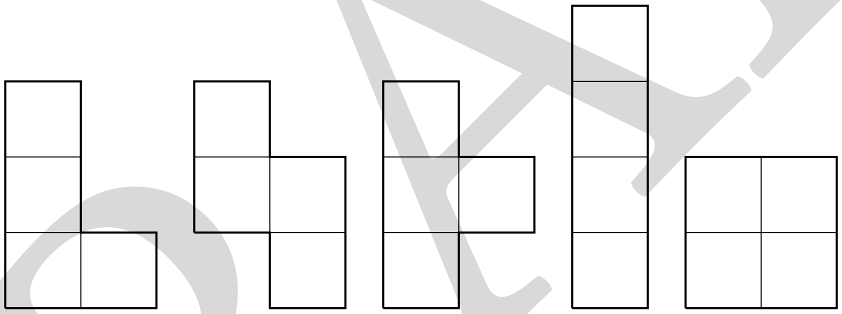


Figure 6.15: The 5 Polyominos with 4 Squares that are used in Tetris

2. Prove Theorem 6.27.

Theorem 6.27. *Signed Edge Matching $\leq_p$ Jigsaw, hence Jigsaw is strongly NP-complete.*

Note: Let Signed Edge Matching′ be the problem of Signed Edge Matching on an $a \times b$ board where there are exactly $2(a+b)$ 'unmatchable' edges (i.e. it is obvious which edges must lie on the border). This special case may be seen to be NP-hard by digging into the previous reduction steps 3-Partition to Signed Edge Matching to Signed Edge Matching. See Demaine & Demaine [DD07] for details.

6.5.4 Polyomino

Definition 6.28. A ***polyomino piece*** is a set of unit squares glued together on the edges. See Figure 6.15 for the 4 polyominos used in Tetris.

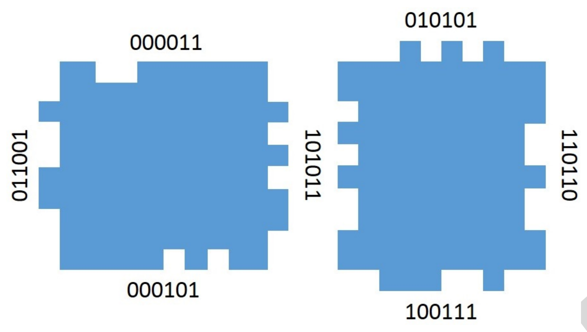


Figure 6.16: Encoding of Jigsaw Pieces into Polyominos

Polyomino
Instance: n polyomino pieces and a target rectangle RECT.
Question: Can the polyomino be packed into RECT without overlap and no gaps? All orthogonal rotations are allowed.

Polyomino is a generalization of Rect-Rect Packing (actually exact Rect-Rect Packing) where each piece is a ***polyomino***. We already know it is NP-hard because it contains Rect-Rect Packing as a special case. However, the following reduction shows the same hardness even when all polyominos are *small.*

Theorem 6.29. *Jigsaw $\leq_p$ Polyomino with pieces of area $O(\log^2 n)$ where n is the number of Jigsaw pieces. Hence Polyomino with this bound on the pieces is strongly NP-complete.*

Proof. Here is the reduction.

1. Input a set of Jigsaw pieces and a target rectangle RECT.

2. First map each type of tab/pocket to a unique binary string b of length L (we determine L later). Now create a polyomino for each jigsaw piece as follows (see Figure 6.16 for example). Start with a solid square of size $(4 + L) \times (4 + L)$. For jigsaw sides with tab type b, append unit squares along the polyomino side corresponding to the position of 1's in b, going in a clockwise direction. For pockets of type b, make unit square cuts into the polyomino, going in a counter-clockwise direction. Leave each 2×2 square at the corner alone as padding. In this way, two polyomino edges fit together with no blank space if and only if their corresponding jigsaw edges fit together. Since this is exact packing, any blank space left between two polyominoes will be unfillable by our large polyominoes. Flat jigsaw edges map to flat polyomino edges. Scaling the container dimensions up by $(4+L)$ completes the reduction. If n is the number of jigsaw pieces, then there are $O(n)$ types of tabs and pockets, so we need L to be $O(\log n)$.

This shows it is NP-hard to decide Polyomino with pieces of dimension $O(\log n) \times O(\log n)$, and thus area $O(\log^2 n)$.

In fact, Polyomino remains NP-hard even when pieces have area $O(\log n)$. In the previous reduction, each edge has length $\Theta(l)$ and gets L 'bits' of information for encoding its jigsaw edge type. However, there are $\Theta(L^2)$ 'pixels' in the polyomino which comprise the solid center. What if we could instead give each edge $\Theta(L^2)$ bits to encode its edge type? In fact, Figures 6.17 and 6.18 show one potential polyomino design which accomplishes this. Now, using edge lengths of size only $\Theta(\sqrt{\log n})$, we still get $\Theta(\log n)$ bits of information per edge: enough to encode the jigsaw edge type. This shows that POLYOMINO is NP-hard to decide for pieces of size $O(\sqrt{\log n} \times \sqrt{\log n})$ and area $O(\log n)$.

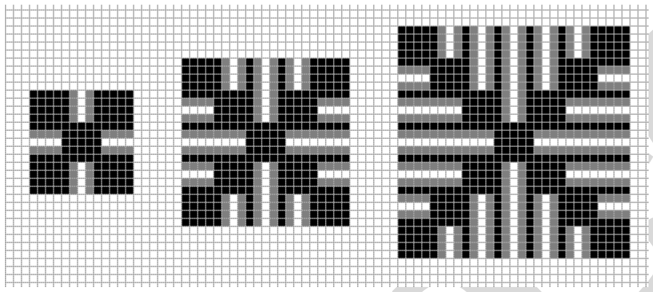


Figure 6.17: A Polyomino Gadget

(In Figure 6.17 the grey area represents free 'bits' which may be used to encode the tab type. The black areas always remain solid, and guarantee that the polyominoes are contigous. (Likewise the white areas determine the shape of the 'key' and also are shaped to remain contiguous.) In the scheme depicted, a square with sidelength $8r + 5$ has $8r^2$ grey bits available for each edge.) □

6.5.5 Closing the Loop

The following is a slight modification of the reduction from 3-PARTITION to SIGNED EDGE MATCHING

Theorem 6.30.

1. *POLYOMINO $\leq_p$ EDGE MATCHING.*

2. *3-PARTITION $\leq_p$ EDGE MATCHING $\leq_p$ SIGNED EDGE MATCHING $\leq_p$ JIGSAW $\leq_p$ POLYOMINO $\leq_p$ EDGE MATCHING where all the reductions have blowup in size at most logarithmic. (This follows from the earlier Theorems in this section. We note that the reduction from JIGSAW to POLYOMINO has log-blowup.)*

Proof. As in Theorem 6.24 we create an outer frame gadget and a gadget for each polyomino piece. Previously, the piece gadgets were $1 \times x$ rectangles. Now, we will create arbitrary polyominoes by fusing many tiles together in the same shape as the polyomino. To make sure the shape is preserved, use a unique color pair for each internal edge of the gadget. (This is the same method we used to force supertiles to stick together - see Figure 6.13.) Lastly, in the gadgets in Figure 6.12, we used dark and light blue for both the internal edges of the frame, and the external edges of each piece, in order to keep the pieces from rotating. In polyominoes, we want the pieces to be freely rotatable, so just use one color instead of 2 for these edges. □

Thus, all four puzzle types are essentially reducible to one another and strongly NP-complete, as shown in Figure 6.19.

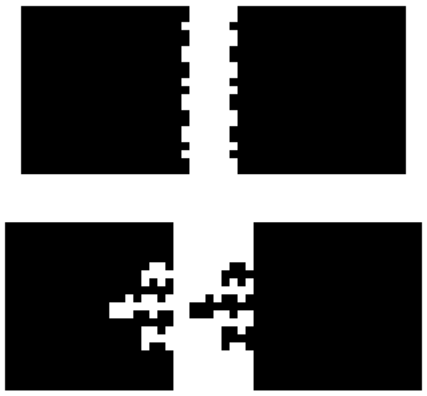


Figure 6.18: Deeper 'Keys' Allows for More Possible Combinations than Bumps and Grooves on the Edges

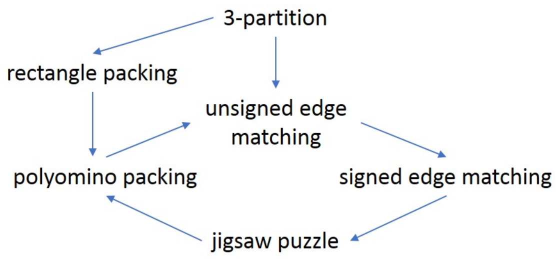


Figure 6.19: Each Arrow Represents a Reduction From one Problem to Another

6.6 Overview

We have established the hardness of two fundamental problems, 3-Partition and Partition. We have exhibited a bunch of reductions from those problems to other numerical and geometrical ones.

We now continue with reductions from 3-Partition and Partition to geometrical problems—we'll also use the fact that the problem of packing n squares into a square without rotations is strongly NP-complete, as we showed last time.

6.7 Three Dimensional Games

In this section we reason more informally than usual. This is because formally defining these problems leads to long definitions that are not enlightening. In addition, having read this far you have a sense of how reductions work.

6.7.1 Edge Folding Polyhedra

Edge Folding Polyhedra
Instance: A polyhedra.
Question: Can the polyhedra be cut along its edges to unfold it into a connected flat piece with no overlapping sections?

One might thing that every polyhedra can be cut as such; however, Abel & Demaine [AD11] have examples of polyhedra that cannot be so cut. Hence the problem is non-trivial. See Figure 6.20 for an example of a YES instance and a discussion of what could cause a NO instance.

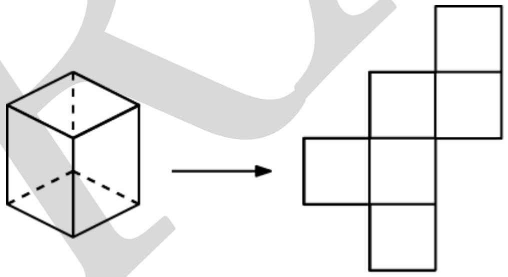


Figure 6.20: Unfolding a Cube

Figure 6.20 gives an example of unfolding a cube. We can always cut along a minimum spanning tree of the edges of a polyhedron; the difficulty lies in cutting so that the result lies flat.

Question: There don't seem to be any numbers here, so in what sense is this problem *strongly* NP-complete?

Answer: There are numbers encoded in the coordinates and sizes of the faces, and in this construction they are all polynomially large.

Abel & Demaine [AD11] showed the following.

Theorem 6.31. *SQ-SQ* Packing $\leq_p$ *EFP,, hence* Edge Folding Polyhedra *is strongly NP-complete.*

Proof sketch: The intuition is as follows: the infrastructure of the polyhedron will contain a large square hole, with a lot of smaller square faces. Unfolding the polyhedron without overlap will require to fitting those square faces into the large square hole, which is exactly the reduction we seek.

However, in a standard square packing problem we're allowed to move the square s around more or less freely within the larger square. Squares on the faces of a polyhedron are obviously quite constrained in their movements, so how do we construct squares that can move freely?

The most important "glue" in the construction is the idea of an ***atom***, a gadget that sits on the face of a polyhedron and allows pieces of the unfolding to mover arbitrarily. Atoms are "bumpy squares" that can go left/right on the surface of the polyhedron and move a *different* direction when they're unfolded. Connecting the squares with atoms solves the problem of moving each of the squares without constraint—see the paper by Abel & Demaine for details. ∎

6.7.2 Disk Packing

Disk Packing
Instance: A set of disks and a square.
Question: Can you place the disks so that all of their centers are in the square?

This is a real life problem, in the sense that a disk packing algorithm can design origami structures by the tree method. People have thought about this problem a lot. Demaine et al. [DFL10] showed the following.

Theorem 6.32. *3-*Partition $\leq_p$ *DISKPACKING.*

Proof sketch:

Begin by thinking about packing an equilateral triangle rather than a square. We set up a triangular "grid" of large congruent disks (disks with the same diameter), and then build 3-Partition gadgets in the space between them, each of which will represent one of the subsets in our partition.

We reduce the elements a_i to circles with diameters based on the size of a_i in such a way that three circles corresponding to a_i, a_j, and a_k fit if and only if $a_i + a_j + a_k \leq t$.

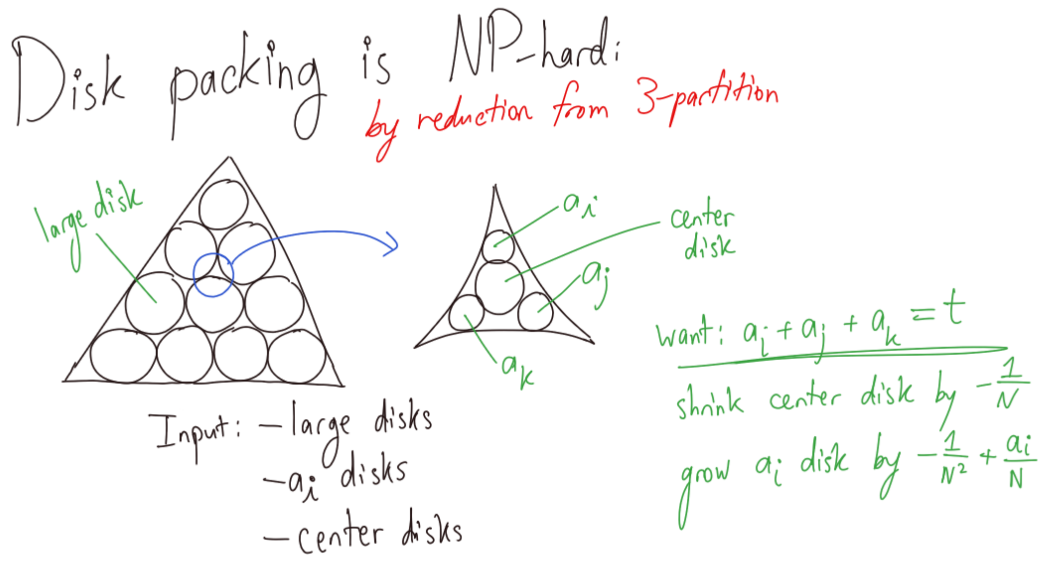

Disk packing is NP-hard:
by reduction from 3-partition
large disk
a_i
center disk
a_j
a_k
want: $a_i + a_j + a_k = t$
shrink center disk by $-\frac{1}{N}$
grow a_i disk by $-\frac{1}{N^2} + \frac{a_i}{N}$
Input: – large disks
– a_i disks
– center disks

Figure 6.21: Disk Packing in a Triangle.

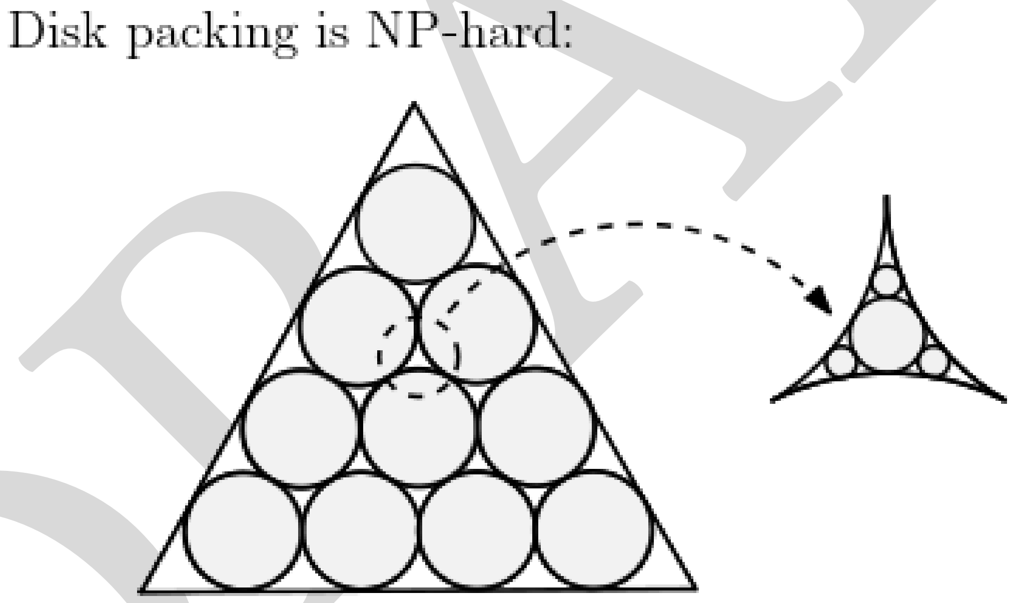


Figure 6.22: Disk Packing in a Triangle.

Working this out requires some subtlety because circles aren't straight (they're circles, after all), so sums don't map so clearly to the dimensions of circles. But they're like segments up to first order, so we use Taylor expansions to calculate the proper radii. In particular, we shrink each circle by an amount $-1/n^2 + a_i/n$, where the $1/n^2$ term cancels higher-order terms in the expansion.

Comment: The point of showing you all this is to give you lots of different ways to represent numbers. The more ways you have of representing numbers, the easier it will be to find reductions from 3-Partition to geometric (and other) settings.

Finally, we have the issue of packing everything into a square. The general idea is to fill up the grid with lots of circles of decreasing sizes in $O(\log n)$ steps, which constrains the gadgets to sit in the right place. By considering circles in decreasing size, we can be convinced at every stage that infrastructure circles are constrained.

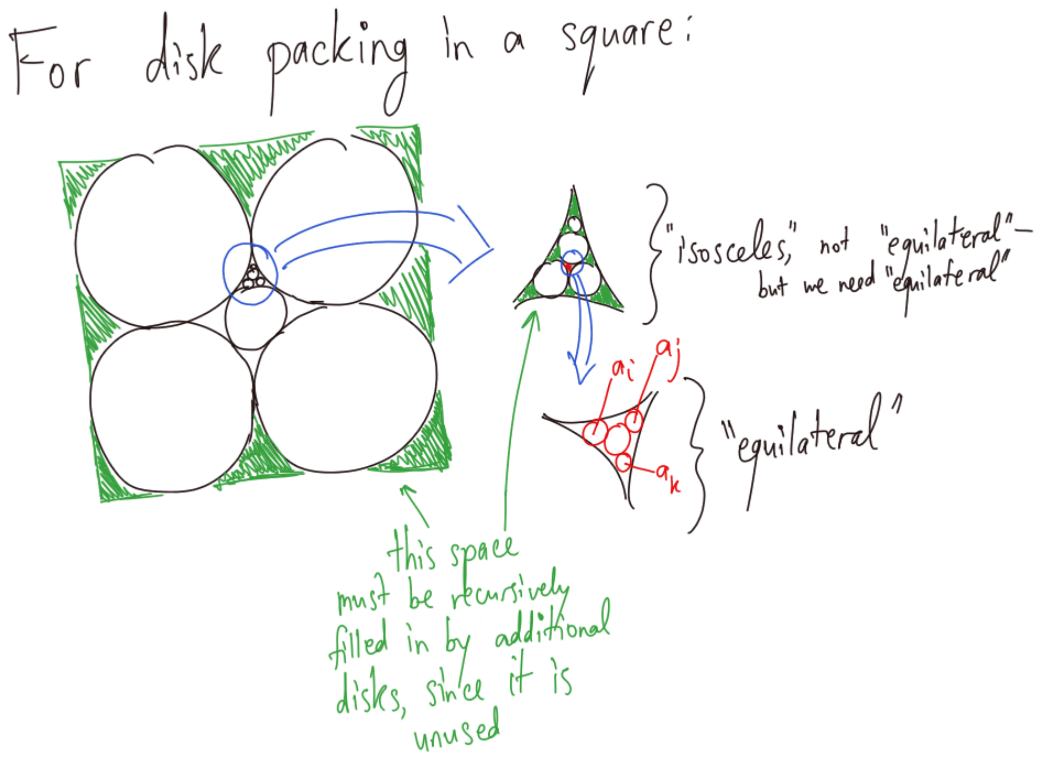

For disk packing in a square:
"isosceles," not "equilateral"— but we need "equilateral"
a_i
a_j
a_k
"equilateral"
this space must be recursively filled in by additional disks, since it is unused

Figure 6.23: How to Adapt Disk Packing to a Square Instead of a Triangle.

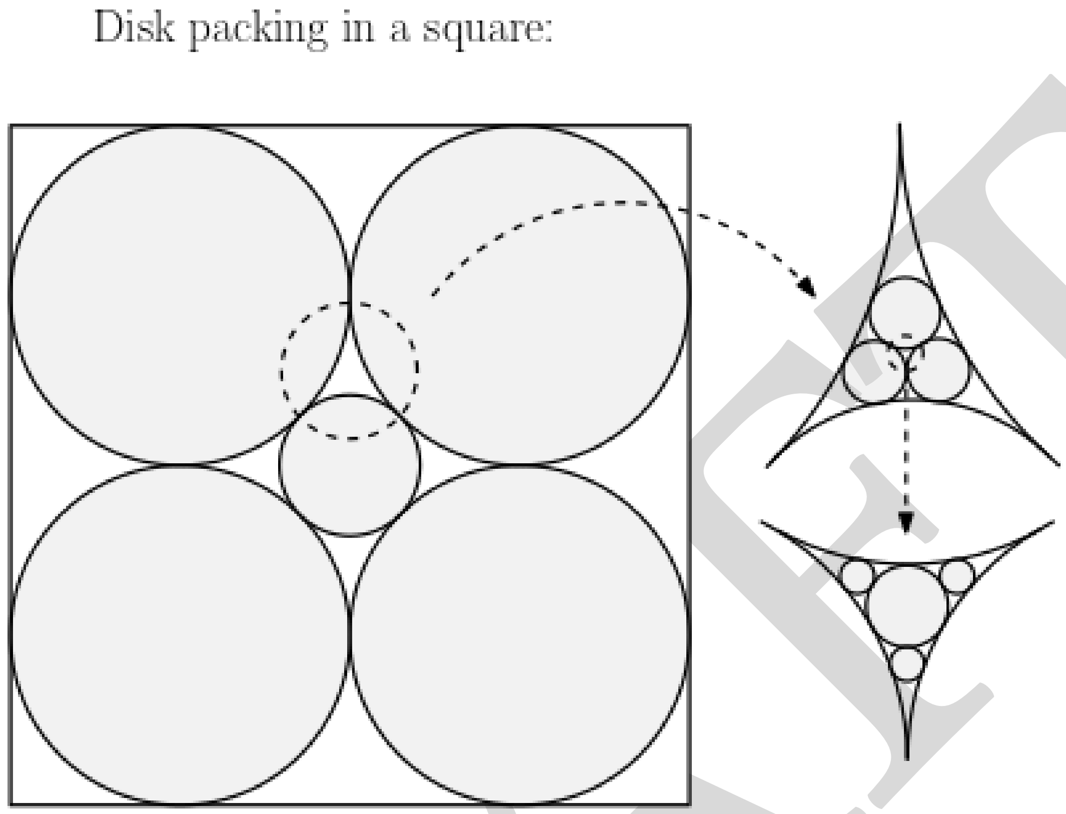


Figure 6.24: How to Adapt Disk Packing to a Square Instead of a Triangle.

■

6.7.3 Clickomania

Clickomania is a computer game. We describe it informally (a more detailed description can be found on the web). The goal is to clear blocks by clicking contiguous groups of n blocks ($n > 1$) of the same color, which is thereby destroyed. Blocks fall column by column to fill up empty space, and empty columns disappear from the board. Solving Clickomania with only one column is in P, by reduction to a context-free grammar. However, Biedl et al. [BDD+01] showed the following.

Theorem 6.33. *3-Partition $\leq_p$ Clickomania, even when Clickomania is restricted to (1) 2 columns and 5 colors, or (2) 5 columns and 3 colors. Hence these problems are strongly NP-complete.*

Proof sketch:

In the 2 column/5 color case, we set it up so that the right column encodes the a_i, and the left column encodes the desired sum t of each partition.

The left column is mostly a symmetric checkerboard pattern interspersed with red squares; the only way to clear the entire column is to clear all the red squares (and thereby solve 3-Partition). Clicking on blocks in the right column corresponds to choosing elements for one subset A_i; when that subset reaches the desired sum, two red blocks in the left and right columns line up exactly and we can clear both. There are a lot of small numerical details to check, but otherwise the construction is straightforward.

∎

6.7.4 Tetris

Tetris is a computer game played on a rectangular board with falling tetronminos which are the polyomino in Figure 6.15 that we saw in the section on Polyominos. Each block can be rotated as it falls from the sky, and filled rows disappear. You lose if your pile of blocks ever reaches the top edge of the board (the "sky").

> Tetris
> *Instance:* An initial board and the sequence of future pieces.
> *Question:* Is there a sequence of moves so that you win?

Note that this is a problem with perfect information: we know everything about the current and future states of game. The actual game is an online problem where you must make decisions with incomplete information. Hence it would seem that Tetris is easier than the real game. Even so, Biedl et al. [BDH+04] showed the following.

Theorem 6.34. *Assume that n is the number of pieces in Tetris. Let $0 < \varepsilon < 1$.*

1. *3-Partition $\le_p$ Tetris, so Tetris is strongly NP-complete.*
2. *Let f be the function that, given an instance of Tetris, outputs the maximum number of lines of blocks until death. It is NP-hard to output a number that is $\ge n^{1-\varepsilon} f(n)$.*
3. *Let g be the function that, given an instance of Tetris, outputs the maximum number of blocks that can be placed until death. It is NP-hard to output a number that is $\ge n^{1-\varepsilon} f(n)$.*

Proof sketch:

We have to reduce from a strongly NP-complete problem because the proof will require encoding the a_i in unary.

The starting board configuration contains gadgets called "buckets", each $\Theta(t)$ call. Filling these buckets corresponds to choosing a_i to fill your subsets. Once these buckets are filled, a single T-block can unlock an empty column to the right of the playing field, which can be filled by straight pieces. Without unlocking this column, it's impossible to survive.

The a_i's correspond to a sequence of tetrominoes which fit perfectly in buckets. A key problem for the proof is the idea of splitting: since each element a_i is a series of pieces, there's the danger that the player will split the a_i's into two or more buckets, which ruins the correspondence between Tetris and 3-Partition. The solution to this problem is to introduce a priming piece between each a_i. The priming piece "unlocks" a bucket, and trying to split an a_i won't properly prime the bucket, and won't lead to a solution.

By building this "lock" structure atop a large, empty space, we can ensure that opening the lock will increase the number of lines before dying by an exponential factor. This means that even approximating the number of lines a player can survive to within $n^{1-\varepsilon}$ is NP-hard.

∎

Figuring out how hard Tetris is in any of the following situations is still open:

- Tetris with an initially empty board
- Tetris with $O(1)$ rows or columns
- Tetris with restricted piece sets (For instance, Tetris with only straight pieces is trivially easy. What subset of pieces is required to make Tetris hard?)
- Tetris without last-minute slides (where ever piece needs to be dropped from the sky)
- Online Tetris (Tetris where you don't know the future sequence of pieces)
- 2-Player Tetris (which might be PSPACE-complete)

6.7.5 Ivan's Hinge

Ivan's Hinges are hinged loops of colored triangles which can be folded into colored shapes in the plane (we will not define this formally). Abel et al. [ADD+14] showed that deciding whether a puzzle can be folded into a certain shape is NP-complete, by another reduction from 3-Partition

This problem also involves finding universal shapes that can form any other shape (in this case, called the GeoLoop), and the a_i are represented as colored blocks in the final construction. Choosing subsets corresponds to filling up contiguous regions of colored blocks, with the rest of the loop dedicated to allowing the a_i to move around freely.

6.7.6 Carpenter's Rule

An "annoying" Carpenter's Rule is a foldable ruler of unit width, where each straight section can double back on the next (we will not define this formally). The goal is to fit the ruler into a long straight box of a specified size (we will not define this formally) Hopcroft et al. [HJW85], showed this problem is *weakly* NP-hard, and, like Partition, admits a pseudo polynomial algorithm.

The reduction is from Partition. Choosing which of the two subsets a_i should fall into corresponds to choosing whether a segment of length a_i goes left or right. If the two subsets are of equal size, then our ruler will start and end at exactly the same point.

Adding two long segments and two slightly shorter segments at each end, we can reduce from the problem of finding two equal subsets to the problem of fitting the ruler exactly into the box.

6.7.7 Map Folding

The problem is to decide whether a map and a given pattern of creases is foldable (we will not define this formally). Arkin [ABD+04] showed this problem is *weakly* NP-hard for orthogonal paper and orthogonal creases or for rectangular paper and orthogonal and diagonal folds. It is not known whether this problem is strongly NP-hard or if there exists a pseudo polynomial algorithm.

The reduction as from Partition and is similar to the reduction for Carpenter's Rule. There are horizontal folds corresponding to the a_i and two vertical folds corresponding to the division between subsets. We can fold some subset of the a_i first, then fold the vertical folds, and then fold the remaining a_i. The creases will match up perfectly if and only if the two subsets have the same size.

This construction uses "orthogonal paper," which doesn't much resemble a traditional map. But adding diagonal creases to a rectangular piece of paper can force it to fold into an orthogonal shape, which can then be used to encode an instance of partition using our reduction for orthogonal paper.

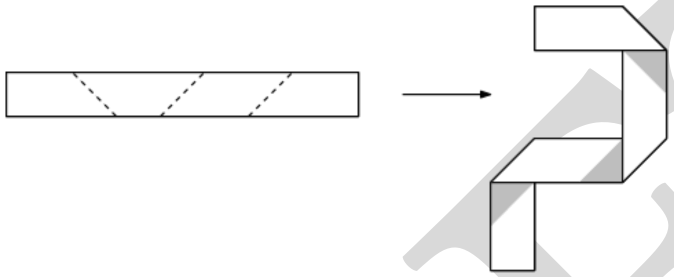


Figure 6.25: How to Form "Orthogonal Paper" From Long Rectangular Paper Using Diagonal Creases.

6.8 Further Results

We list problems that were proven strongly NP-complete by a reduction from 3-Partition. You can either consider these as exercises (so to not look up the references) or recommended reading (so do look up the references). We state them as function problems; however, the reader can translate them to set problems for the reductions.

1. (Mauro Dell'Amico & Silvano Martello [DM99]) BP (Natural Number Version). Given $a_1, \ldots, a_n \in \mathbb{N}$ and bin capacity $C \in \mathbb{N}$, find the least number of bins B so that $a_1, \ldots, a_n$ can be packed into B bins of capacity C.

2. (Mauro Dell'Amico & Silvano Martello [DM99]) 0-1 Multiple Knapsack Problem. Given (1) n items, each represented by an ordered pairs of naturals (p_i, w_i) (p_i is the profit of the ith item, w_i is the weight of the ith item), (2) m knapsacks, each represented by its capacity c_j. Pack the items into the knapsacks such that the weight in each knapsack is $\leq$ the capacity, and the profit is maximized.

3. (Jiang et al. [JZZZ12]) Minimum Common String Partition A ***partitioned string*** is a string where you are also given a way to break it up. So you are given x and $(x_1, \ldots, x_m)$ where $x = x_1 \cdots x_m$. A ***common partition*** of strings x, y is a way to partition them so that each partition is a permutation of the other. Finally, here is the problem: Given strings x, y determine whether there is a common partition, and if so then find it. They showed that the even if the alphabet size is 2, there is a reduction from 3-Partition.

4. (Bernstein et al. [BRG89]) Scheduling Problems for Parallel/Pipelined Machines. Imagine that there are two processors P_1, P_2 and two types of jobs, so that P_j can process jobs of type j. The problem: Given n jobs $J = \{J_1, J_2, \ldots, J_n\}$ where each job is represented by $J_i = (K_i, T_i, D_i, R_i)$ where K_i is the job type, T_i is the execution time in unary, D_i is the time

a job needs before it begins processing (called the Delay Time), and R_i resource requirement (this is either 0 or 1 and you cannot have two jobs with resource requirement 1 being processed at the same time). We are also given $G = (J, E)$ be the precedence graph which specifies which job should be executed before the other one. Find a way to schedule the jobs so as to minimize the the total completion time of all jobs.

5. Roemer [Roe06] and Bachman & Janiak [BJ04] have also looked at problems with job scheduling that are strongly NP-complete via a reduction from 3-Partition.

6. (Cieliebak & Eidenbenz [CE04]) Measurement-errors. Given $m, \Delta \in \mathbb{N}$, and a multiset D of $\binom{m}{2}$ natural numbers, does there exist m points in the plane, on a line, such that the multiset of pairwise difference can be mapped to D such that if $|p - q|$ maps to d then $d - \delta \leq |p - q| \leq d + \delta$. The above problem is additive error. They also showed that the problem for multiplicative error is reducible from 3-Partition.

7. (Formann & Wagner [FW90]) The VLSI layout problem. Given a graph G where every vertex has degree ≤ 4, and also given $A \in |N$, can G be embedded in a grid with area $\leq A$? (There are variants of this problem that are also strongly NP-complete.)

8. (Brimkov et al. [BCLR96]) Matrix-similarity. Given an upper triangular matrix A with distinct diagonal elements, $\tau \geq 1$, is there a matrix M with condition number $\leq \tau$ such that $G^{-1}AG$ is a 2×2 block diagonal matrix.

Chapter 7

Exponential Time Hypothesis

7.1 Introduction

If Alice proved that $P \neq NP$ by showing that 3SAT required at least $n^{\Omega(\log\log\log(n))}$ (where n is the number of variables) then she would still win the Millennium prize; however, the result would not really tell us what we want to know. All NP-complete problems would be proven to take at least $n^{\Omega(\log\log\log(n))}$ which is a much weaker lower bound than what we believe to be true.

What lower bounds on 3SAT are reasonable to assume? Lets turn this around: what are the best algorithm for 3SAT? For kSAT?

1. Makino et at. [MTY13] have obtained a deterministic $O(1.3303^n)$ algorithm for 3SAT.

2. Hertli [Her14] has obtained a randomized $O(1.308^n)$ algorithm for 3SAT.

3. Dantsin et al. [DGH+02] have obtained a deterministic $O((2-\frac{1}{k+1})^n)$ algorithm for $kSAT$.

4. Paturi et al. [PPZ99] have obtained a randomized $O(2^{n-(n/k)})$ algorithm for $kSAT$.

(See the book by Schoning and Toran [ST13] for an overview of SAT algorithms.)

It is plausible that the bounds for 3SAT will be improved to $O(\alpha^n)$ for some $1 < \alpha < 1.3$. However, it seems likely that the bounds will always be of the form $O(\alpha^n)$. An algorithm running in time $O(n^{O(\log n)})$ seems unlikely. It is plausible that the bounds for kSAT will be improved to $O(2^{n-(\alpha n/k)})$ for some $\alpha > 1$. However, it seems likely that as k goes to infinity, the exponent of 2 will have limit n.

We are going to hypothesize that the best algorithm for kSAT has time complexity $2^{s_k n}$ for some s_k. But this is not quite right. It is possible (though, we think, very unlikely) that, for example,

- For all i there is an algorithm for 3SAT that runs in time $O(2^{(0.1+(1/10^i)n})$.

- There is no algorithm for 3SAT that runs in time $O(2^{0.1n})$.

This (remote) possibility leads to the following notation, which we will need to state the hypothesis.

Notation 7.1. For $k \geq 1$,

$$s_k = \inf\{s \mid \text{ there is an } O(2^{sn}) \text{ algorithm for } k\text{SAT}\}.$$

Looking at the algorithms for kSAT mentioned above note that (1) they all have time of the form 2^{cn}, and (2) as k goes to infinity, the exponent goes to n. Based on these intuitions Impagliazzo and Paturi [IP01] formulated the following hypotheses:

Hypothesis 7.2.

1. *The **Exponential Time Hypothesis (ETH)**: For all $k \geq 3$, $s_k > 0$. Since $s_3 \leq s_4 \leq \cdots$ this is equivalent to $s_3 > 0$. It is also equivalent to 3SAT requiring time $2^{\Omega(n)}$.*

2. *The **Strong Exponential Time Hypothesis (SETH)**: The sequence $s_3, s_4, \ldots$ converges to 1. We will not be using SETH; however, in Section 7.4 we list some problems that are hard assuming SETH.*

In the ETH we are using n, the number of variables, as the length of the input. What if we took the real length of the input? Impagliazzo et al. [IPZ01] showed that such a hypothesis would be equivalent to ETH and SETH. We give the key lemma.

Lemma 7.3. (The Sparsification Lemma) Let $k \in \mathbb{N}$ and $\varepsilon > 0$. There is a constant c (that depends on k, ε) and an algorithm such that the following hold when the input is a k-CNF formula φ on n variables:

1. The output is t k-CNF formulas $\varphi_1, \ldots, \varphi_t$.

2. $\varphi \in$ SAT if and only if there is an i such that $\varphi_i \in$ SAT.

3. $t \leq 2^{\varepsilon n}$.

4. Each φ_i has $\leq cn$ clauses.

5. The algorithm runs in time $O(2^{\varepsilon n} p(n))$ for some polynomial p.

Exercise 7.4. Let ETH' (SETH') be ETH (SETH) only instead of n use the actual length of the input. Show that ETH' is equivalent to ETH. Show that SETH' is equivalent to SETH.

ETH is stronger than P $\neq$ NP; hence we do not think it will be proven anytime soon. However, we believe that it is true. Assuming ETH we can prove strong lower bounds on many problems.

Definition 7.5. Let $A \leq_p B$ via f.

1. f has ***linear blowup*** if $|f(x)| \leq O(|x|)$.

2. f has ***quadratic blowup*** if $|f(x)| = \Theta(|x|^2)$. (Technically we need this equality to hold for infinitely many x.)

Exercise 7.6. For this problem assume ETH holds.

1. Assume that 3SAT $\leq_p B_1 \leq_p B_2 \leq_p \cdots \leq_p B_L$ and all of the reductions have linear blowup. Show that, for every $1 \leq i \leq L$, every algorithm for B_i runs in time $2^{\Omega(n)}$.

2. Assume that 3SAT $\le_p B_1 \le_p B_2 \le_p \cdots \le_p B_L$ and (a) one of the reductions is quadratic, but (b) the rest of the reductions are linear. Show every algorithm for B runs in time $2^{\Omega(\sqrt{n})}$.

3. Speculate on why we did not ask an analog of problem 1 for quadratic blowup.

Note the following parallel:

1. By assuming P $\neq$ NP and using polynomial time reductions, we can show many problems are not in P.

2. By assuming ETH and using polynomial time reductions with linear (quadratic) blowup we can show many problems require $2^{\Omega(n)}$ ($2^{\Omega\sqrt{n}}$) time.

7.2 ETH Yields $2^{\Omega(n)}$ Lower Bounds

By Exercise 7.6 if a reduction has linear blowup, and we assume ETH, we get a $2^{\Omega(n)}$ lower bound. Many of the reductions we have already done had linear blowup. Hence we have, assuming ETH, many $2^{\Omega(n)}$ lower bounds.

Theorem 7.7. *Assume ETH. Then the following problems require $2^{\Omega(n)}$ time to solve.*

1. *The Vertex Cover Problem. The proof of Theorem 2.14 gives a reduction with linear blowup from 3SAT to Vertex Cover.*

2. *3-colorability of graphs. The proof of Theorem 2.11 gives a reduction with linear blowup from 3SAT to 3COL.*

3. *Dominating set. The proof of Theorem 2.16 gives a reduction with linear blowup from Vertex Cover to Dom Set.*

4. *Clique. The proof of Theorem 2.3 gives a reduction with linear blowup from 3SAT to Clique.*

5. *Directed Hamiltonian Cycle. The proof of Theorem 2.19 gives a reduction with linear blowup from 3SAT to Ham Cycle.*

6. *Directed Hamiltonian Path, Hamiltonian Cycle, Hamiltonian Path. We leave this to the reader.*

Exercise 7.8. Assume ETH. Prove that Directed Hamiltonian Path, Hamiltonian Cycle, and Hamiltonian Path all require $2^{\Omega(n)}$ time.

7.3 ETH Yields $2^{\Omega(\sqrt{n})}$ Lower Bounds

By Exercise 7.6, if a reduction has quadratic blowup, and we assume ETH, we get a $2^{\Omega(\sqrt{n})}$ lower bound. Some of the reductions in this book had quadratic blowup. Hence, assuming ETH, we have some $2^{\Omega(\sqrt{n})}$ lower bounds.

Recall the proof that planar 3-colorability is NP-complete. We took the proof that 3-colorability is NP-complete and used cross-over gadgets. This caused the linear reduction for 3-colorability to be a quadratic reduction for planer 3-colorability. This is common for NP-completeness results for graph problems restricted to planar graphs.

Theorem 7.9. *Assume ETH. Then the following problems require $2^{\Omega(\sqrt{n})}$ time to solve.*

1. *3-colorability of planar graphs. The proof of Theorem 2.11 gives a quadratic reduction from 3SAT to 3COL. Alternatively there is a reduction with quadratic blowup from 3SAT to* Planar *3SAT (Theorem 2.11) and a linear reduction from* Planar *3SAT.*

2. *3-colorability of planar graphs of degree 4. The proof of Theorem 2.11 gives a quadratic reduction from 3SAT to 3COL.*

3. *Dominating set for planar graphs. The proof of Theorem 2.16 gives a reduction from 3SAT to* Planar Dominating Set *that has quadratic blowup.*

4. *Directed Hamiltonian Cycle for planar graphs. The Theorem 2.19 there is a reduction from 3SAT to* Directed Ham Cycle *which has quadratic blowup.*

5. *Vertex Cover for planar graphs. The proof of Theorem 2.14 gives*

Exercise 7.10.

1. Assume ETH. Prove that Directed Hamiltonian Path, Hamiltonian Cycle, and Hamiltonian Path restricted to planar graphs all require $2^{\Omega(\sqrt{n})}$ time.

2. Theorem 7.9 did not mention the Clique problem. What is known about Clique restricted to planar graphs?

Can we improve the lower bounds from $2^{\Omega(\sqrt{n})}$ to $2^{\Omega(n)}$? This is one of the rare times in this book we can give an answer without a hypothesis:

All of the problems in Theorem 7.9 are in $2^{O(\sqrt{n})}$.

These algorithms come from the theory of bidimensionality. See either the papers of Demaine et al. [DHT04, DFHT05], the references in those papers, or the slides of Marx [Mar13].

7.4 Some Consequences of SETH

Orthogonal Vector Problem (OrVecProb)
Instance: $A, B \subseteq \{0,1\}^d$, $|A| = |B| = n$.
Question: Does there exist $\vec{a} \in A, \vec{b} \in B$ with $\vec{a} \cdot \vec{b} \equiv 0 \pmod 2$?
Note: It is easy to see that Orthogonal Vector Problem can be solved in $O(n^2 d)$ time. Hence this *seems* unrelated to ETH or SETH. But things are not always how they seem.

Conjecture 7.11. *The* Orthogonal Vectors Conjecture *(OVC) is that, for all $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ algorithm for* Orthogonal Vector Problem.

Williams [Wil05] showed the following:

Theorem 7.12. *SETH implies OVC.*

We discuss consequences of OVC in Chapter 17.

Definition 7.13. A lattice $\mathcal{L}$ in $\mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$.

Closet Vector Problem (ClVecProb)
Instance: A lattice L (specified through a basis) together with a target vector $\vec{v} \in \mathbb{R}^n$.
Question: A vector $\vec{u} \in L$ that is closest to $\vec{v}$.
Note: What do we mean by closest? Let $1 \le p \le \infty$. Then the ***p*-norm** of a vector $(x_1, \ldots, x_n)$ is

- $(|x_1|^p + \cdots + |x_n|^p)^{1/p}$ if $p \neq \infty$.
- $\max_{1 \le i \le n} |x_i|$ if $p = \infty$.

We will discuss a variety of values for p.

The common case is $p = 2$ which is the standard Euclidean distance. To indicate that the p-norm is being used, the notation ClVecProb$_p$ is the convention. Aggarwal et al. [ABGS21] showed the following:

Theorem 7.14. *Assuming SETH. For all $\varepsilon > 0$, for all $p \notin 2\mathbb{Z}$, there is no $2^{(1-\varepsilon)n}$ algorithm for ClVecProb$_p$.*

It is unfortunate that they do not have the result for $p = 2$ which is the case of most interest. They comment that the gadgets they use do not exist for even values of p.

Shortest Vector Problem (ShVecProb)
Instance: A lattice $\mathcal{L}$ (specified through a basis).
Question: Output a vector $\vec{v} \in \mathcal{L}$ of minimal norm.
Note: A norm p will be specified as well.
Note: This problem is the basis for many post-quantum crypto systems. Thats just a fancy way of saying crypto systems that do not depend on number-theoretic hardness assumptions.

The common case is $p = 2$ which is the standard Euclidean distance. To indicate that the p-norm is being used, the notation ClVecProb$_p$ is the convention. Aggarwal et al. [ABGS21] showed the following:

Theorem 7.15. *Assuming SETH. Let $\varepsilon > 0$ and let $p \notin 2\mathbb{Z}$. there is no $2^{(1-\varepsilon)n}$ algorithm for ClVecProb$_p$ or ShVecProb$_p$.*

It is unfortunate that they do not have the result for $p = 2$ which is the case of most interest.
Huck Bennet et al. [BGSD20] have a survey of open problems on the complexity of lattice problems. We mention one.

Theorem 7.16. *Assume SETH. There is no $O(2^{0.99n})$ time algorithm for ShVecProb.*

Based on their names, one would think that SETH $\Rightarrow$ ETH. While this is true, it is not obvious. The interested reader should see the paper by Impagliazzo et al. [IPZ01]. Does ETH $\Rightarrow$ SETH? This is open.

7.5 Further Results

Graph Homomorphism
Instance: Graphs G, H.
Question: Is there a homomorphism from G to H. (A ***homomorphism*** from $G = (V, E)$ and $H = (V', H')$ is a function $f : V \to V'$ such that if $(u, v) \in E$ then $(f(u), f(v)) \in E'$.
Note: If H is K_c then this question is asking whether G is c-colorable. Hence Graph Homomorphism is NP-complete.

Subgraph Isomorphism
Instance: Graphs G, H.
Question: Is there H isomorphic to some subgraph of G?
Note: If $H = K_c$ then this question is asking whether G has a clique of size k. Hence Subgraph Isomorphism is NP-complete.

Cygan et al. [CFG+17] showed that, assuming ETH, the following hold:

1. Graph Homomorphism can't be done in $|V(H)^{O(|V(G)|)}$ time.
2. There is no $|V(H)^{O(|V(G)|)}$ algorithm for Subgraph Isomorphism.

We will now look at a graph coloring problem.

Definition 7.17. An $(a : b)$ coloring of a graph is a coloring where you assign b colors to each vertex out of a total a colors, so that adjacent vertices have disjoint sets of colors. One may also say informally that G is $\frac{a}{b}$-colorable. This number is called the ***Fractional Chromatic Number***.

Fractional Chom Numb(a, b)
Instance: A graph G and a, b.
Question: Is G $(a : b)$-colorable?
Note: If $b = 1$ this is asking whether G is b-colorable. Hence this problem is NP-completed

The study of fractional chromatic number was motivated as follows.

- Appel et. al [AH77a, AHK77] showed that every planar graph is 4-colorable. Their proof made extensive use of a computer program to check a massive amount of cases. Robertson et al. [RSST97] had a simpler proof, though it still needed a computer program. In short, the proof is not *human readable.*
- By contrast, the proof that every planar graph is 5-colorable is easy to follow and is clearly human-readable.
- Fractional chromatic number was defined with the goal of finding human-readable proofs that every planar graph is c-colorable for some values of $c < 5$. Cranston & Rabern [CR18] showed that every planar graph is 4.5-colorable. It is open to lower that.

The following lower bounds are known.

Theorem 7.18.

1. *(Grötschel et al. [GLS81]) For all $c > 2$, $c \in \mathbb{Q}$, determining whether G is c-colorable is NP-complete.*

2. *(Bonamy et al. [BKP+19]) Assume ETH. Fix $a, b \in \mathbb{N}$ such that $b \leq a$. For any computable function f, FRACTIONAL CHOM NUMB(a, b) cannot be solved in time $O(f(b)2^{o(\log b)n})$.*

7.6 We Will Return to ETH and SETH Later

The assumptions ETH and SETH are used to show several results in this book:

1. Lower bounds in Parameterized Complexity: Sections 8.1 and 8.15.

2. Lower bounds on the approximate shortest vector problem: Section 10.8.3.

3. Lower bounds on the approximate nearest neighbor problem: Section 10.8.4.

4. Lower bounds for dSUM: Theorem 17.34.

5. Quadratic lower bounds: Section 18.8.

6. Lower bounds on the approximate Nash Equilibrium problem: Theorem 22.33.

Chapter 8

Parameterized Complexity

8.1 Introduction

In this chapter we will present several results about fixed parameter tractability and parameterized complexity. For more on these fields there are several good books. We mention three: Downey & Fellows [DF99], Niedermeier [Nie06], and Flum & Grohe [FG06]. and [CFK+15].

Recall that the following two problems are NP-complete

$$\textsc{Vertex Cover} = \{(G,k) \mid G \text{ has a Vertex Cover of size } k\ \}$$

$$\textsc{Clique} = \{(G,k) \mid G \text{ has a Clique of size } k\ \}.$$

Let $k \in \mathbb{N}$. Let

$$\textsc{Vertex Cover}_k = \{G \mid G \text{ has a Vertex Cover of size } k\ \}$$

$$\textsc{Clique}_k = \{G \mid G \text{ has a Clique of size } k\ \}.$$

Both of these problems are in time $O(n^k)$ by a brute force algorithm. Hence they are both in P. Note that the degree of the polynomial depends on k. Is there a polynomial time algorithm for either $\textsc{Vertex Cover}_k$ or $\textsc{Clique}_k$ such that the degree of the polynomial does not depend on k? For $\textsc{Vertex Cover}_k$ the answer is yes.

Mehlhorn [Meh84] showed the following.

Theorem 8.1. *$\textsc{Vertex Cover}_k$ can be solved in time $O(2^k n)$.*

Proof. Algorithm for $\textsc{Vertex Cover}_k$:

1. Input $G = (V, E)$.

2. Form a tree as follows. The root has $(G, e_1, \emptyset)$ where $e_1 = (u, v)$ is an edge (chosen arbitrarily) and the set $X = \emptyset$ which we will add to. It later on in the tree it will hopefully be a vertex cover of size $\le k$. Note that either u or v is in the Vertex Cover. We will branch two ways depending on if we choose u or v. The two children of the root are as follows:

   (a) The right child is $(G - \{u\}, e_2, \{u\})$ where e_2 is some edge of $G - \{u\}$, together with $X \cup \{u\}$.

   (b) The left child is $(G - \{v\}, e_3, \{v\})$ where e_3 is some edge of $G - \{v\}$, together with $X \cup \{v\}$.

3. Keep doing this until either the tree is of height k or the graph attached to the node of the tree has no edges. In the latter case you have a vertex cover of size k. In the former case check whether if one of the leaves has the empty graph. If so then there is a vertex cover of size $\le k$. If not, then there is not.

There are $O(2^k)$ nodes on the tree. Forming each node takes $O(n)$ time. Hence the algorithm runs in $O(2^k n)$ time.

□

Are better results known or likely? The following two theorems give a yes or a no depending on your definitions of "better" and "likely".

1. Chen et al. [CKX10] showed VERTEX COVER$_k$ is in time $O(kn + 1.2738^k)$.

2. Cai & Juedes [CJ03] showed that if VERTEX COVER$_k$ is in time $2^{o(k)}n^L$ for some L, then 3SAT is in time $2^{o(n)}$, which violates the ETH. So, for example, it is unlikely that there is an algorithm for VERTEX COVER$_k$ that runs in time $O(2^{k/logk}n^{100})$. We will prove this in Theorem 8.44.

What if we restrict to planar graphs?

1. Demaine et al. [DFHT05] showed that Planar k-Vertex Cover can be done in $2^{O(\sqrt{k})}n^{O(1)}$ time.

2. Cai & Juedes [CJ03] showed that if VERTEX COVER$_k$, restricted to planar graphs, is in time $2^{O()\sqrt{k})}n^L$ for some L, then 3SAT is in time $2^{o(n)}$, which violates the ETH. We will prove this in Theorem 8.44.

We will later see that there are reasons to think any polynomial time algorithm for CLIQUE$_k$ will have degree that depends on k.

8.2 Kernelization

By Theorem 8.1 VERTEX COVER$_k$ is in time $O(2^k n)$. We will show a better algorithm which will be an example of kernelization. The key to the algorithm is that if there is a vertex of degree $k+1$ then it must be in the vertex cover, else all $k+1$ of its neighbors has to be in the vertex cover.

The following theorem is (correctly) attributed to S. Buss, though it is not published.

Theorem 8.2. *VERTEX COVER$_k$ can be solved in time $O(kn + 2^{2k^2})$.*

Proof.

1. Input $G = (V, E)$ and k.

2. Let $U = \emptyset$. We will be adding vertices of the vertex cover to U. Let $L = k$. We are currently looking for a vertex cover of size $L = k$. At each step we will be looking for a vertex cover of size L. We will always have $|U| + L = k$.

3. If there is a vertex v of degree $\ge L+1$ then (1) $U \leftarrow U \cup \{v\}$, (2) $L \leftarrow L - 1$, (3) $G \leftarrow G - \{v\}$. Repeat until the answer is NO. There will be at most k iterations, each taking at most $O(n)$ steps, so $O(kn)$ steps.

4. (This is a comment, not part of the algorithm.) Every vertex of G has degree $\le L$. Hence if there is a vertex cover of size L, with each vertex covering $\le L$ edges, there has to be at most L^2 edges, so $\le 2L^2$ vertices. We will assume the worse case which is that initially the graph has all vertices of degree $\le k$, so $L = k$.

5. If G has more than k^2 edges then output NO and stop.

6. (If you got here then G has $\le k^2$ edges.) Do the algorithm from Theorem 8.1 to try to find a vertex cover of size k. If you find one, then output YES. This step takes 2^{2k^2} time.

From the comments in the algorithm it works and it takes time $O(kn + 2^{2k^2})$. □

The algorithm in Theorem 8.2 took a Vertex Cover problem of size n with parameter k and reduced it to a Vertex Cover problem of size 2^{2k^2}. This is a common technique so it has a name.

Definition 8.3.

1. ***Kernelization*** is the procedure of reducing a problem $X\{n, k\}$ down to a problem $X\{f(k), g(k)\}$ using only $n^{O(1)}$ preprocessing time.

2. The new problem of size $f(k)$ is ***the kernel***. Hence $f(k)$ is the size of the kernel.

3. A problem ***has a kernel*** it if can be kernelized.

In general $f(k)$ can be exponential but a good kernel should be polynomial or even linear in k.

In Theorem 8.2 the kernel was of size $2k^2 = O(k^2)$. Soleimanfallah & Yeo [SY11] achieve a kernel of sizes $2k - c$ vertices for any constant c. Lampis [Lam11] achieve a kernel of size $2k - c \log k$ for any constant c which is the best currently known. How much smaller can the kernel be? We state two theorems about this. The first is an easy exercise for the reader. The second is a difficult result of Dell & Melkebeek [DvM14].

Theorem 8.4.

1. *If there is a kernelization of* Vertex Cover$_k$ *with kernel of size* $O(\log k)$ *then* $P = NP$*.*

2. *If there exists* ε *and a kernelization of* Vertex Cover$_k$ *with kernel of size* $O(k^{2-\varepsilon})$ *then coNP* $\subseteq$ *NP/poly (which is thought to be unlikely and implies that* $\Sigma_2 = \Pi_2$*).*

From the definition of kernelization, having a kernel implies a running time of $n^{O(1)} + h(k)$ for some h. But in fact we know something much stronger.

Theorem 8.5. *Let A be a parameterized problem. A has a kernel if and only for all k,* A_k *is in P with a polynomial of degree that does not depend on k (this will soon be called* ***Fixed Parameter Tractable****).*

8.3 Fixed Parameter Tractable

For Vertex Cover$_k$ there is a polynomial-time algorithm where the degree of the polynomial does not depend on k. We will define this notion formally. We will look at (usually) NP-complete problems that have a parameter (like k for Vertex Cover$_k$) and ask what happens if k is fixed.

Definition 8.6. Let A be a problem with a natural parameter k and let A_k be the problem A with that parameter fixed at k. A is ***Fixed Parameter Tractable (FPT)*** if there is a function f such that, for all k, A_k can be solved in time $f(k)n^{O(1)}$. Note that (1) $f(k)$ might be quite large, and (2) the degree of the polynomial $f(k)n^{O(1)}$ does not depend on k.

8.4 Parameterized Reductions

As usual, we'll show problems are hard via reductions. In general, we have a parameterized problem A, a parameterized problem B, and a map from (x, k) to (x', k'). This is going to look similar to Karp-style reductions, but with a few tweaks.

Definition 8.7. Let A and B be parameterized problems. A ***parameterized reduction*** of A to B maps an instance (x, k) of A to an instance (x', k') of B such that the following occurs.

1. x' only depends on x.
2. k' only depends on k.
3. The function that maps x to x' can be computed in polynomial time.
4. There exists a computable function $g : N \to N$ such that $k' \le g(k)$. Note that g might be quite large. In particular we are not requiring any time bound on the computation of g. This is because k will be a fixed constant parameter.
5. The reduction needs to be answer preserving: $(x, k) \in A$ if and only if $(x', k') \in B$.

Exercise 8.8. Prove the following.

1. if $B \in$ FPT and A is parameterized reducible to B then $A \in$ FPT.
2. If $A \notin$ FPT and A is parameterized reducible to B then $B \notin$ FPT. (This is just the contrapositive of Part 1. However, we will use this to show, under assumptions, that some problems are not FPT.)

Many reductions that we have seen before *are not* parameterized reductions. We give an example of a reduction that is *not* a parameterized reduction and then an example of one that is.

Recall that Independent Set is the Independent Set Problem, Vertex Cover is the Vertex Cover Problem, and Clique is the Clique Problem.

Example 8.9. (This is really a counterexample.) The standard reduction of Independent Set to Vertex Cover is to map (G, k) to $(G, n - k)$. Note that $n - k$ does not just depend on k, it also depends on n. Hence this is not a parameterized reduction.

Is there a parameterized reduction from INDEPENDENT SET to VERTEX COVER? Unlikely since (1) VERTEX COVER $\in$ FPT and (2) Chen et al. [CHKX06] showed that if CLIQUE is FPT then ETH is false.

Example 8.10. The standard reduction of INDEPENDENT SET to CLIQUE is to map (G, k) to $(\overline{G}, k)$ Here $k' = k$, so k' depends on k and is easily bounded by a function of k. Hence this is a parameterized reduction. Note that the reduction from INDEPENDENT SET to CLIQUE is very similar and is also a parameterized reduction.

Notation 8.11. Henceforth, in this chapter, "reduction" means "parameterized reduction".

8.5 The Complexity Class W[1]

In order to use reductions to show problems are not FPT we need to have some problems that we already think are not FPT. That is, we need an analog of SAT for showing problems not in P. Recall that we think SAT is not in P because (1) by the Cook-Levin Theorem, if SAT $\in$ P then P = NP, (2) people have been trying to get SAT (and many other problems in NP) into P without any success (some of the effort was before P was defined), and (3) gee, it just seems to require brute force.

Is there some problem that we are confident is not in FPT? There is, though frankly, the evidence is nowhere near as strong as for SAT not being in P. The evidence is similar to points (2) and (3) about SAT. Once we have the problem defined we will define a complexity class based on it.

NONDETERMINISTIC TURING MACHINE ACCEPTANCE (NONDET TM ACCEPTANCE)
Instance: A Nondeterministic Turing machine M and a number k. We will assume that there are $O(n)$ states, $O(n)$ alphabet size, and $O(n)$ choices at each step.
Question: If M is run on 0^n is there an accepting path of length k?

NONDET TM ACCEPTANCE can be solved in $O(n^k)$ steps. Fix k. There does not seem to be any way to do this problem in $f(k)n^{O(1)}$ for some (even quite large) f. In short, it looks like this problem is not in FPT. Indeed, we will assume that it is not.

We now define a complexity class based on NONDET TM ACCEPTANCE. It is called W[1] for reasons we will get into in Section 8.10.

Definition 8.12. Let A be a parameterized problem.

1. $A \in$ W[1] if there is a parameter reduction from A to NONDET TM ACCEPTANCE. (This will usually be easy to show.)

2. A is W[1]-hard if there is a parameter reduction from NONDET TM ACCEPTANCE to A.

3. A is W[1]-complete if it is in W[1] and is W[1]-hard.

Some sources, including Niedermeier's book [Nie06], use a different problem, WEIGHTED 2SAT, to define W[1]-complete. We will relook at Weighted SAT problems in Section 8.9 as a prelude to defining the W-hierarchy.

Weighted 2SAT
Instance: A 2CNF φ and $k \in \mathbb{N}$. (k is the parameter.)
Question: Is there a satisfying assignment of φ with exactly k variables set to true. This is called ***an assignment of weight k***.
Note: This is a terrible name for this problem since ***weighted SAT*** usually means that the clauses have weights and you want to maximize the sum of the weights of the satisfied clauses.

It turns out that Nondet TM Acceptance and Weighted 2SAT are reducible to each other by parameterized reductions so the two definitions do not conflict. We will use the definition based on Nondet TM Acceptance.

According to these slides:

http://fptschool.mimuw.edu.pl/slides/lec4.pdf

there are hundreds of W[1]-complete problems. This seems like an exaggeration; however, there are many W[1]-complete problems. If any of them are FPT they are all FPT. Many are problems that have been worked on a lot. This is good evidence that (1) none of them are FPT, and (2) hence Nondet TM Acceptance is not FPT.

8.6 Independent Set and Clique and Variants are W[1]-Complete

Definition 8.13. Let N be a NTM over alphabet Σ and state set Q. Let x be an input to it and let $k \in \mathbb{N}$. Run $N(x)$ for k steps. A ***configuration*** is an element of $\Sigma^*(Q \times \Sigma)\Sigma^*$ that represents the content of the tape, the position of the head (on the $Q \times \Sigma$ spot), and the state of the machine. Note that since the machine has run for $\leq k$ steps we can assume that all configurations are of length $O(k)$.

Theorem 8.14.

1. *Independent Set is W[1]-complete.*
2. *Clique is W[1]-complete. (This is an easy reduction from Independent Set hence we omit it.)*

Proof.
We show Independent Set is parameter reducible to Nondet TM Acceptance and Nondet TM Acceptance is parameter reducible to Independent Set.
Independent Set reducible to Nondet TM Acceptance: Given (G, k) set up an NTM that has G build into it and guesses k vertices (that's k moves) and then check that each pair is not an edge (that's k^2 moves). The length of the path in the NTM is $O(k^2)$. Thus Independent Set $\in$ W[1].
And now the interesting part: we reduce Nondet TM Acceptance to Independent Set.

1. Input NTM N and $k \in \mathbb{N}$. Note that N has $O(n)$ states, $O(n)$ alphabet size, $O(n)$ nondeterministic branching. N has alphabet Σ and state set Q. The alphabet is Σ and the state set is Q.

2. k' will be k^2.

3. Imagine the possible sequence of k configurations of the NTM M going for k steps. This sequence will have k^2 cells which are labeled (i, j) for ith row, jth column.

4. For each cell we create a clique (which we describe soon). So far there are k^2 cliques.

5. Each clique is the same: the vertices are $\Sigma \cup \Sigma \times Q$ which are all possible entries in a cell. So there are $O(n^2)$ nodes in the clique.

6. The idea is to put in edges between nodes that cannot both be in the sequence of configurations.

   (a) Put an edge between two vertices in the same row that both think the head is there. Formally let $i \in \mathbb{N}$ and $j_1 \neq j_2$. Any vertex of the (i, j_1) clique labeled with an element of $\Sigma \times Q$ (so the head is at j_1) will have an edge to any element of the (i, j_2)-clique labeled with an element of $\Sigma \times Q$ (so the head is at j_2).

   (b) Put an edge between two vertices from different rows that clearly cannot occur. Formally let $i, j \in \mathbb{N}$ and look at a vertex of the (i, j)-clique that is labeled with an element of $\Sigma \times \mathbb{Q}$, say (a, p). If the Turing Machine was in state p and was looking at an a then the transition function constrains what happens next. Put an edge between that vertex and the vertices in the $(i+1, j-1)$-clique, $(i+1, j)$-clique, and $(i+1, j+1)$-clique that are incompatible with the (i, j) cell having (p, a).

7. Call this graph G. We show that G has an Independent Set of size k^2 if and only if there is an accepting path in N of length k.

If G has a Independent Set of size k^2 then it has to take one node from each of the (i, j)-cliques. These nodes code an accepting path of length k.

If there is an accepting path of length k then that tells you how to fill out all the cells, and hence gives an Independent Set of size k^2. □

Mathieson & Szeider [MS08] showed that Clique is still W[1] complete when restricted to regular graphs.

Theorem 8.15.

1. *Clique restricted to regular graphs is W[1]-complete.*

2. *Independent Set restricted to regular graphs is W[1]-complete. (This is a reduction from Clique restricted, so we omit the proof.)*

Proof. Since Clique $\in$ W[1], this restricted version of Clique is in W[1].

We show that Clique can be reduced to Clique restricted to regular graphs.

1. Input (G, k) Let Δ be the max degree of G. This is also called ***the degree of*** G.

2. We create a graph G' as follows. (See Figure 8.1 for a picture of what we are doing.)

(a) There are Δ copies of G. For v a vertex of G let $v_1, \ldots, v_\Delta$ be the analogs of G in the Δ graphs.

(b) Let v be a vertex of G. Let its degree be $\deg(v)$. Create $\Delta - \deg(v)$ new nodes. Connect all of the $v_1, \ldots, v_\Delta$ to all $\Delta - \deg(v)$ new nodes. Each v_i now has degree $\deg(v) + (\Delta - \deg(v)) = \Delta$. Each of the new vertices has degree Δ since it connects to one vertex from each copy of G_i.

In the case where $\Delta = 5$ and $d = 3$, the construction follows is illustrated in the image below.

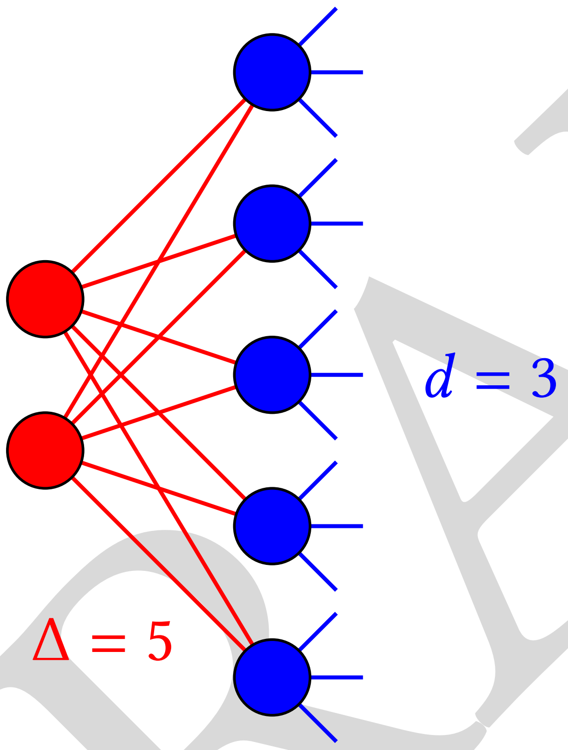


Figure 8.1: Reduction of Clique to Regular Clique

Note: It follows that Δ-regular k-Independent Set is W[1]-Hard since there is a simple reduction from Clique to Independent Set (analogous to the reduction in the other direction done in 3.1).

$\square$

Partial Vertex Cover
Instance: A graph G and two numbers $k, l \in \mathbb{N}$. We will regard this as a parameterized problem with parameter k.
Question: Is there a set of vertices of size k that covers l edges?

We will show this problem is W[1]-complete. Had we made l the parameter then this problem would be FPT. Note that if there is more than one choice for a parameter, which one is chosen matters!

Theorem 8.16. *Partial Vertex Cover is W[1]-hard.*

Proof. By Theorem 8.15, Independent Set on regular graphs is W[1]-complete. We show that Independent Set for regular graphs is reducible to Partial Vertex Cover.

1. Input (G, k). G is regular with degree Δ.
2. We will just use G, k, and $l = \Delta k$.

If G has an Independent Set of size k then those k vertices cover Δk edges since no two of the vertices are adjacent to the same edge.

If G has a set of k vertices that cover Δk edges then they must be independent since otherwise they would cover $< \Delta k$ edges. □

Note: The question "Is Partial Vertex Cover in W[1]?" seems to be open.

Multi Colored Clique (MClique) and Multicolored Independent Set (MIS)
Instance: A graph $G = (V, E)$ and a partition $V = V_1 \cup \cdots \cup V_k$. k will be the parameter.
Question: Is there a clique (independent set) with one vertex from each V_i? (It is called "multicolored" since we think of each V_i as a color; however, there is no restriction on this coloring.)

Pietrzak [Pie03] and independently Fellows et al. [FHRV09] showed the following.

Theorem 8.17.

1. *Multi Colored Clique is W[1]-complete.*
2. *Multicolored Independent Set is W[1]-complete (this is an easy reduction from Part 1, or a proof similar to Part 1, so we omit the proof).*

Proof. Multi Colored Clique is reducible to Clique (exercise) which is in W[1], hence Multi Colored Clique is in W[1].

We show that Clique is reducible to Multi Colored Clique.

1. Input $G = (V, E)$ and k.
2. Create a graph G' and a partition of the vertices as follows.
   (a) For every $x \in V$ there are k vertices $x_1, \ldots, x_k$.
   (b) V_i is all the set of all vertices with subscript i.
   (c) (x_i, y_j) is an edge if $i \neq j$ and $(x, y) \in E$. Note that within V_i there are no edges.
3. If G' has a clique with a vertex in each V_i, then G has a clique of size k. We leave the easy proof to the reader.

□

Exercise 8.18.

1. Show that Multi Colored Clique is reducible to Clique.

2. Show that the construction in the proof of Theorem 8.17 works.

Exercise 8.19. Prove there is a parameterized reduction from Dom Set to Set Cover.

For the next exercise we will consider the following variant of the Node-Weighted Steiner Tree.

Strongly Connected Steiner Subgraph Problem (SCSS)
Instance: A directed graph $G = (V, E)$, a set $U \subseteq V$, and $k \in \mathbb{N}$.
Question: Is there a strongly connected subgraph of G with $\le k$ vertices that contains every vertex of U? (A directed graph is strongly connected if there is a directed path between every pair of vertices.)

Exercise 8.20. Show that there is a parameterized reduction from Multi Colored Clique to SCSS. Note that this shows SCSS is W[1]-hard.

8.7 Parameterized Complexity of Flood-It on a Tree

Recall that in Section 5.4 we defined the game Flood-it on Graphs and showed it was NP-complete. In this section we define Flood-it on Trees and show that it is W[1]-complete.

The reader is asked to read Section 5.4 to remind themselves what Flood-it on Graphs and SCS are.

Pietrzak [Pie03] showed the following.

Theorem 8.21. *SCS with* $|\Sigma| = 2$ *is W[1]-complete.*

We omit the proof which is complicated.

Exercise 8.22. Prove or look up (in Fellows et al [FdSSPdS15]) the proof that RSCS is W[1]-hard.
Hint: Give a parameterized reduction of SCS to RSCS.

Definition 8.23. The game ***Flood-It (on a tree)*** is defined as follows.

1. The parameters are $n, c, g \in \mathbb{N}$.

2. The starting position is a tree T (with designated root r) on n vertices that is c-colored. There are no restrictions on the coloring. The goal is to, through a sequence of $\le g$ moves (to be defined soon), make the tree monochromatic.

3. A move consists of a player picking one of the c colors.

4. The move causes the following to happen. Take the mono-region that contains r. Change all of the vertices in it to color c. Note that if there are vertices colored c next to the vertices that changed to c then the mono-region containing v is now larger.

5. The game ends when all of the vertices are the same color. If this is accomplished within g moves then the player wins. Else he loses.

> Flood-it on Trees (Flood-it-T)
> *Instance:* Parameters $n, c, g \in \mathbb{N}$ and a tree T on n vertices that is c-colored.
> *Question:* Can the player win the game?

Fellows et al. [FRdSS18] surveys many different parameterized versions of Flood-it-T on different graphs. We present one result due to Fellows et al [FdSSPdS15].

Theorem 8.24. *Flood-it-T parameterized by the number of colors is W[1]-complete.*

Proof sketch:
We present a parameterized reduction RSCS $\le_p$ Flood-it-T.

1. Input is an alphabet Σ and a set $S \subseteq \Sigma^*$ such that no string has two of the same character in a row.

2. We create a colored tree T as follows.

   (a) The colors are $\Sigma \cup \{NC\}$ where NC is a new color.

   (b) The root is colored NC.

   (c) For each $w = w_1 \cdots w_n \in S$ form a sequence of n vertices starting at the root. They are colored $w_1, w_2, \ldots, w_n$.

In Exercise 8.25 we guide the reader through the proof that the reduction works. ▮

Exercise 8.25. Throughout this exercise we are referring to the reduction in Theorem 8.24.

1. Let y be a supersequence of all $x \in S$. Show that the sequence of colors in y is a strategy that makes T monochromatic.

2. Let z be a sequence of colors that is a strategy to make T monochromatic. Show that z is a supersequence of all $x \in S$.

3. Show that the reduction is an FPT-reduction.

8.8 Dominating Set

We remind the reader of the definition of Dom Set that was encountered in Section 2.7.

> Dom Set
> *Instance:* A graph $G = (V, E)$ and a number k. k is the parameter.
> *Question:* Is there a dominating set D of size k. (Every $v \in V$ is either in D or has an edge to some $u \in D$.)

It is not obvious (and it is likely false) that Dom Set is in W[1]. After guessing the k vertices for the dominating set D one then has to check whether *every* vertex is in D or adjacent to D.

This takes $O(kn)$ steps. Why is Dom Set (apparently) not in W[1] whereas Clique is? Because Clique is local: once you guess a set C for the clique you need only check whether every pair in C has an edge. We will later define W[2] and note (but not prove) that Dom Set is W[2]-complete. For now we show the following:

Theorem 8.26. *Dom Set is W[1]-hard.*

Proof. The proof that Dom Set is in W[1] is easy and omitted.

We reduce Multicolored Independent Set to Dom Set.

1. Input $G = (V, E)$ with V partitioned as $V_1 \cup \cdots \cup V_k$
2. Create a graph G' which will be formed by adding vertices and edges to G as follows
   (a) For each i (1) put an edge between every pair of vertices in V_i, (2) create two new vertices x_i, y_i, (3) for all $v \in V_i$ put in an edge from v to x_i and from v to y_i. Note that there is no edge between x_i and y_i. Hence any dominating set will have to have at least one vertex form each V_i. Any dominating set of size k will have exactly one vertex from each V_i.
   (b) For every edge $e = (u, v)$ where $u \in V_i$, $v \in V_j$, $i \neq j$, create a new node w_e. Put an edge between w_e and (1) every vertex of $V_i - \{u\}$ and (2) every vertex in $V_j - \{v\}$. (See Figure 8.2 for a picture of G')
3. We will show that G has an independent set with one vertex from each V_i if and only if G' has a dominating set of size k.

Assume G has an independent set I with one vertex from each V_i. We show that I is a dominating set for G'. Since there is one vertex in each V_i, $|I| = k$ and every vertex in $V_i \cup \{x_i, y_i\}$ is covered. Let $e = (u, v)$ and w_e be as in the construction. The set I cannot have both u and v, hence w_e is covered.

The proof that if G' has a dominating set of size k then G has an independent set with one from each V_i is similar.

□

Exercise 8.27. Show that Clique reduces to the following problems.

1. Set Cover.
2. Connected Dominating Set (the Domination Set has to be connected).
3. Independent Dominating Set (the Domination Set has to be independent).

What happens if we restrict G to be planar? The problem is known to be in FPT. See Alber et al. [ABF+02] for the most recent results and the history of prior results. In a nutshell: Planar Dominating set was in $O(c^k n)$ for some c, and Alber et al. [ABF+02] got it down to $O(c^{\sqrt{k}} n)$ for some c.

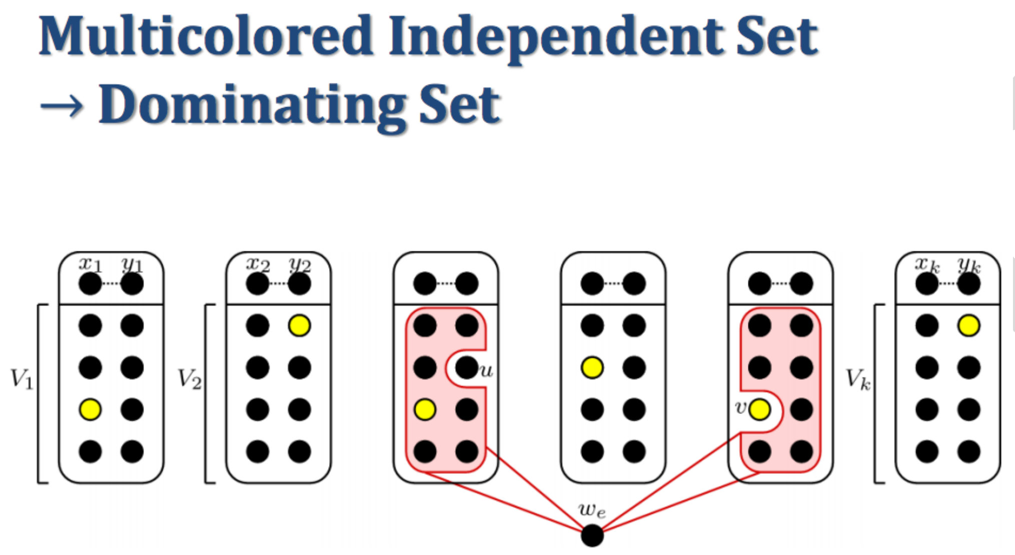


Figure 8.2: The Reduction of MIS to DOM

8.9 Circuit SAT and Weighted Circuit SAT

Definition 8.28.

1. A Boolean circuit is a circuit consisting of input gates, negation (~), AND (∧), OR (∨) and output gates.
2. An assignment of variables has ***weight k*** if k of the input are TRUE.

> Circuit SAT) and Weighted Circuit SAT
> *Instance:*A Boolean circuit C and (for Weighted Circuit SAT) $k \in \mathbb{N}$.
> *Question:* Is there an assignment on the inputs of C (for Weighted Circuit SAT the assignment has weight k) such that the output is T?

Theorem 8.29. *parameterized*

1. *Independent Set is reducible to Weighted Circuit SAT. Hence Weighted Circuit SAT is W[1]-hard.*
2. *Dom Set is reducible to Weighted Circuit SAT. We will later see that this means it is unlikely that Weighted Circuit SAT is in W[1].*

Proof.
1) We reduce Independent Set to Weighted Circuit SAT. See the circuit labeled ***Independent Set*** in Figure 8.3 for an example of the reduction.

1. Input $G = (V, E)$ and $k \in \mathbb{N}$.
2. We create a circuit C as follows.

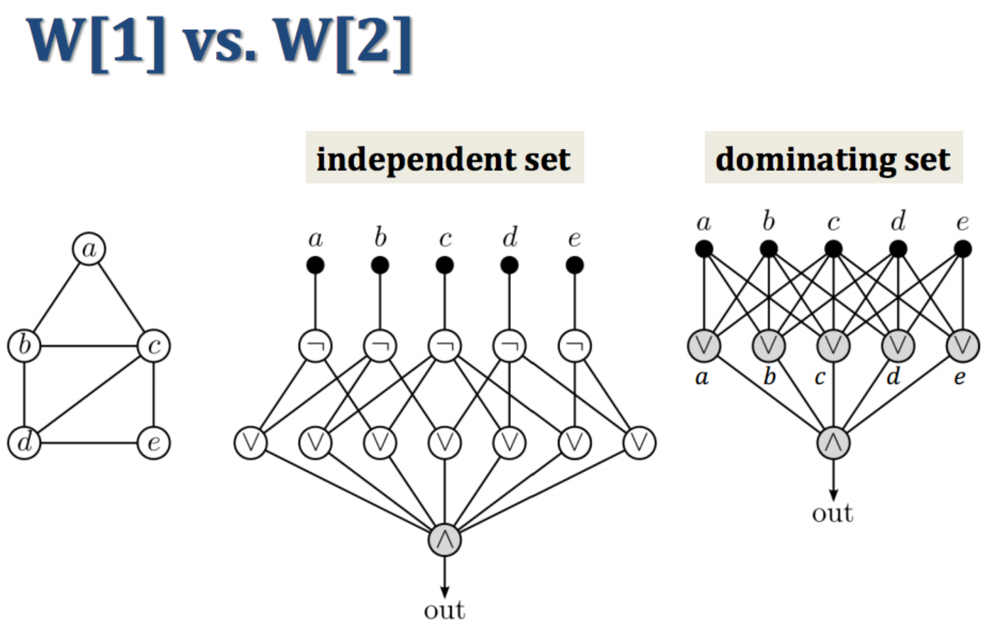


Figure 8.3: Circuits for INDEPENDENT SET and for DOM SET

(a) For each $v \in V$ we have an input.

(b) On the second level of the circuit we have the negation of all of the inputs.

(c) On the third level, for all $(u, v) \in E$, there is a gate that computes $\neg u \vee \neg v$.

(d) The fourth level is the AND of all of the gates on the third level.

3. It is easy to see that G has an INDEPENDENT SET of size k if and only if there is an input of weight k that satisfies C.

2) We reduce DOM SET to WEIGHTED CIRCUIT SAT. See the circuit labeled ***Dominating Set*** in Figure 8.3 for an example of the reduction.

1. Input $G = (V, E)$ and $k \in \mathbb{N}$.

2. We create a circuit C as follows.

   (a) For each $v \in V$ we have an input.

   (b) For every $v \in V$ we have a gate at the second level that computes the OR of (from the input level) v and all of the neighbors of v.

   (c) The third level is the AND of all of the gates on the second level.

3. It is easy to see that there is an G has an DOM SET of size k if and only if there is an input of weight k that satisfies C.

□

8.10 W-Hierarchy

Let us recap what we have.

1. The following problems are W[1]-complete and hence equivalent to each other: Nondet TM Acceptance, Independent Set, Clique, Independent Set restricted to regular graphs, Clique restricted to regular graphs, partial vertex cover, Multi Colored Clique, Multicolored Independent Set, SCS.

2. The following problems are W[1]-hard but do not seem to be in W[1]: Dom Set, Weighted Circuit SAT.

Why is (say) Clique in W[1] but Dom Set does not appear to be? As noted earlier Clique is local: once you have a potential clique of k vertices, to check they are a clique you need only look at hose k vertices. Dom Set is global: once you have a potential dominating set of k vertices, to check that they are a dominating set you need to look at all vertices.

With an eye towards formalizing the difference, we state the difference in terms of circuits. The different circuits we used for Independent Set and Dom Set (see Figure 8.3) are instructive. Both circuits take a set of vertices (as a bit vector and determine whether it satisfies the condition (Independent Set or Dom Set). But note the following:

1. In the Independent Set-circuit, all paths from an input to the output pass through only one gate that has a large number of inputs (in Figure 8.3 the gate has 7 inputs but more generally the number of edges between the k vertices, so at most $O(k^2)$).

2. In the Dom Set-circuit, all paths from an input to the output pass through two gates that have a large number of inputs (1) on the second level gate x has $deg(x) + 1$ inputs, (2) the output node will have n (the number of vertices) inputs.

We will measure the complexity of a problem by the maximum number of big gates it has on a path from the input to the output.

Definition 8.30.

1. A ***large gate*** is a gate that has more than two inputs. We may make it more than some constant number of inputs if we are discussing a family of circuits.

2. The ***depth of a circuit*** is the maximum length of a path from an input to the output.

3. The ***weft of a circuit*** is the maximum number of large gates on a path from an input to the output.

4. Let $C[t, d]$ be the set of all circuits having weft at most t and depth at most d. We will also use it to stand for the parameterized problem with parameter k where we seek a satisfying input of weight k.

We define the W-hierarchy. Downey & Fellows [DF99] first defined the W-hierarchy; however, J. Buss & Islam [BI06] is an excellent modern reference with simplified proofs.

Definition 8.31. Let A be a parameterized problem. Let $t \in \mathbb{N}$.

1. $A \in \mathrm{W}[t]$ if there is a constant d (which may be a function of k) and a reduction from A to $C[t, d]$. (This will usually be easy to show.)

2. A is ***W[t]-hard*** if, for all d, $C[t, d]$ reduces to A.

3. A is ***W[t]-complete*** if it is in $\mathrm{W}[t]$ and is $\mathrm{W}[t]$-hard.

4. $A \in \mathrm{W[SAT]}$ if A is reducible to SAT.

5. $A \in \mathrm{W[P]}$ if A is reducible to CIRCUIT SAT.

6. $A \in \mathrm{XP}$ if there exists functions f, g such that A_k can be solved in time $f(k)n^{g(k)}$.

The following are known.

1. FPT problems are in $C[0, O(f(k))]$ and thus in $\mathrm{W}[0]$.

2. We have defined $\mathrm{W}[1]$ in two ways, one using NONDET TM ACCEPTANCE and one using $C[1, d]$. These are equivalent.

3. One can also define $\mathrm{W}[i]$ just like the NONDET TM ACCEPTANCE definition of $\mathrm{W}[1]$ except we allow the Turing machine to have i tapes.

4. If $i \le j$ then $\mathrm{W}[i] \subseteq \mathrm{W}[j]$ (this is obvious). This hierarchy is believed to be proper.

5. INDEPENDENT SET is $\mathrm{W}[1]$-complete and DOM SET $\in$ $\mathrm{W}[2]$-complete. Hence if DOM SET is reducible to INDEPENDENT SET then $\mathrm{W}[1] = \mathrm{W}[2]$, which, as noted in the last point, is thought to be unlikely.

6. There are many problems that are $\mathrm{W}[1]$-complete. There are a few that are $\mathrm{W}[2]$-complete. There seem to be very few problems that are in higher levels or even seem to be in higher levels.

7. Downey & Fellows [DF99] (see also [Nie06]) showed the following problem is $\mathrm{W}[2]$-complete.

HITTINGSET
Instance: A hypergraph $H = (V, E)$ and a number $k \in \mathbb{N}$. (k will be the parameter.)
Question: Is there a set $V' \subseteq V$ such that, for all $e \in E$, there exists $v \in V' \cap E$? The set V' is called a ***Hitting Set***.

So we have

$$\mathrm{FPT} = \mathrm{W}[0] \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots \subseteq \mathrm{W[SAT]} \subseteq \mathrm{W[P]} \subseteq \mathrm{XP}.$$

There is a (somewhat) natural problem in $\mathrm{XP} - \mathrm{FPT}$. We present it and give some points about the proof.

Det TM Empty String Acceptance (DTMESA)
Instance: A Deterministic Turing Machine M, $n \in \mathbb{N}$ (in unary) and $k \in \mathbb{N}$ (the parameter).
Question: Does M accept the empty string in time $\leq n^k$? Note that Det TM Empty String Acceptance $\in$ XP.

Downey & Fellows [DF99] proved the following:

Theorem 8.32.

1. *XP − FPT ≠ ∅. This is proven by a diagonalization and hence does not yield a natural set in XP − FPT. Let D ∈ XP − FPT be this set.*
2. *Det TM Empty String Acceptance is XP complete under parameterized reductions. Hence D reduces to Det TM Empty String Acceptance.*
3. *Det TM Empty String Acceptance ∈ XP − FPT. This follows from items 1 and 2.*

Very few other XP-complete sets are known.

Note: Downey et al. [DFR98] have defined an alternative (but equivalent) definition of the W hierarchy that uses descriptive complexity theory.

For the next exercise we will consider the following variant of Dom Set.

Connected Dominating Set
Instance: A graph G and a number k.
Question: Is there a dominating set X of size $\leq k$ such that X induces a connected graph?

Exercise 8.33. 1. Prove there is a parameterized reduction from Dom Set to Connected Dominating Set.

2. Prove Connected Dominating Set $\in$ W[2] by creating an instance of Weighted Circuit SAT with weft two for it.

3. Prove that Connected Dominating Set is W[2]-complete.

8.11 Perfect Codes: A Problem With an Interesting History

Perfect Codes
Instance: A graph $G = (V, E)$ and $k \in \mathbb{N}$.
Question: Is there a $V' \subseteq V$, $|V'| = k$, such that, each vertex $v \in V$, there is exactly on $v' \in V'$ that is either v or adjacent to v.

This problem has an interesting history. Downey & Fellows [DF99] showed the following:

Theorem 8.34.

1. *(Downey & Fellows [DF99]) Independent Set is reducible to Perfect Codes, hence Perfect Codes is W[1]-hard.*

2. *(Downey & Fellows [DF99]) Perfect Codes $\in$ W[2].*

3. *(Cesati [Ces03]) Perfect Codes is reducible to Nondet TM Acceptance and hence Perfect Codes $\in$ W[1]. Therefore Perfect Codes is W[1]-complete.*

Downey & Fellows conjectured that Perfect Codes is intermediary: in W[2] − W[1] but not W[2]-complete. Hence it was a surprise when Cesati showed Perfect Codes is W[1]-complete. It was interesting that Cesati showed Perfect Codes $\in$ W[1] by a reduction.

Exercise 8.35. This exercise will show why the problem is called "Perfect Code". If x, y are strings of the same length then $d(x, y)$ is the number of bits they differ on.

Let $G_n = (V, E)$ be the graph with $V = \{0, 1\}^n$ and $E = \{(x, y) \mid d(x, y) = 1\}$.

1. Show that G_3 has a hitting set of size 2.

2. Show that G_4 has a hitting set of size 8.

3. Find a function $f(n) < 2^n$ such that, for all n, G_n has a hitting set of size $f(n)$.

4. Alice wants to send $x \in \{0, 1\}^n$ to Bob. She sends it over a line which sometimes flips 1 bit but never more. Restrict the strings Alice may send so that Bob can tell whether an error occurred, and if so, correct it.

5. Read the literature on error correcting code. (Warning: It is vast.)

8.12 Lower Bounds on Approximations via W[1]-Hardness

We define Polynomial Time Approximation Schemes (PTAS) and Efficient Polynomial Time Approximation Schemes (EPTAS). We then use the assumption W[1] ≠ FPT to show that some problems are unlikely to have an EPTAS. In Chapters 9 and 10 we will use the assumption P ≠ NP to show that some problems are unlikely to have a PTAS.

Definition 8.36.

1. Let A be a min-problem (e.g., given a graph return the size of the smallest vertex cover). A ***polynomial time approximation scheme*** (PTAS) for A is an algorithm that takes as input (x, ε) (x an instance of A and $\varepsilon > 0$) and outputs a solution that is $\le (1 + \varepsilon)A(x)$. The only runtime constraint is that if we fix ε, the PTAS must run in time polynomial in the size of the remaining input. Note that this allows very bad runtimes in terms of ε. For example, a PTAS can run in time $n^{2^{2^{\varepsilon^{-1}}}}$ because for any given value of ε this is a polynomial runtime.

2. Let A be a max-problem (e.g., given a graph return the size of the largest clique). A ***polynomial time approximation scheme (PTAS)*** for A is an algorithm that takes as input (x, ε) (x an instance of A and $\varepsilon > 0$) and outputs a solution that is $\ge (1 - \varepsilon)A(x)$. We have the same runtime constraint as in part 1.

Taking a hint from our study of FPT we define the following.

Definition 8.37. Let A be a min-problem (a similar definition can be made for max-problems). An ***efficient polynomial time approximation scheme*** (EPTAS) for A is an algorithm that takes as input (x, ε) (x an instance of A and $\varepsilon > 0$) and outputs a solution that is $\leq (1 + \varepsilon)A(x)$. The only runtime constraint is that there is a computable function f so that the algorithm runs in time $f(\frac{1}{\varepsilon})n^{O(1)}$. Note that this allows very bad runtimes in terms of ε. For example, a EPTAS can run in time $2^{2^{\varepsilon^{-1}}} n^2$ because for any given value of ε this is of the right form. One caveat: we will have a problem where sometimes there is no solution at all. In this case, an EPTAS will just output "no solution".

The following problem comes up in the human genome project. We use the formulation of Deng et al. [DLL+02].

Definition 8.38. If x, y are string of the same length then $d(x, y)$ is the number of places they differ on.

Dist Substring Sel)
Instance: A set $B = \{b_1, \ldots, b_{n_1}\}$ of bad strings, a set $G = \{g_1, \ldots, g_{n_2}\}$ of good strings, and $L, k_b, k_g \in \mathbb{N}$. All of the strings in B are of length $\geq L$ and all the strings in G are of length L. All of the strings are over the same alphabet Σ which we will assume is a constant. We will take $n = \max\{n_1, n_2\}$ and we assume that the strings in B are of length $\leq 2L$, so the input size is $O(nL)$.
Question: (Set Version) Is there a string s such that the following occur:

1. For every $1 \leq i \leq n_1$ there is a substring t_i of b_i such that $d(s, t_i) \leq k_b$. (So s is close to a substring of every bad string.)
2. For every $1 \leq i \leq n_2$, $d(s, g_i) \geq k_g$. (So s is far from every good string.)

Question: (Function version) If no such string s exists then output NO; however if a string exists, output one of them.
Note: Dist is for Distinguished and Sel is for Selection.
Note: The terminology "good" and "bad" has its motivation in the application to designing genetic markers to distinguish the sequences of harmful germs, for which it is good for the markers to bind, from the human sequences, for which it is bad for the markers to bind. This terminology may have its origin in Lanctôt et al. [LLM+03].

Deng et al. [DLL+02] proved the following.

Theorem 8.39. *There is a PTAS for Dist Substring Sel. In particular, there is a function f and an algorithm A such that on an instance of Dist Substring Sel and an ε, A runs in time $(nL)^{f(1/\varepsilon)}$ and outputs, if it exists, an s such that the following holds:*

1. *For every $1 \leq i \leq n_1$ there is a length-L substring t_i of b_i such that $d(s, t_i) \leq (1 + \varepsilon)k_b$.*
2. *For every $1 \leq i \leq n_2$, $d(s, g_i) \geq (1 - \varepsilon)k_g$.*

We have used the term Dist Substring Sel for both the function version and set version of the problem. We will now need to distinguish them.

Notation 8.40.

1. Dist Substring Sel-fun is the function version of Dist Substring Sel.
2. Dist Substring Sel-set-k_b, k_g is the parameterized version with k_b, k_g fixed.

We will now link approximating Dist Substring Sel-fun to Dist Substring Sel-set-k_b, k_g having an FPT. Our proof is essentially due to Cesati & Trevisan [CT97] who proved a general theorem relating EPTAS's and FPT (which you will prove in the exercises after the next theorem).

Theorem 8.41. *If Dist Substring Sel-fun has an EPTAS then the Dist Substring Sel-set-k_b, k_g is in FPT.*

Proof. Assume there is an EPTAS for Dist Substring Sel-fun. It runs in time $f(1/\varepsilon)(nL)^{O(1)}$.

The following is an FPT algorithm for Dist Substring Sel-set-k_b, k_g. Fix k_b and k_g and let $k = \max\{k_b, k_g\}$.

1. Input an instance I of Dist Substring Sel-set-k_b, k_g. We take I to be the the entire instance including k_b, k_g.
2. Run the EPTAS on I with $\varepsilon = \frac{1}{2k}$. Note that this takes time $f(2k)(nL)^{O(1)}$.
3. (This is commentary and is not part of the algorithm.)

   If the EPTAS outputs a string s then the following holds:

   (a) For every $1 \le i \le n_1$ there is a length-L substring t_i of b_i such that $d(s, t_i) \le (1+\frac{1}{2k})k_b = k_b + \frac{k_b}{2k}$. Since $d(s, t_i) \in \mathbb{N}$ we have $d(s, t_i) \le k_b$.

   (b) For every $1 \le i \le n_2$, $d(s, g_i) \ge (1 - \frac{1}{2k})k_g = k_g - \frac{k_g}{2k}$. Since $d(s, t_i) \in \mathbb{N}$ we have $d(s, g_i) \ge k_g$.

   (c) Because of the above two points, $I \in$ Dist Substring Sel-set-k_b, k_g

   If the EPTAS outputs "there is no such string" then clearly $I \notin$ Dist Substring Sel-set-k_b, k_g.

4. If the EPTAS returns a string then output YES $I \in$ Dist Substring Sel-set-k_b, k_g.

   If the EPTAS does not return a string then output NO, $I \notin$ Dist Substring Sel-set-k_b, k_g.

From the comments made in the algorithm we have that it is correct and works in time $f(k)(nL)^{O(1)}$). Hence Dist Substring Sel-set-k_b, k_g is FPT. □

The above theorem is only interesting if Dist Substring Sel-set-k_b, k_g is W[1]-hard. It is! Gramm et al. [GGN06] proved the following which we will not prove.

Theorem 8.42.

1. *Dist Substring Sel-set-k_b, k_g is W[1]-hard.*

2. *If there is an EPTAS for* Dist Substring Sel *then W[1] = FPT. (This follows from Part 1.)*

Exercise 8.43.

1. Show that if there is an EPTAS for Dom Set then FPT = W[1].

2. Formulate a general theorem that links EPTAS's and FPT.

The general theorem that you will prove in exercise 8.43 and that Cesati & Trevisan [CT97] proved has very few applications. See the paper of Cesati &Trevisan for one more.

8.13 Consequence of ETH for Parameterized Complexity

In Chapter 8 we showed that, under suitable hypothesis, some problems were unlikely to be FPT. We now show that, assuming ETH, we can obtain better lower bounds for both problems within FPT and outside of FPT.

8.13.1 ETH Implies $2^{\Omega(k)}n$ Lower Bounds

Recall from Theorem 8.1 that Vertex Cover$_k$ is in time $O(2^k n)$. Can this be improved to, say, $2^{k^{0.9}}n$? $2^{k^{0.9}}n^L$ for some L? What about other parameterized problems? Cai & Juedes [CJ03] showed the such an algorithm would violate the ETH:

Theorem 8.44. *Assume ETH. Let $L \in \mathbb{N}$. Then the k-parameterized version of vertex cover, dominating set, clique, and directed Hamiltonian cycle require $2^{\Omega(k)}n^L$ to solve. For vertex cover this bound is tight.*

Proof. We prove this for Vertex Cover. The rest are similar.

By Theorem 7.7, assuming ETH, Vertex Cover requires $2^{\Omega(n)}$ time. If Vertex Cover$_k$ could be done in $2^{o(k)}n^L$ time then, since $k \le n$, this would yield a $2^{o(n)}n^L$ algorithm for Vertex Cover. This violates the lower bound of $2^{\Omega(n)}$.

Chen et al. [CKX10] showed Vertex Cover$_k$ is in time $O(kn+1.28^k)$. Hence the lower bound is tight. □

Exercise 8.45. Prove the rest of Theorem 8.44.

What happens if we restrict the problems mentioned in Theorem 8.44 to planar graphs? We omit Clique from this discussion since Clique for planar graphs is in P. For the rest we have the following result:

Exercise 8.46. Assume ETH. Show that the following parameterized problems restricted to planar graphs vertex cover, dominating set, directed Hamiltonian cycle, require $2^{\Omega(\sqrt{k})}n$ time.

We note when the lower bounds of Exercise 8.46 matching the known upper bounds:

- Alber et al. [ABF+02] show that Planar k-Dominating set can be done in $2^{O(\sqrt{k})}n^{O(1)}$ time.

- Demaine et al. [DFHT05] show that Planar k-Vertex Cover can be done in $2^{O(\sqrt{k})}n^{O(1)}$ time.

There are other planar FPT problems for which, assuming ETH, we have matching upper and lower bounds.

8.13.2 ETH Implies $f(k)n^{\Omega(k)}$ Lower Bounds

Recall that, by Theorem 8.14, assuming FPT $\neq$ *Wone*, Clique$_k$ and Independent Set$_k$ are not FPT. More precisely, there is no function f such that Clique$_k$ can be done in time $f(k)n^{O(1)}$. What about (say) $f(k)n^{\sqrt{k}}$? What about other parameterized problems?

Theorem 8.47. *Assume ETH. Let $f(k)$ be any computable function. Then Clique$_k$ and Independent Set$_k$ require $f(k)n^{\Omega(k)}$ time.*

Proof. By Theorem 7.7, assuming ETH, 3COL requires $2^{\Omega(n)}$ time. We give a reduction from 3COL to Clique$_k$. We will need to carefully keep track of how long the reduction takes.

1. Input $G = (V, E)$, a graph. We will assume k divides n for notational convenience.
2. Split V into k groups $V_1, \dots, V_k$ of roughly $\frac{n}{k}$ vertices each.
3. For each $1 \le i \le k$ find all valid 3-colorings of V_i. For each one we have a new vertex. There are at most $3^{n/k}$ vertices per i, so at most $k3^{n/k}$ new vertices. These new vertices will be our vertex set V'. Note that this step takes $O(k3^{n/k})$ time.
4. If two vertices in V' correspond to the same V_i, there is no edge between them.
5. Let x and y be vertices in V' such that x comes from V_i, y comes from V_j, and $i \neq j$. If the coloring $x \cup y$ of $V_i \cup V_j$ is valid then put an edge between x and y, otherwise, do not.
6. Let E' be the set of edges, so $G' = (V', E')$ is the new graph.

It is easy to see that G has a 3-coloring if and only if G' has a k-clique. We need to analyze the time of the reduction carefully to get our result.

Assume Clique$_k$ is solvable in $f(k)n^{o(k)}$ time. Then there exists a monotone increasing and unbounded s such that Clique$_k$ is solvable in time $f(k)n^{k/s(k)}$ where s is monotone increasing and unbounded.

Set k as large as possible such that $f(k) \le n$ and $k^{k/s(k)} \le n$. Let $k = k(n)$ be the minimum of the 2 inverses of the above functions. The running time of our 3-coloring algorithm is

$$f(k)((k3^{n/k})^{k/s(k)}) \le nk^{k/s(k)}3^{n/s(k)} \le n^2 3^{n/s(k(n))} \le 2^{o(n)}$$

which contradicts the lower bound on 3COL that comes from ETH. □

We can now use Clique$_k$ and a certain kind of reduction to get, assuming ETH, $f(k)n^{\Omega(k)}$ lower bounds.

Definition 8.48. Let A and B be parameterized problems. A ***k*-linear FPT reduction from *A* to *B*** is an FPT reduction such that when (x, k) is mapped to (y, k'), $k' = O(k)$.

Exercise 8.49. Assume ETH. Let f be a computable function.

1. Let A_k be a parameterized problem with parameter k. Assume that Clique$_k$ is reducible to A_k with a k-linear reduction (so A_k is *Wone*-hard). Then A_k requires $f(k)n^{\Omega(k)}$ time.
2. Assume ETH. Let f be any computable function. Then the following problems have run-times $\ge f(k)n^{\Omega(k)}$: Multicolored Clique/Independent Set, Dominating Set, Set Cover, and Partial Vertex Cover.

8.14 Grid Tiling

All of the results in this section are from Cygan et al. [CFK+15].

We introduce two parameterized Grid Tiling problems and obtain lower bounds on them. This problems are contrived; however, by reductions, we will use them to get lower bounds on other parameterized problems that we do care about.

8.14.1 Grid Tiling

Definition 8.50. Let G be a grid.

1. Two cells are ***up-down neighbors*** if one is directly above the other.
2. Two cells are ***left-right neighbors*** if one is directly to the right of the other.

k-Grid Tiling Problem (Grid$_k$)
Instance: A $k \times k$ grid, an $n \in \mathbb{N}$, and each cell $S(i,j)$ in the grid has a subset of $\{1,\ldots,n\} \times \{1,\ldots,n\}$. (We will use $S(i,j)$ to denote the set or ordered pairs.)
Question: Is there a way to pick out an ordered pair from each cell such that (1) all up-down neighbors have the same first coordinate, and (2) all right-left neighbors have the same second coordinate. Such a way to pick out ordered pairs is called ***a solution***.

Theorem 8.51.

1. *There is a k-linear FPT reduction from* Clique$_k$ *to* Grid$_k$.
2. Grid$_k$ *is W one-hard. (This follows from Part 1.)*
3. *Let f be any computable function. Assuming ETH,* Grid$_k$ *requires $f(k)n^{\Omega(k)}$ (This follows from Part 1 and Exercise 8.49.)*

Proof. Here is the reduction:

1. Input (G,k). Let $G = (V,E)$. We assume $V = \{1,\ldots,n\}$. The k-parameter for the Grid problem will be k, and the n will be n.
2. For $1 \le i,j \le n$ we define the set $S(i,j)$:
   (a) For $1 \le i \le n$, $S(i,i) = \{(a,a) \mid 1 \le a \le n\}$. (So all of the diagonal cells have the same set of ordered pairs.)
   (b) For $1 \le i < j \le n$, $S(i,j) = \{(a,b) \mid \{a,b\} \in E\}$. (So all of the off-diagonal cells have the same set of ordered pairs. Note that if a cell has (a,b) then it also has (b,a).)

If G has k-clique $\{v_1,\ldots,v_k\}$ then there is a solution to the Grid problem:

1. For $1 \le i \le k$ pick (v_i,v_i) from $S(i,i)$.

2. For $1 \le i < j \le n$ pick (v_i, v_j) out of $S(i,j)$ and $S(j,k)$. Note that since $\{v_1, \dots, v_k\}$ is a clique, $(v_i, v_j) \in E$, so $(v_i, v_j) \in E$.

We leave the following as an exercise: if the Grid has a solution then G has a clique of size k. □

We will now use the lower bounds on Grid$_k$ to get lower bounds on a parameterized version of list coloring.

> List Coloring
> *Instance:* Graph $G = (V, E)$, and, for every $v \in V$ a subset L_v of colors. We take the colors to be $\{1, \dots, n\}$ and note that n is *not* the number of vertices.
> *Question:* Is there a proper coloring of G where vertex v is colored by some color in L_v? When the problem is restricted to planar graphs of treewidth k we call it Planar List Coloring$_k$.

Theorem 8.52.

1. *There is a k-linear FPT reduction from Grid$_k$ to Planar List Coloring$_k$.*

2. *Planar List Coloring$_k$ is Wone-hard. (This follows from Part 1.)*

3. *Let f be any computable function. Assuming ETH, Planar List Coloring$_k$ requires $f(k)n^{\Omega(k)}$. (This follows from Part 1 and Exercise 8.49.)*

Proof. Here is the reduction:

1. Input: k, n, a $k \times k$ grid of cells $S(i,j)$ which contain a set of element of $\{1, \dots, n\} \times \{1, \dots, n\}$. We will call this instance of Grid$_k$, $\mathcal{G}$.

2. For every $a, a' \in \{1, \dots, n\}$ such that $a \ne a'$, and every $b, b' \in \{1, \dots, n\}$ (no restriction), create a vertex v with $L_v = \{(a,b), (a',b')\}$. Let X be the set of these $\binom{n}{2}n^2$ vertices. We will need several copies of X but we do not subscript it to avoid to much notation.

3. For every $1 \le i \le j \le n$ create node $v_{i,j}$ with $L_{v(i,j)} = S(i,j)$.

4. We now put in the horizontal edges. For every $1 \le i \le k-1$ and $1 \le j \le k$ take a copy of X. Put an edge between (1) $v_{i,j}$ and every vertex in X, and (2) $v_{i+1,j}$ and every vertex in X.

5. We now put in the vertical edges. For every $1 \le i \le k$ and $1 \le j \le k-1$ take a copy of X. Put an edge between (1) $v_{i,j}$ and every vertex in X, and (2) $v_{i,j+1}$ and every vertex in X.

6. Call the resulting graph together with the lists of colors (G, L).

Exercise 8.53 completes the proof. □

Exercise 8.53. This exercise refers to Theorem 8.52

1. Show that $\mathcal{G}$ has a solution if and only if (G, L) has a list coloring.

2. Show that G has treewidth $\le k$.

8.14.2 Grid tiling with $\le$

We now look at a variant of Grid Tiling that will help us prove a lower bound on the Scattered Set Problem.

> The k-Grid Tiling LE Problem(Grid LE$_k$)
> *Instance:* A $k \times k$ grid, an $n \in \mathbb{N}$, and, for each cell $S(i, j)$ in the grid, a subset of $\{1, \dots, n\} \times \{1, \dots, n\}$. (We will use $S(i, j)$ to refer to the set of ordered pairs.)
> *Question:* Is there a way to pick out an ordered pair from each cell such that (1) all up-down neighbors have the left first coordinate $\le$ the right first coordinate, and (2) all left-right neighbors have the upper second $\le$ to the lower second coordinate. Such a way to pick out ordered pairs is called ***a solution***.

Theorem 8.54.

1. *There is a k-linear FPT reduction from Grid$_k$ to Grid LE$_k$.*
2. *Grid LE$_k$ is Wone-hard. (This follows from Part 1.)*
3. *Let f be any computable function. Assuming ETH, Grid LE$_k$ requires $f(k)n^{\Omega(k)}$. (This follows from Part 1 and Exercise 8.49.)*

Proof. Here is the reduction:

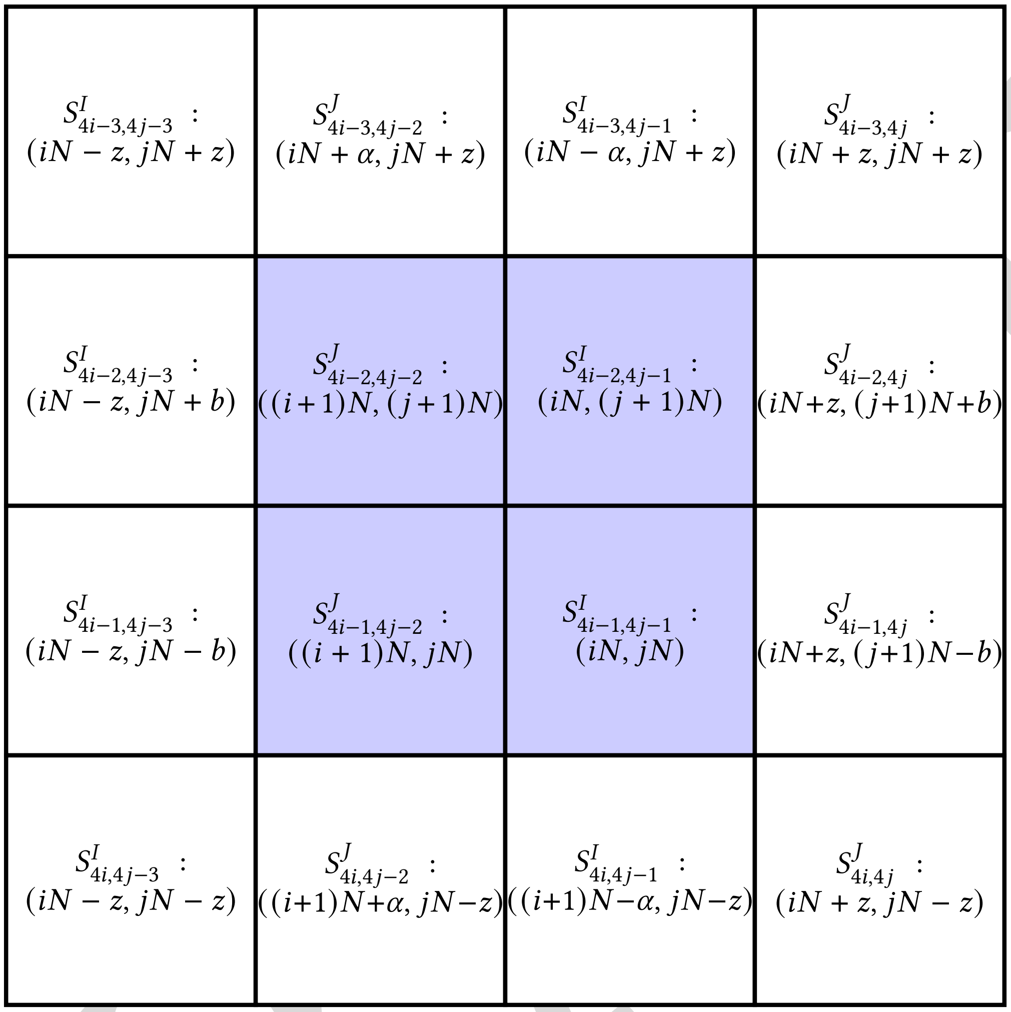
Takes the same input as Grid Tiling, but instead requires that the first coordinate of $x_{i,j} \le$ the first coordinate of $x_{i+1,j}$. Similarly, the second coordinate of $x_{i,j} \le$ the second coordinate of $x_{i,j+1}$. We can prove this problem is Wone-hard and imply no $f(k)n^{o(k)}$ algorithm by reduction from Grid Tiling. We use the following gadget to blow up each tile in our original instance into a a grid of four by four tiles. □

> Scat Set)
> *Instance:* A graph G and two number k, d.
> *Question:* Are there k vertices with pairwise distances $\ge d$?

If $d = 2$ then the Scat Set problem is just Independent Set, which we already know is Wone-hard (though we do not know if ETH yields an $f(k)n^{\Omega(k)}$ lower bound). However, the Planar Independent set problem is FPT. What about Planar Scat Set? We state what is known:

Theorem 8.55.

1. *There is a k-linear FPT reduction from Grid LE$_k$ to Scat Set.*
2. *Scat Set is Wone-hard. (This follows from Part 1.)*
3. *Let f be any computable function. Assuming ETH, Scat Set requires $f(k)n^{\Omega(k)}$. (This follows from Part 1 and Exercise 8.49.)*



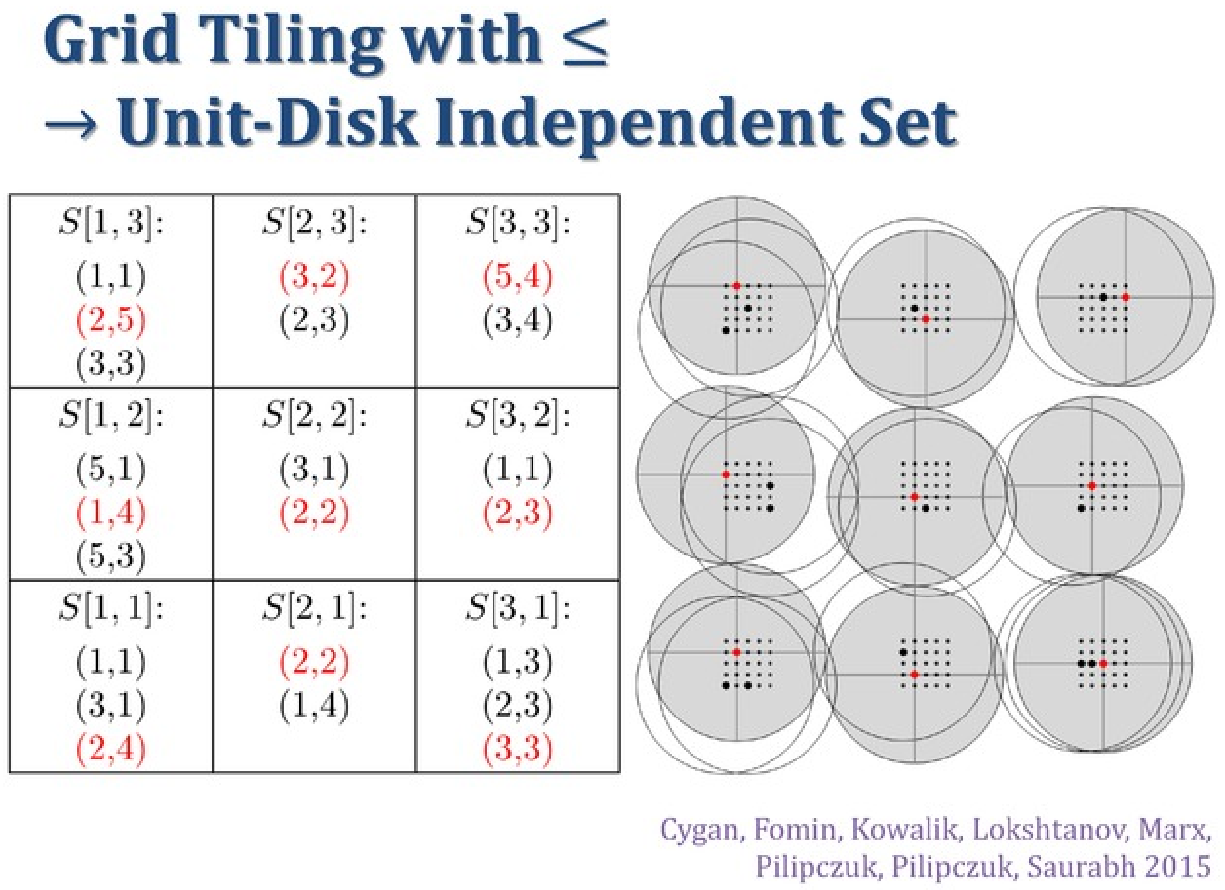
$S^I_{4i-3,4j-3}$: $(iN - z, jN + z)$	$S^J_{4i-3,4j-2}$: $(iN + \alpha, jN + z)$	$S^I_{4i-3,4j-1}$: $(iN - \alpha, jN + z)$	$S^J_{4i-3,4j}$: $(iN + z, jN + z)$
$S^I_{4i-2,4j-3}$: $(iN - z, jN + b)$	$S^J_{4i-2,4j-2}$: $((i+1)N, (j+1)N)$	$S^I_{4i-2,4j-1}$: $(iN, (j+1)N)$	$S^J_{4i-2,4j}$: $(iN+z, (j+1)N+b)$
$S^I_{4i-1,4j-3}$: $(iN - z, jN - b)$	$S^J_{4i-1,4j-2}$: $((i+1)N, jN)$	$S^I_{4i-1,4j-1}$: (iN, jN)	$S^J_{4i-1,4j}$: $(iN+z, (j+1)N-b)$
$S^I_{4i,4j-3}$: $(iN - z, jN - z)$	$S^J_{4i,4j-2}$: $((i+1)N+\alpha, jN-z)$	$S^I_{4i,4j-1}$: $((i+1)N-\alpha, jN-z)$	$S^J_{4i,4j}$: $(iN + z, jN - z)$

Figure 8.4: Grid Tiling Reduction

8.14.3 Unit disk graphs

Unit Disk Graph
Instance: A set of points P in the plane.
Question: Can you draw k unit disks centered on $p \in P$ without the disks intersecting.

We can prove this problem is Wone-hard and imply no $f(k)n^{o(\sqrt{k})}$ algorithm by reduction from Grid Tiling $\leq$. We create arrange k^2 n by n grids of dots in a k by k grid. For each of these n by n grids, only the $S_{i,j}$ points are present. We are forced to choose one point in each n by n grid of dots, and the unit disks around these points will only not intersect if the coordinates of the points we choose in adjacent n by n grids are monotonically increasing.



Grid Tiling with ≤
→ Unit-Disk Independent Set

$S[1,3]$: (1,1) (2,5) (3,3)	$S[2,3]$: (3,2) (2,3)	$S[3,3]$: (5,4) (3,4)
$S[1,2]$: (5,1) (1,4) (5,3)	$S[2,2]$: (3,1) (2,2)	$S[3,2]$: (1,1) (2,3)
$S[1,1]$: (1,1) (3,1) (2,4)	$S[2,1]$: (2,2) (1,4)	$S[3,1]$: (1,3) (2,3) (3,3)

Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk, Saurabh 2015

Figure 8.5: Unit Disk

8.15 Further Results

8.15.1 Another Look at Dom Set

We have stated that Dom Set is W[2]-complete and hence unlikely to be in FPT. However, using ETH and SETH, one can obtain sharper bounds on the parameterized complexity of Dom Set.

Let $k \in \mathbb{N}$. Let Dominating Set$_k$ be the problem of, given a graph G, is there a Dominating set of size k. Clearly this problem is in time $O(n^{k+1})$. Eisenbrand & Grandoni [EG04] have obtained slightly better algorithms. We state two known lower bounds. They are probably folklore since our only source is a workshop on fine-grained complexity held by the Max Plank Institute in 2019:
https://www.cs.umd.edu/gasarch/BLOGPAPERS/maxplankfinegrained.pdf

Theorem 8.56.

1. *Assume ETH. There exists $\delta > 0$ such that, for large k, Dominating Set$_k$ requires time $\Omega(n^{\delta k})$.*

2. *Assume SETH. Let $k \geq 3$ and $\varepsilon > 0$.* Dominating Set$_k$ *requires time* $\Omega(n^{k-\varepsilon})$.

Those same notes leave the following as an exercise:

Exercise 8.57. Assume ETH. Show that Subset Sum cannot be solved in time $2^{o(n)}$.

8.15.2 Graph Problems

1. The Planar Multiway Cut Problem: Given a planar graph G with k terminal vertices, find a minimum set of edges whose removal pairwise separates the terminals from each other. k is the parameter. Marx et al. [Mar12] showed that, (1) assuming ETH, for all computable f there is no $f(k) \cdot n^{o(\sqrt{k})}$, time algorithm for this problem and (2) this problem is W[1]-hard.

2. Firefighter problem: Consider the following model of how fires spread. A graph $G = (V, E)$ and a vertex $s \in V$ are given. At time $t = 0$ vertex s ignites. Firefighters then protect one node from being burned. At time $t = 1$ all of the neighbors of s that are not protected ignite. Firefighters then protect one node. At time t all unprotected neighbors of burning vertices ignite, and the firefighters protect one vertex. The process continues until the fire can no longer spread. The problem is to to find a strategy for the firefighters that minimizes the number of burned vertices. This problem is NP-hard. (It is not known to be in NP.) Bazgan et al. [BCC$^+$14] showed that if you take the parameter to be either the number of saved vertices, the number of burned vertices, or the number of protected vertices, then the problem is W[1]-hard.

8.15.3 Restrictions on Graphs

Some graph problems are in FPT if the graphs are restricted. Courcelle [Cou90] and independently Borie et al. [BPT92] showed the following: Let $\mathcal{P}$ be some graph property that is definable in Monadic Second Order Logic, Let $k \in \mathbb{N}$. There is a linear time algorithm for $\mathcal{P}$ restricted to graphs of treewidth $\leq k$. (See Flum & Grohe [FG06] for a complete treatment of Courcelle's theorem and its applications.) From this theorem the following problems are FPT with the parameter being the tree-width.

1. Given a Boolean Circuit, is it satisfiable?

2. Given a graph G and a number k, is G k-colorable?

3. Given a graph G and a number k, does G gave an Independent set of size k?

4. Given a graph G and a number k, is the crossing number of $G \leq k$? (This one has parameter k + Treewidth.)

These results raised the question of whether bounding the clique width can also be used to put a problem into FPT.

Let f be any computable function. Let t bound the clique number. Assume ETH. Fomin et al. [FGLS10] showed that the following problems cannot be solved in time $f(t)n^{o(t)}$.

1. Edge Domination Set: Given a graph $G = (V, E)$ and a $k \in \mathbb{N}$ is there a set $E' \subseteq E$ such that (a) $|E| \leq k$, and (b) every $e \in E$ is either in E' or shares a vertex with some $e' \in E'$.

2. Max-Cut: Given a graph G, find the largest cut of edges such that the graph on these edges form a bipartite graph.

3. Maximum Bisection: Given graph G and integer k, decide if there exists a cut of G into two equally sized vertex sets such that the cut has size at least k.

8.15.4 Problems From Computational Geometry

Bonnet & Miltzow [BM20] showed the following.

1. Point Guard Art Gallery: Given a simple polygon $\mathcal{P}$ on n vertices, two points are visible if the line segment between them is in $\mathcal{P}$. Find the minimum set S such that every point in $\mathcal{P}$ is visible from a point in S. Bonnet & Miltzow [BM20] showed the following: (1) assuming ETH, for any computable function f, this problem has no algorithms in time $f(k)n^{o(k/\log k)}$ (2) with parameter $|S|$, this problem is W[1]-hard.

2. The Vertex Guard Art Gallery: The same problem as Point Guard Art Gallery but S is now a subset of $\mathcal{P}$. Bonnet & Miltzow [BM20] showed the following: (1) assuming ETH, for any computable function f, this problem has no algorithms in time $f(k)n^{o(k/\log k)}$ (2) with parameter $|S|$, this problem is W[1]-hard.

3. Hypervolume Indicator: is a measure for the quality of a set of n solutions in $\mathbb{R}^d$. The parameter is d . Bringmann & Friedrich [BF13] showed the following: (1) assuming ETH the problem has no algorithm in time $n^{o(d)}$, (2) the problem is W[1]-hard (3) there is an average case FPT algorithm.

Chapter 9

Basic Lower Bounds on Approximability via PCP and Gap Reductions

9.1 Introduction

In this chapter we will show that some problems are NP-hard to approximate.

When Cook and Levin showed SAT is NP-complete they *could not* take some known NP-complete set A and show $A \le_p$ SAT since *there were not known NP-complete sets!* SAT was the first one. We are initially in the same position with regard to hardness-of-approximation. To show a problem is hard to approximate we cannot use a reduction. We need a basic problem (actually several basic problems) analogous to SAT for NP-completeness, that we have reason to believe is hard-to-approximate.

To show a problem is hard-to-approximate we will show that approximating it is NP-hard.

Convention 9.1. When working with a function problem we will use the notation OPT(x) for the optimal value. For example we will use OPT(x) instead of Vertex Cover(x). When we use OPT, the problem at hand is understood. If we are dealing with two different problems we may use subscripts like $\text{OPT}_{\text{Vertex Cover}}$.

We use TSP and Max 3SAT (to be defined) as running examples. Recall that TSP is the following:

- Input: a weighted graph G and a number k.
- Determine whether there is a Hamiltonian cycle of weight $\le k$.

For most of this book we have looked at *decision problems* where every instance has a *yes* or *no* answer. For example, TSP is a YES-NO question.

In the real world TSP is *not* a decision problem; indeed, in the real world one wants to *find* the optimal (minimum weight) cycle. This is the *function* version of TSP. We touched on this distinction in Chapter 0 and concluded (correctly) that, with regard to polynomial time, the decision problem and the function problem are equivalent. But let's get back to the real world. One way to cope with a problem being NP-hard is to approximate it. This concept only makes sense if we are talking about a *function*, not a *set*. In this chapter we will look at functions that are naturally associated to NP-complete problems and show that they are NP-hard to approximate. In this chapter we will show how to use *Gap Reductions* to get lower bounds on approximations contingent on P ≠ NP.

We now define Max 3SAT which will be one of our basic problems.

> Max 3SAT
> *Instance:* A 3CNF formula φ.
> *Question:* What is the max number of clauses that can be satisfied simultaneously?
> (One might also ask for the assignment that achieves this max.)

Definition 9.2. An ***optimization problem*** consists of the following:

- The set of instances of the problem (e.g., the set of weighted graphs for TSP, the set of 3CNF formulas for Max 3SAT).
- For each instance: the set of possible solutions (e.g., the set of Hamiltonian cycles for TSP, the set of truth assignments for Max 3SAT).
- For each solution: a nonnegative cost or benefit (e.g., the cost of the Hamiltonian cycle for TSP, the number of clauses that are satisfied for Max 3SAT).
- An objective: either min or max (e.g., min cost of a Hamiltonian cycle for TSP, max the number of clauses that are satisfied for Max 3SAT).

The goal of an optimization problem is to find a solution which achieves the objective: either minimize a cost or maximize a benefit.

Now we can define an NP-optimization problem. The class of all NP-optimization problems is called NPO; it is the optimization analog of NP.

Definition 9.3. An ***NP-optimization problem*** is an optimization problem with the following additional requirements:

- All instances and solutions can be recognized in polynomial time (e.g., (1) TSP: you can tell whether a proposed cycle is Hamiltonian, (2) Max 3SAT: you can tell whether a proposed string is a truth assignment of length n).
- All solutions are of length polynomial in the length of the instance which they solve (e.g., (1) TSP: a Hamiltonian cycle is clearly of length polynomial in the size of the graph—it's actually shorter, (2) Max 3SAT: An assignment is clearly of length polynomial in the size of the formula—it's actually shorter).
- The cost or benefit of a solution can be computed in polynomial time (e.g., (1) TSP: given a Hamiltonian cycle in a weighted graph, one can easily find the weight of the cycle, (2) given an assignment for a formula, one can easily find the number of clauses it satisfies).

We can convert any NPO problem into an analogous decision problem in NP. For a min problem we ask "Is $\mathrm{OPT}(x) \leq q$?" and for a maximization problem we ask "Is $\mathrm{OPT}(x) \geq q$?" The optimal solution can serve as a short easily verified certificate of a "yes" answer, and so these analogous decision problems are in NP.

This means that NPO is, in some sense, a generalization of NP problems.

Convention 9.4. For the rest of this chapter *problem* means *NPO problem*. When A and B are mentioned they are NPO problems. We will often say whether A is a min-problem or a max-problem.

Note that when we speak of solving an NPO problem we only mean finding $\mathrm{OPT}(x)$ which is the cost or benefit of the optimal solution. So we are not quite in the real world: for us a solution to the TSP problem is just to say how much the min Ham cycle costs, not to find it. However, this will suit our purposes:

- We will show that it's hard to approximate OPT up to certain factors. Hence clearly it will be hard to find an approximate solution.
- All of the algorithms in the literature for approximation problems actually do find an approximate solution.

9.2 Approximation Algorithms

Let's say you are trying to find a polynomial-time algorithm that will return the cost of the optimal TSP solution. You do not succeed; however, you get a polynomial-time algorithm that returns a number that is at most twice the optimal. Is that good? Can you prove that no algorithm is better unless P = NP? Before asking these questions we need to define our terms.

Definition 9.5. In the two definitions below, ALGORITHM is a polynomial-time algorithm and $c \geq 1$ is a constant. (We will later generalize to the case where c is a function.)

- Let A be a min-problem. ALGORITHM is a ***c-approximation algorithm for A*** if, for all valid instances x,
$$\textsc{Algorithm}(x) \leq c\mathrm{OPT}(x).$$
- Let A be a max-problem. ALGORITHM is a ***c-approximation algorithm for A*** if, for all valid instances x,
$$\mathrm{OPT}(x) \leq c\textsc{Algorithm}(x).$$

Some caveats and conventions.

1. The definition of c-approximation for a min-problem makes sense. For example, if you have an algorithm for approximate Euclidean TSP (we do!) that returns a number that is $\leq \frac{3}{2}\mathrm{OPT}$, that is called a $\frac{3}{2}$-approximation algorithm.
2. The definition of c-approximation for a max-problem is awkward. or example, if you have an algorithm for approximate MAX 3SAT that returns a number that is $\geq \frac{7}{8}\mathrm{OPT}$ (we do!), that is called an $\frac{8}{7}$-approximation algorithm. Really!
3. We personally do not like this definition. We will use it for min problems. For max problems we will either avoid it or make sure to remind the reader about what is going on.
4. Why is the definition the way it is? Because this way in both max and min cases we seek c-approximation algorithms where $c \geq 1$. We will later see that this makes the definition of PTAS smooth.

In some sense an approximation algorithm is doing pretty well if it is a c-approximation algorithm with some constant value c. But sometimes, we can do even better! There are cases where, for all $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation.

Definition 9.6.

1. Let A be a min-problem. A ***polynomial time approximation scheme (PTAS)*** for A is an algorithm that takes as input (x, ε) (x an instance of A and $\varepsilon > 0$) and outputs a solution that is $\le (1+\varepsilon)\mathrm{OPT}(x)$. The only runtime constraint is that if we fix ε, the PTAS must run in time polynomial in the size of the remaining input. Note that this allows very bad running times in terms of ε. For example, a PTAS can run in time n^{1/ε^2} because for any given value of ε this is a polynomial runtime.

2. Let A be a max-problem. A ***polynomial time approximation scheme*** (PTAS) for A is an algorithm that takes as input (x, ε) (x an instance of A and $\varepsilon > 0$) and outputs a solution that is $\ge (1-\varepsilon)\mathrm{OPT}(x)$. We have the same runtime constraint as in part 1.

We can now define several complexity classes:

Definition 9.7.

1. The class PTAS is the set of all problems for which a PTAS exists. We use the term PTAS for both the type of an approximation algorithm and the set of all problems that have that type of approximation algorithm.

2. Let A be a min-problem. $A \in$ APX if there is a constant $c \ge 1$ and an algorithm M such that $M(x)$ is $\le c \times \mathrm{OPT}(x)$. (This is just a c-approximation; however, we phrase it this way so that you will see the other classes are variants of it.)

3. Let A be a max-problem. $A \in$ APX if there is a constant $c \ge 1$ and an algorithm M such that $M(x)$ is $\ge \frac{1}{c} \times \mathrm{OPT}(x)$.

4. Let A be a min-problem. $A \in$ Log-APX if there is a constant $c \ge 1$ and an algorithm M such that $M(x)$ is $\le c \times \log x \times \mathrm{OPT}(x)$.

5. Let A be a max-problem. $A \in$ Log-APX if there is a constant c and an algorithm M such that $M(x)$ is $\ge \frac{1}{c \log x} \times \mathrm{OPT}(x)$.

6. Let A be a min-problem. $A \in$ Poly-APX if there is a polynomial p and an algorithm M such that $M(x)$ is $\le p(x) \times \mathrm{OPT}(x)$.

7. Let A be a max-problem. $A \in$ Poly-APX if there is a polynomial p and an algorithm M such that $M(x)$ is $\ge \frac{1}{p(x)} \times \mathrm{OPT}(x)$.

The following examples are known.

Example 9.8. All of our examples are variants of TSP.

1. The ***metric TSP*** problem is the TSP problem restricted to weighted graphs that are symmetric and satisfy the triangle inequality: $w(x,y)+w(y,z) \geq w(x,z)$. There is an algorithm discovered independently by Christofides [Chr76] (in 1976) and Serdyukov [Ser78] (in 1978) that gives a $\frac{3}{2}$-approximation to the metric TSP problem. Hence the metric TSP problem is in APX.

2. Karlan, Klein, Oveis-Gharan [KKOG20], in 2020, obtained the first improvement over the $\frac{3}{2}$-approx. They showed that there is a $(\frac{3}{2}-\varepsilon)$-approximation to the metric TSP problem where $\varepsilon > 10^{-36}$. This does not improve the class that metric TSP is in—it is still in APX—but it is interesting that one can do better than $\frac{3}{2}$ which was, until this result, a plausible limit on approximation.

3. The ***Euclidean TSP*** problem is the TSP problem when the graph is a set of points in the plane and the weights are the Euclidean distances. Arora [Aro07] and Mitchell [Mit99], in 1998, independently showed a PTAS for the Euclidean TSP problem. Both of their algorithms will, on input n points in the plane (which defines the weighted graph) and ε, produce a $(1+\varepsilon)$-approximation in time $O(n(\log n)^{O(1/\varepsilon)})$.

4. Arora and Mitchell actually have an algorithm that works on n points in $\mathbb{R}^d$ that runs in time $O(n(\log n)^{O(\sqrt{d}/\varepsilon)^{d-1}})$.

9.3 The Basic Hard-to-Approximate Problems

The following are basic hard-to-approximate problems. We include both the upper and the lower bounds. All of the lower bounds are under the assumption P ≠ NP.

1. TSP ∉ Poly-APX.

2. Clique ∈ Poly-APX − Log-APX.

3. Set Cover ∈ Log-APX − APX.

4. Max 3SAT ∈ APX − PTAS.

From the results listed above we have the following.

Theorem 9.9. *If P ≠ NP then*

$$\textit{PTAS} \subset \textit{APX} \subset \textit{Log-APX} \subset \textit{Poly-APX}.$$

We will discuss all four of the results in the order given above. The lower bound on approximating TSP is elementary and we can present it in its entirety. The lower bound on approximating Clique and Max 3SAT use the PCP machinery and hence we will have a section on that. The lower bound on approximating Set Cover uses a different machinery which is a close cousin to the PCP machinery; however, we will not discuss it.

9.4 Lower Bounds on Approximating TSP

Recall that, by Example 9.8, the metric TSP problem (where $w(a,b) + w(b,c) \le w(a,c)$) is in APX. What about TSP problems without that condition? We show that if TSP $\in$ Poly-APX, then P = NP. Let OPT(H) be the cost of the min Hamiltonian cycle in weighted graph H. Informally, we will map instances G of Ham Cycle to instances G' of TSP such that:

If $G \in$ Ham Cycle then OPT(G') is small.
If $G \notin$ Ham Cycle then OPT(G') is large.

We will then use the alleged approximation algorithm for TSP to determine which is the case. This is called a ***Gap Reduction*** because of the large gap between the costs of the optimal routes.

Theorem 9.10. *If TSP $\in$ Poly-APX then P = NP.*

Proof. Assume, by way of contradiction, that TSP $\in$ Poly-APX with polynomial $p(n)$. To avoid notational clutter we call this ***the approx algorithm***. We use this assumption to show that Ham Cycle $\in$ P.

Let $c(n)$ be a polynomial to be named later. As part of our reduction we will map instances G of Ham Cycle to instances G' of TSP such that:

If $G \in$ Ham Cycle then OPT(G') $= n$.
If $G \notin$ Ham Cycle then OPT(G') $\ge c(n)$.

Here is an algorithm for Ham Cycle.

1. Input $G = (V, E)$, an unweighted graph.
2. Create an instance G' of TSP as follows: (1) if $e \notin E$ then give e weight $c(n)$, (2) if $e \in E$ then give e weight 1.
3. (This is a comment, not part of the algorithm.)
   (a) If $G \in$ Ham Cycle then OPT(G') $\le n$ since you can just use the Hamiltonian cycle.
   (b) If $G \notin$ Ham Cycle then OPT(G') $\ge c(n)$ since any cycle in G' will have to use at least one edge of cost $c(n)$ (actually OPT(G') $\ge c(n) + n - 1$ but this is not needed).
4. Run the approx algorithm on G'.
5. (This is a comment, not part of the algorithm.)
   (a) If $G \in$ Ham Cycle then the approx alg run on G' returns a route of size $\le np(n)$.
   (b) If $G \notin$ Ham Cycle then the approx alg run on G' returns a route of size $\ge c(n)$.

   To ensure these cases do not overlap we pick $c(n) > np(n)$.
6. If the approx alg outputs a number $\le np(n)$ then output YES. If the approx alg outputs a number $> c(n)$ then output NO. By the commentary in the algorithm, no other case will occur.

□

The proof of Theorem 9.10 took G and *produced a gap*. We actually showed the following:

Theorem 9.11. *Let $c(n) = np(n)$ where $p(n)$ is a polynomial of degree ≥ 1. Let $c(n)$-GAP-TSP be defined as follows. Given a TSP problem $G = (V, E, w)$ where you are promised that exactly one of the following occurs:*

- *There is a solution of cost n.*
- *All solutions have cost $\geq c(n)$.*

Then, for all $c(n)$, Ham Cycle reduces to $c(n)$-Gap-TSP.

9.5 The Gap Lemmas

In this section we prove two easy lemmas that show how to use a reduction that causes a gap (like the one in Theorem 9.10) to obtain a lower bound on approximation algorithms. This first lemma is for max-problems, and the second one is for min-problems. The proofs are similar, hence the proof of the second one is omitted.

Convention 9.12. We will often use the notation $|y|$. This is the size of y; however, we will use *size* in a different way for different inputs.

1. If y is a string then $|y|$ is the length of y.
2. If y is a graph then $|y|$ is the number of vertices.
3. If y is a 3CNF formula then $|y|$ might be either the number of variables or the number of clauses depending on our application.

Definition 9.13. Let g be a max-problem (e.g., Clique). Let $a(n)$ and $b(n)$ be functions from $\mathbb{N}$ to $\mathbb{N}$ such that $\frac{b(n)}{a(n)} < 1$. Then Gap$(g, a(n), b(n))$ is the following problem.

Gap$(g, a(n), b(n))$
Instance: y for which you are promised that either $g(y) \geq a(|y|)$ or $g(y) \leq b(|y|)$.
Question: Determine which is the case.

Lemma 9.14. Let A be an NP-hard set. Let g be a max-problem. Let $a(n)$ and $b(n)$ be functions from $\mathbb{N}$ to $\mathbb{N}$ such that (1) $\frac{b(n)}{a(n)} < 1$, and (2) b is computable in time polynomial in n. Assume there exists a polynomial time reduction that maps x to y such that the following occurs:

- If $x \in A$ then $g(y) \geq a(|y|)$.
- If $x \notin A$ then $g(y) \leq b(|y|)$.

Then:

1. Gap$(g, a(n), b(n))$ is NP-hard (this follows from the premise).

2. If there is an approximation algorithm for g that, on input y, returns a number $> \frac{b(|y|)}{a(|y|)}g(y)$, then P = NP.

Proof. We just prove part 2.

We use the reduction and the approximation algorithm to obtain $A \in$ P. Since A is NP-hard we obtain P = NP.

Algorithm for A

1. Input x.
2. Run the reduction on x to get y.
3. Run the approximation algorithm on y.
4. (This is a comment and not part of the algorithm.)
   $x \in A \rightarrow g(y) \geq a(|y|) \rightarrow$ approx on y returns $> \frac{b(|y|)}{a(|y|)}a(|y|) = b(|y|)$.
   $x \notin A \rightarrow g(y) \leq b(|y|) \rightarrow$ approx on y returns $\leq b(|y|)$.
5. If the approx returns a number $> b(|y|)$ then output YES. Otherwise output NO. (This is the step where we need $b(|y|)$ to be computable in time polynomial in $|y|$.)

□

We now look at min-problems.

We use the same name, Gap$(g, a(n), b(n))$ for the following problem.

Definition 9.15. Let g be a min-problem (e.g., TSP). Let $a(n)$ and $b(n)$ be functions from $\mathbb{N}$ to $\mathbb{N}$ such that $\frac{b(n)}{a(n)} > 1$. Then Gap$(g, a(n), b(n))$ is the following problem.

We use the same notation Gap$(g, a(n), b(n))$ for min-problems as we did for max-problems. When we use these lemmas the meaning will be clear from context.

Gap$(g, a(n), b(n))$
Instance: y for which you are promised that either $g(y) \leq a(|y|)$ or $g(y) \geq b(|y|)$.
Question: Determine which is the case.

We now state a lemma that is useful for obtaining lower bounds on approximation min-problems. The proof is similar to that of Lemma 9.14 and hence is omitted.

Lemma 9.16. Let A be an NP-complete set. Let g be a min-problem. Let $a(n)$ and $b(n)$ be functions from $\mathbb{N}$ to $\mathbb{N}$ such that (1) $\frac{b(n)}{a(n)} > 1$, and (2) b is computable in time polynomial in n. Assume there exists a polynomial time reduction that maps x to y such that the following occurs:

- If $x \in A$ then $g(y) \leq a(|y|)$.
- If $x \notin A$ then $g(y) \geq b(|y|)$.

Then:

1. Gap$(g, a(n), b(n))$ is NP-hard (this follows from the premise).

2. If there is an approximation algorithm for g that, on input y, returns a number $< \frac{b(|y|)}{a(|y|)}g(y)$, then P = NP.

Definition 9.17. We will refer to reductions like the ones in Lemma 9.14 and 9.16 as ***Gap Reductions with ratio*** $\frac{b(n)}{a(n)}$.

9.6 The PCP Machinery

In this section we discuss a characterization of NP in terms of ***Probabilistically Checkable Proofs***. This characterization has a rather long proof that we will omit. However, once we have the characterization we will use it to construct gap reductions which will show some approximation problems are NP-hard.

Recall the following notation and definition.

Notation 9.18. Let $\exists^p y$ mean there exists y such that $|y|$ is of length polynomial in $|x|$, where x is understood. Let $\forall^p y$ mean for all y such that $|y|$ is of length polynomial in $|x|$, where x is understood.

Definition 9.19. $A \in$ NP if there exists a polynomial predicate B such that

$$A = \{x \mid \exists^p y : B(x, y)\}.$$

We want to rewrite this and modify it.

Definition 9.20.

1. An ***Oracle Turing Machine–bit access*** (henceforth OTM-BA) is an Oracle Turing Machine where (1) the oracle is a string of bits, and (2) the requests for the bits is made by writing down the address of the bit. By convention, if the string is s long and a query is made for bit $t > s$ then the answer is NO.

2. We denote an oracle Turing machine by $M^{()}$. If $M^{()}$ is an Oracle Turing Machine and y is the string being used for the oracle, and x is an input, we denote the computation of $M^{()}$ on x with oracle y by $M^y(x)$.

3. A ***Polynomial OTM-BA (POTM-BA)*** is an OTM-BA that runs in polynomial time. Note that a POTM-BA can use an oracle string of length 2^{poly} since it can write down that it wants bit position (say) 2^{n^2} with n^2 bits.

4. We give two equivalent definitions of a ***Randomized POTM-BA (RPOTM-BA)***. One is intuitive and the other is better for proofs.

   (a) A ***Randomized POTM-BA (RPOTM-BA)*** is a POTM-BA that is allowed to flip coins. So there will be times where, rather than do STEP A it will do STEP A with probability (say) 1/3 and STEP B with probability 2/3. Hence we cannot say *The machine accepts x using oracle bit string y* but we can say *The machine will accept x using oracle bit string y with probability* ≥ 0.65.

(b) Note that for a RPOTM-BA computation many coins are flipped and are used. We can instead think of the string of coin flips as being part of the input, and then asking what fraction of the inputs accept. Formally, a ***Randomized POTM-BA (RPOTM-BA)*** is a POTM-BA that has 2 inputs x, τ. We will be concerned with the fraction of τ's ($|\tau|$ will be a function of $|x|$) for which $M^y(x, \tau)$ accepts. We will refer to this as *the probability that x with oracle bit string y is accepted* since we think in terms of the string τ being chosen at random. We will refer to τ as a string of coin flips.

The following is an alternative definition of NP.

Definition 9.21. $A \in \mathsf{NP}$ if there exists a POTM-BA $M^{()}$ such that:
$x \in A \Rightarrow \exists^p y : M^y(x) = 1$
$x \notin A \Rightarrow \forall^p y : M^y(x) \neq 1$

If $x \in A$ then we think of y as being the EVIDENCE that $x \in A$. This evidence is short (only $p(|x|)$ long) and checkable in polynomial time. Note that the computation of $M^y(x)$ may certainly use all of the bits of y. What if we (1) restrict the number of bits of the oracle that the computation can look at, and (2) use an RPOTM-BA?

Definition 9.22. Let $q(n)$ and $r(n)$ be monotone increasing functions from $\mathbb{N}$ to $\mathbb{N}$. An ***$r(n)$-random $q(n)$-query RPOTM-BA $M^{()}$*** is a RPOTM-BA where, for all y and for all x of length n, $M^y(x)$ flips $r(n)$ coins and makes $q(n)$ queries.

Definition 9.23. Let $r(n)$ and $q(n)$ be monotone increasing functions from $\mathbb{N}$ to $\mathbb{N}$ and $\varepsilon(n)$ be a monotone decreasing function from $\mathbb{N}$ to $[0, 1]$. $A \in \mathsf{PCP}(r(n), q(n), \varepsilon(n))$ if there exists an $r(n)$-random, $q(n)$-query RPOTM-BA $M^{()}$ such that, for all n, for all $x \in \{0, 1\}^n$, the following holds.

1. If $x \in A$ then there exists y such that, for all τ with $|\tau| = r(n)$, $M^y(x, \tau)$ accepts. In other words, the probability of acceptance is 1.

2. If $x \notin A$ then for all y at most $\varepsilon(n)$ of the τ's with $|\tau| = r(n)$ make $M^y(x, \tau)$ accept. In other words, the probability of acceptance is $\leq \varepsilon(n)$.

3. (This item is not formally needed.) One of the two cases above must happen. That is, there will never be a case where (say) $\varepsilon(n) < \frac{1}{2}$ and the probability of acceptance is $2\varepsilon(n)$.

We are only going to be concerned with $r(n) = O(\log n)$ and $q(n) = O(1)$ or $O(\log n)$. We will see below that we can assume $|y| = 2^{q(n)+r(n)}$, which is polynomial in n.

Note: The queries are made adaptively. This means that the second question asked might depend on the answer to the first. Hence if $M^y(x, \tau)$ asks $O(q(n))$ questions then the total number of questions possible to ask is $2^{O(q(n))} - 1$. Since there are $2^{O(r(n))}$ values of z there are a total of roughly $2^{O(q(n)+r(n))}$ queries that can be asked. Hence we can take $|y| = 2^{q(n)+r(n)}$.

Example 9.24.

1. SAT $\in$ PCP$(0, n, 0)$. The y value is the satisfying assignment. $M^{()}$ makes all n queries and does not use random bits.

2. If φ is a formula let C be the number of clauses in it. 3SAT $\in$ PCP$(\lg(C), 3, \frac{C-1}{C})$. The y value is the satisfying assignment. M picks a random clause and queries the 3 truth assignments. If they satisfy the clause, output YES, else NO. If $\varphi \in$ 3SAT then the algorithm will return YES. If $\varphi \notin$ 3SAT then the worst case is if y satisfies all but one of the clauses, hence the probability of error is $\leq \frac{C-1}{C}$.

3. Let $L \in \mathbb{N}$. 3SAT $\in$ PCP$(L\lg(C) + O(1), 3L, \frac{C-L}{C})$. Iterate the proof in part 2 L times.

Arora et. al [ALM+98], building on the work of Arora et. al [AS98], proved the following Theorem.

We omit the proof which is difficult.

Theorem 9.25.

1. *SAT $\in$ PCP$(O(\log n), O(1), \frac{1}{2})$.*

2. *For all constants $0 < \varepsilon < 1$, SAT $\in$ PCP$(O(\log n), O(1), \varepsilon)$. (This is easily obtained by iterating the protocol from Part 1.)*

3. *SAT $\in$ PCP$(O(\log^2 n), O(\log n), \frac{1}{n})$. (This is easily obtained by iterating the protocol from Part 1.)*

The result SAT $\in$ PCP$(O(\log^2 n), O(\log n), \frac{1}{n})$ is *not* good enough for proving problems hard to approximate. Ajtai–Komlós–Szemerédi [AKS87] and Impagliazzo & Zuckerman [IZ89] improved it by using a technique to reuse random bits by doing a random walk on an expander graph. The next theorem is *not* in their papers; however, one can obtain it from their papers.

See Vazirani [Vaz01] (Theorem 29.18).

Theorem 9.26. *SAT $\in$ PCP$(O(\log n), O(\log n), \frac{1}{n})$.*

9.7 Clique is Hard to Approximate

Arora et. al [ALM+98], building on the work of Feige et. al [FGL+96], proved that Clique is hard to approximate. We will recognize the proof as a gap reduction.

Notation 9.27. If G is a graph then OPT(G) is the size of the largest clique in G.

We state both the upper and lower bound in this proof; however, we only prove the lower bound.

In the following theorem, the size of a graph is the number of vertices.

Theorem 9.28.

1. *There is an algorithm that, on input graph G with N vertices, outputs a clique of size $\Omega(\frac{\log^3 N}{N(\log\log N)^2} OPT(G))$. Hence Clique $\in$ Poly-APX. (Feige [Fei04] proved this. We omit the proof. We use N for the number of vertices since n will be the length of a string in an NP-set A.)*

2. *Let A be an NP-complete problem. Let c, d be such that $A \in PCP(c \lg n, d \lg n, \frac{1}{n})$ (such a c, d exists by Theorem 9.26). There is a reduction that maps $x \in \Sigma^*$ ($|x| = n$) to a graph G on $N = n^{c+d}$ vertices such that:*

   - *If $x \in A$ then $OPT(G) \geq n^c = N^{c/(c+d)}$.*
   - *If $x \notin A$ then $OPT(G) \leq n^{c-1} = N^{(c-1)/(c+d)}$.*

3. *GAP(CLIQUE, $N^{c/(c+d)}, N^{(c-1)/(c+d)}$) is NP-hard. (This follows from Part 2 and Lemma 9.14.)*

4. *If there is an approximation algorithm for CLIQUE that, on input G, where G has N vertices, returns a number $> \frac{1}{N^{1/(c+d)}} OPT(G)$, then $P = NP$. (This follows from Part 2 and Lemma 9.14.)*

5. *Assuming $P \neq NP$, CLIQUE $\in$ Poly-APX $-$ Log-APX. (This follows from parts 1,4.)*

Proof. We just prove part 2.

Let A, c, d be as in the statement of the theorem. To avoid notational clutter we say "run the PCP on (x, τ)" rather than give the RPOTM-BA for A a name.

1. Input x of length n.

2. Form a graph $G = (V, E)$ as follows:

   (a) $V = \{0,1\}^{c \lg n + d \lg n}$. Note that there are $N = n^{c+d}$ vertices.

   (b) This step will help us determine the edges. For each vertex write it as $\tau\sigma$ where $|\tau| = c \lg n$ and $|\sigma| = d \lg n$. Run the PCP on (x, τ) and answer the ith query made with the ith bit of σ. Keep track of which queries were made, what the answers were, and if the computation accepted.

   (c) We now determine the edges. Let $\tau\sigma$ and $\tau'\sigma'$ be two vertices. We know both the queries and the answers made when running PCP(x, τ) using σ for the answers, and running PCP(x, τ') using σ' for the answers. Connect the two vertices $\tau\sigma$ and $\tau'\sigma'$ if (1) both represent computations that accept, (2) $\tau \neq \tau'$, and (3) the answers to queries do not contradict.

3. Output the graph.

Note the following:

1. If $x \in A$ then there exists a consistent way to answer the bit-queries such that, for all $\tau \in \{0,1\}^{c \lg n}$, the PCP on (x, τ) accepts. Hence OPT$(G) \geq 2^{c \lg n} = n^c = N^{c/(c+d)}$.

2. If $x \notin A$ then any consistent way to answer the bit-queries will make $\leq \frac{1}{n}$ of the $\tau \in \{0,1\}^{c \lg n}$ accept. Hence OPT$(G) \leq n^{c-1} = N^{(c-1)/(c+d)}$.

□

Note: Theorem 9.28 showed that, for $\delta = \frac{1}{c+d}$, if there is an algorithm that returns a number $> \frac{1}{n^\delta}$OPT(G) then P = NP. Better results are known:

1. Hastad [Has99] showed that, for all $0 \le \delta < 1$, if there is an algorithm that returns a number $\ge \frac{1}{n^\delta}\mathrm{OPT}(G)$ then ZPP = NP.
2. Zuckerman [Zuc07] showed that, for all $0 \le \delta < 1$, if there is an algorithm that returns a number $\ge \frac{1}{n^\delta}\mathrm{OPT}(G)$ then P = NP.

9.8 Set Cover is Hard to Approximate

Set Cover
Instance: n and Sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.
Question: What is the smallest size of a subset of S_i's that covers all of the elements in $\{1, \ldots, n\}$?

Notation 9.29. An algorithm ***approximates Set Cover within a factor of $f(n)$*** if it outputs a number that is $\le f(n)$ times the optimal.

The following are known.

Theorem 9.30.

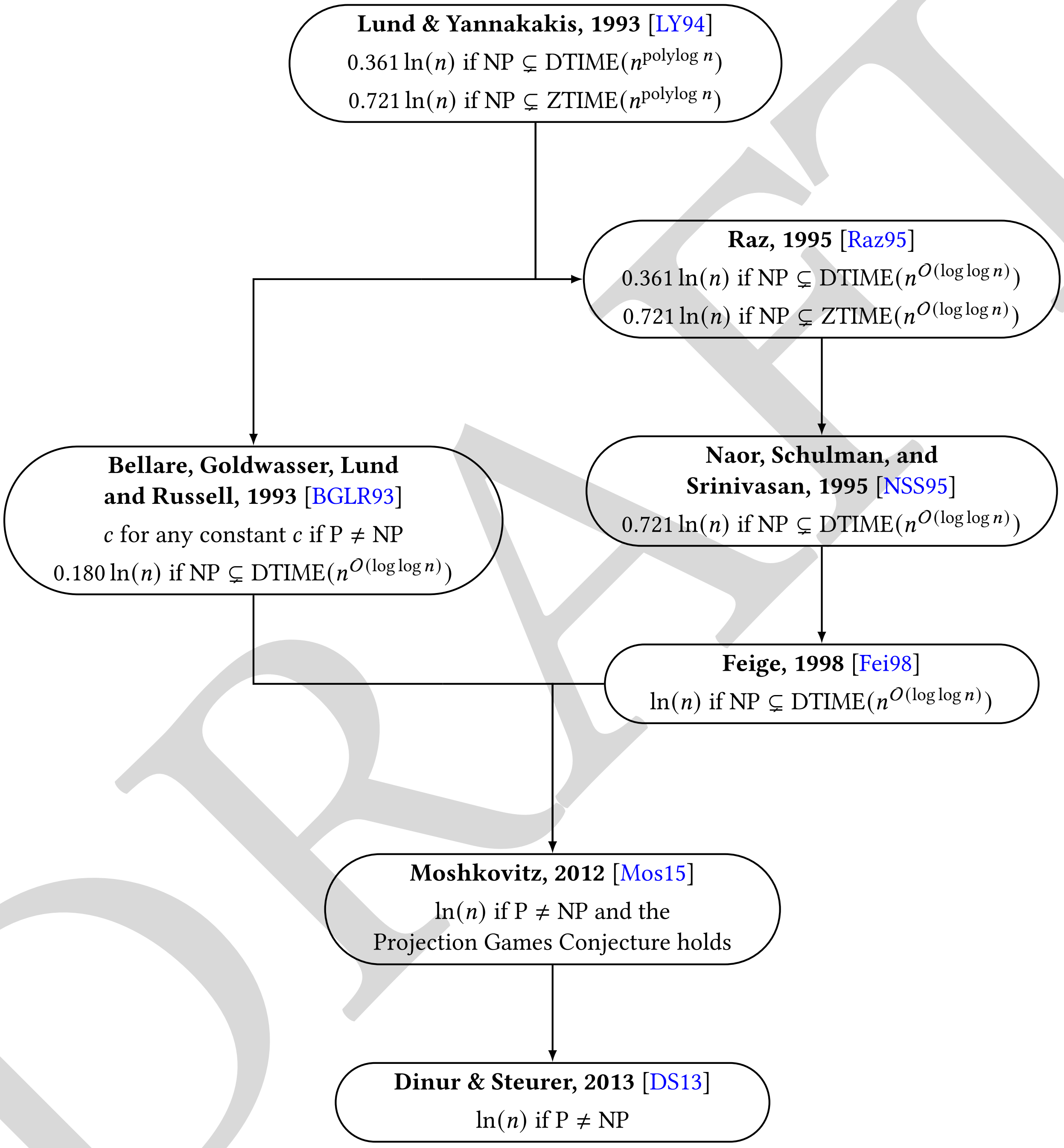
1. *Chvatal [Chv79] showed that a simple greedy algorithm approximates Set Cover within a factor of* $\ln(n)$.
2. *Slavík [Sla97] gave a slight improvement by replacing the* $\ln n$ *with*
   $\ln n - \ln(\ln n) - O(1)$.
3. *Lund & Yannakakis [LY94] showed that, for any* $0 < c < 1/4$, *if Set Cover can be approximated within a factor of* $c \lg n$ *then* $NP \subseteq DTIME(n^{\mathrm{polylog}(n)})$.
4. *Dinur & Steurer [DS13] showed that if Set Cover can be approximated within a* $(1 - o(1)) \ln n$ *then* $P = NP$.
5. *There were many intermediary results between these two. Melder [Mel21] presents a guide to the papers needed to obtain the result and how they fit together This guide is summarized in in Figure 9.1.*

Note: How does m, the number of sets, impact these results? The lower bound proofs apply for m very small, like $n^{0.0001}$. Hence, when we later do reductions of Set Cover to other problems we can assume m is small.

The lower bound papers do not use PCP's. They instead use a close cousin: 2-prover-1-round interactive proof systems.

For the next exercise we will consider the following variant of Set Cover.

Max Coverage
Instance: n, Sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, and k.
Question: What is the largest size of a set $X \subseteq \{1, \ldots, n\}$ such that there exists a set of k S_i's that contain the elements of X.



Lund & Yannakakis, 1993 [LY94]
$0.361 \ln(n)$ if NP $\subsetneq$ DTIME$(n^{\mathrm{polylog}\, n})$
$0.721 \ln(n)$ if NP $\subsetneq$ ZTIME$(n^{\mathrm{polylog}\, n})$
Raz, 1995 [Raz95]
$0.361 \ln(n)$ if NP $\subsetneq$ DTIME$(n^{O(\log\log n)})$
$0.721 \ln(n)$ if NP $\subsetneq$ ZTIME$(n^{O(\log\log n)})$
Bellare, Goldwasser, Lund and Russell, 1993 [BGLR93]
c for any constant c if P $\neq$ NP
$0.180 \ln(n)$ if NP $\subsetneq$ DTIME$(n^{O(\log\log n)})$
Naor, Schulman, and Srinivasan, 1995 [NSS95]
$0.721 \ln(n)$ if NP $\subsetneq$ DTIME$(n^{O(\log\log n)})$
Feige, 1998 [Fei98]
$\ln(n)$ if NP $\subsetneq$ DTIME$(n^{O(\log\log n)})$
Moshkovitz, 2012 [Mos15]
$\ln(n)$ if P $\neq$ NP and the Projection Games Conjecture holds
Dinur & Steurer, 2013 [DS13]
$\ln(n)$ if P $\neq$ NP

Figure 9.1: The History of Set Cover Lower Bounds

Exercise 9.31. Let $\varepsilon > 0$. Show that if there is an approximation algorithm for Max Coverage which returns $(1 - \frac{1}{e} + \varepsilon)$OPT then P = NP.
Hint: Use Theorem 9.30.4 and a gap reduction.

9.9 Max 3SAT is Hard to Approximate

Arora et. al [ALM⁺98] proved that Max 3SAT is hard to approximate. We will recognize the proof as a gap reduction.

Notation 9.32. If φ is a 3CNF formula (so every clause has ≤ 3 literals) then OPT(φ) is the max number of clauses that can be satisfied simultaneously.

We state both the upper and lower bound in this proof. We prove one of the upper bounds; however, our main interest is in the lower bound.

Notation 9.33. Let $q \in \mathbb{N}$. Then $C(q)$ is the the maximum number of clauses in a 3CNF formula on 2^q variables. Note that $C(q) = O(2^{3q})$. Since q is constant, $C(q)$ is constant.

In the following theorem, the size of a 3CNF formula is the number of clauses.

Theorem 9.34.

1. *Restrict Max 3SAT to formulas that have* exactly *three literals per clause. There is an algorithm that, given such a φ, returns a number that is $\geq 0.875OPT(\varphi)$.*

2. *Karloff & Zwick [KZ97] have a randomized polynomial time algorithm for Max 3SAT (note–clauses can have 1,2, or 3 literals) that, on input φ, does the following: (1) if $\varphi \in SAT$ returns an assignment that satisfies $\geq 0.875OPT(\varphi)$ of the clauses, (2) if $\varphi \notin SAT$ then there is good evidence that the algorithm still returns an assignment that satisfies at least $0.875OPT(\varphi)$.*

3. *Let A be NP-complete. Let $c, q \in \mathbb{N}$ such that $A \in PCP(c \lg n, q, 0.25)$ (such a c, q exists by Theorem 9.25). There is a reduction that maps $x \in \Sigma^*$ to a 3CNF formula φ such that:*

   (a) *If $x \in A$ then $OPT(\varphi) = |\varphi|$. ($\varphi \in 3SAT$ so all $|\varphi|$ clauses are satisfied.)*

   (b) *If $x \notin A$ then $OPT(\varphi) \leq \left(1 - \frac{3}{4C(q)}\right)|\varphi|$.*

   (c) *The output φ has exactly 3 literals per clause.*

4. *Gap(Max 3SAT, m, $\left(1 - \frac{3}{4C(q)}\right)m$) is NP-hard (where m is the number of clauses). This holds even if φ is restricted to having exactly 3 literals per clause. (This follows from Part 3 and Lemma 9.14.)*

5. *If there is an approximation algorithm for Max 3SAT that, on input φ, returns a number $> \left(1 - \frac{3}{4C(q)}\right)OPT(\varphi)$ then $P = NP$. (This follows from Part 3 and Lemma 9.14.)*

6. *Assuming $P \neq NP$, Max 3SAT $\in APX - PTAS$. (The problem that separates them is Max 3SAT restricted to formulas that have exactly 3 literals per clause. This Part follows from Parts 1 and 5.)*

Proof. We just prove parts 1,3.
1) We first give a randomized algorithm: Assign each variable to TRUE or FALSE at random. The probability of a particular clause being satisfied is $\frac{7}{8} = 0.875$, so by linearity of expectation we expect $\frac{7}{8}$ of the clauses to be satisfied. This gives a randomized algorithm that outputs a number $\geq 0.875\text{OPT}(\varphi)$. This algorithm can be derandomized using the method of conditional probabilities. Details can be found in either Vazirani's book [Vaz01] or Shmoys-Williamson's book [WS11].

3) Let A, c, q be as in the statement of the theorem. To avoid notational clutter we say "run the PCP on (x, τ)" rather than give the RPOTM-BA for A a name.

1. Input x.
2. Form a 3CNF formula ψ as follows:
   (a) The PCP for A can only make $2^{q+c\lg n} = 2^q n^c$ possible bit-queries. There are $2^q n^d$ variables, one for each possible bit-query.
   (b) For every $\tau \in \{0,1\}^{c\lg n}$ do the following. For every $\sigma \in \{0,1\}^q$ run the PCP(x,τ) using σ for the query answers. Keep track of which ones accepted and which ones rejected. From this information form a formula on $\leq 2^q$ variables that is TRUE if and only if the PCP accepts with those answers (using τ for the coin flips). Constructing the formula takes polynomial time since q (and hence 2^q) is constant. Convert this formula to 3CNF (this easily takes polynomial time). Call the result ψ_τ. Note that ψ_τ has between 1 and $C(q)$ clauses.
   (c) ψ is the AND of the n^c formulas from the last step. Note that ψ is in 3CNF form.

Note the following.

1. Assume $x \in A$. Then there exists a consistent way to answer the bit-queries such that, for all $\tau \in \{0,1\}^{c\lg n}$, the PCP on (x,τ) accepts. Hence every ψ_τ can be satisfied simultaneously. Therefore the fraction of clauses of ψ that can be satisfied is 1. Hence $\text{OPT}(\varphi) = |\varphi|$.

2. Assume $x \notin A$. Then every consistent way to answer the bit-queries will make $\leq \frac{1}{4}$ of the $\tau \in \{0,1\}^{c\lg n}$ accept. We need to estimate the fraction of clauses of ψ that are satisfied. This fraction is maximized when the following occurs: (1) all n^c of the ψ_τ have $C(q)$ clauses, (2) there is an assignment that satisfies all $C(q)$ clauses in $1/4$ of the ψ_τ, and $C(q)-1$ clauses in $3/4$ of the ψ_τ. Hence the fraction of clauses satisfied is

$$\frac{(n^c/4)C(q) + (3n^c/4)(C(q)-1)}{n^cC(q)} = \frac{n^cC(q) - (3n^c/4)}{n^cC(q)} = 1 - \frac{3}{4C(q)}$$

Hence $\text{OPT}(\varphi) \leq \left(1 - \frac{3}{4C(q)}\right)|\varphi|$.

□

Theorem 9.34 shows that if there is an algorithm that returns a number $> (1 - \frac{3}{4(q)})\text{OPT}(\varphi)$ then P = NP. This suffice to show MAX 3SAT $\notin$ PTAS but still leaves open the the question of whether the best known approximation, $0.875\text{OPT}(\varphi)$ is optimal. It is. Hastad [Has01] proved the following:

Theorem 9.35.

1. *Let $0 < \varepsilon < 1$. Let our formulas have $4m$ clauses. Then*

$$\text{Gap}(\text{Max 3SAT}, 4(1-\varepsilon)m, 3.5(1+\varepsilon)m)$$

   is NP-hard.

2. *Let $0 < \varepsilon < 1$. Let our formulas have m clauses. Then*

$$\text{Gap}(\text{Max 3SAT}, (1-\varepsilon)m, 0.875(1+\varepsilon)m)$$

   is NP-hard. (This follows from Part 1.) This result holds when Max 3SAT is restricted to having exactly 3 literals per clause. Hence this result is a lower bound that matches the upper bound in Theorem 9.34.1.

3. *Let $0 < \delta < 1$. If there is an approximation algorithm for Max 3SAT that, on input φ, returns a number $> \big(0.875 + \delta\big)OPT(\varphi)$ then $P = NP$. (This follows from Part 2.)*

We will use this result to obtain lower bounds on approximation Vertex Cover.

9.10 Vertex Cover is Hard to Approximate

We show a lower bound on how well Vertex Cover can be approximated by using a reduction from the a GAP version Max 3SAT that was discussed in Theorem 9.35 to a GAP version of Vertex Cover. Since we are reducing a GAP problem to a GAP problem we will not use Lemma 9.16; however, we will leave as an exercise to form a lemma similar to Lemmas 9.14 and 9.16 for reductions between GAP problems.

Theorem 9.36. *We consider Vertex Cover. There is an algorithm that will, given a graph G, output a number that is $\le 2OPT(G)$.*

Proof sketch:

Here is the algorithm.

1. Input $G = (V, E)$.
2. Let $M = \emptyset$. Add edges to M so that no two edges in M share a vertex, until you can add no more. This gives a maximal matching.
3. Output U, the set of endpoints of the edges in M.

U is a vertex cover since, if $e = (a, b)$ has neither endpoint in U then M was not maximal as it could have added e.

We leave it to the reader to show that $U \le \text{OPT}(G)$. ∎

The following theorem seems to be new; however, it is weaker than the best known results. We prove a lower bound on approximating Vertex Cover, however we give enough information in the proof so that the reader can obtain lower bounds on approximating Independent Set and Clique.

Theorem 9.37. *We are considering the problem* VERTEX COVER. *Let* $\delta > 0$. *If there is an algorithm that, on input a graph* G, *returns a number* $\le (\alpha - \delta)OPT(G)$ *where* $\alpha = 1.107$, *then* $P = NP$.

Proof. We assume that there is an algorithm that, given a graph G, returns a number $\le (\alpha - \delta)\mathrm{OPT}_{\textsc{Vertex Cover}}(G)$. To avoid notational clutter we will call it ***the algorithm***. We will prove this with α and only at the end see what α has to be to make the proof work.

Let ε be a parameter to be named later. It will depend on δ and α. By Theorem 9.35

$$\textsc{Gap}(\textsc{Max 3SAT}, (1-\varepsilon)m, 0.875(1+\varepsilon)m)$$

is NP-hard. We present an polynomial-time algorithm for this GAP problem that uses the approximation algorithm for VERTEX COVER.

We will use the fact that, if G has n vertices, then $\mathrm{OPT}_{\textsc{Independent Set}}(G) = n - \mathrm{OPT}_{\textsc{Vertex Cover}}(G)$.

1. Input $\varphi = C_1 \wedge \cdots \wedge C_m$, a formula in 3CNF where every clause has exactly 3 literals. We are promised that either

   (1) $\mathrm{OPT}_{\textsc{Max 3SAT}}(\varphi) \ge (1-\varepsilon)m$, or

   (2) $\mathrm{OPT}_{\textsc{Max 3SAT}}(\varphi) \le (0.875+\varepsilon)m$.

2. Create a graph G as follows.

   (a) For each C_i we have a vertex for each literal. Hence G has $3m$ vertices.

   (b) Put an edge between every pair of vertices in the same C_i. Put an edge between vertices from different C_i's if they contradict each other.

3. (This is commentary, not part of the algorithm).

   Note the following two cases.

   (1) $\mathrm{OPT}_{\textsc{Max 3SAT}}(\varphi) \ge (1-\varepsilon)m$, so

   (a) $\mathrm{OPT}_{\textsc{Independent Set}}(G) \ge (1-\varepsilon)m$

   (b) $\mathrm{OPT}_{\textsc{Vertex Cover}}(G) = 3m - \mathrm{OPT}_{\textsc{Independent Set}}(G) \le 3m - (1-\varepsilon)m = (2+\varepsilon)m$

   (c) $\mathrm{OPT}_{\textsc{Clique}}(\bar{(}G)) \ge (1-\varepsilon)m$

   (2) $\mathrm{OPT}_{\textsc{Max 3SAT}}(\varphi) \le (0.875+\varepsilon)m$, so

   (a) $\mathrm{OPT}_{\textsc{Independent Set}}(G) \le (0.875+\varepsilon)m$

   (b) $\mathrm{OPT}_{\textsc{Vertex Cover}}(G) = 3m - \textsc{Independent Set} \ge 3m - (0.875+\varepsilon)m = (2.125-\varepsilon)m$

   (c) $\mathrm{OPT}_{\textsc{Clique}}(G) \le (0.875+\varepsilon)m$

4. Run the algorithm on G to produce number z.

5. (This is commentary, not part of the algorithm).

   Note the following two cases.

   (1) $\mathrm{OPT}_{\textsc{Max 3SAT}}(\varphi) \ge (1-\varepsilon)m$ so $\mathrm{OPT}_{\textsc{Vertex Cover}}(G) = (2+\varepsilon)m$. Hence $z \le (\alpha-\delta)(2+\varepsilon)m$.

(2) $\text{OPT}_{\text{MAX 3SAT}}(\varphi) \le (0.875+\varepsilon)m$ so $\text{OPT}_{\text{VERTEX COVER}}(G) = (2.125-\varepsilon)m$. Hence $z \ge (2.125-\varepsilon)m$.

To make these two cases disjoint we need

$$(\alpha - \delta)(2+\varepsilon) < (2.124 - \varepsilon)$$

$$\alpha - \delta < \frac{2.214 - \varepsilon}{2+\varepsilon}$$

We want to make α as large as possible so take $\alpha = \frac{2.214}{2} = 1.107$. Given δ we can then pick ε such that the inequality above holds.

6. If $z \le (\alpha-\delta)(2+\varepsilon)m$ then output $\text{OPT}(\varphi) \ge (1-\varepsilon)m$.

   If $z \ge (2.125-\varepsilon)$ then output $\text{OPT}(\varphi) \le (0.875+\varepsilon)m$.

□

Exercise 9.38. In the proofs of Theorems 9.28 and 9.34 we used the Gap lemmas (Lemmas 9.14,9.16) which have as a premise a reduction from an *NP-hard set A* to a GAP problem. In the proof of Theorem 9.37 we directly reduced a GAP problem to a GAP problem. Devise and prove Gap Lemmas that use a reduction from a GAP problem to a GAP problem. Use those lemmas to streamline the proof of Theorem 9.37.

Are better lower bounds for approximating VERTEX COVER known? Yes. We state two results.

Theorem 9.39. *Let $\delta > 0$.*

1. *(Hastad [Has01]) If there is an algorithm that, on input a graph G, returns a number $\le (1.166\ldots - \delta)OPT(G)$, then $P = NP$ (the number is actually $\frac{7}{6}$).*

2. *(Dinur & Safra [DS05]) If there is an algorithm that, on input a graph G, returns a number $\le (1.3606\ldots - \delta)OPT(G)$, then $P = NP$ (the number is actually $10\sqrt{5}-21$). Dinur & Safra [DS05] showed this.*

Are better lower bounds for approximating VERTEX COVER known? There are no NP-hardness result known. However, in Chapter 11 we will state the *Unique Games Conjecture* from which several lower bounds can be proven, including a lower bound of $2-\delta$ for approximating VERTEX COVER.

9.11 MAX REP and MIN REP are Hard to Approximate

We define two gap problems, state that they are NP-hard, and then use them to show lower bounds on other problems. The problems are *not* natural; they are a means to an end.

Definition 9.40. Let $G = (A, B, E)$ be a bipartite graph. We assume the following.

1. $|A| = |B| = n$.

2. There is a partition of A into sets $A_1, \ldots, A_k$ where $|A_i| = \frac{n}{k}$.

3. There is a partition of B into sets $B_1, \ldots, B_k$ where $|B_i| = \frac{n}{k}$.

4. We create a new bipartite graph as follows.

   (a) The vertices on the left are the A_i's.

   (b) The vertices on the right are the B_i's.

   (c) There is an edge from A_i to B_j if there exists $a \in A_i$ and $b \in B_j$ such that $(a, b) \in E$ (the original edges). These new edges are called ***superedges***.

5. A ***label cover*** is two subsets $A' \subseteq A$ and $B' \subseteq B$.

6. Given a label cover we say that a superedge (A_i, B_j) is ***covered*** if there exists $a \in A_i \cap A'$, $b \in B_j \cap B'$ such that $(a, b) \in E$. Notice that so far we have not demanded anything of our label covering.

ε-GAP-MAX REP.
Instance: A bipartite $G = (A, B, E)$ as in Definition 9.40.
Question: We only look at label covers which take exactly one element from each A_i and each B_j. We are promised that one of the following occurs.

- There is such a label covering which covers all superedges.
- Every such label covering covers at most an ε fraction of the superedges.

The question is to determine which case happens.
Note: It is called MAX REP since we view the element from A_i as a *representative*.

Raz [Raz98] showed the following. The proof uses the PCP machinery and is omitted.

Theorem 9.41. *Let $0 < \varepsilon < 1$.*

1. *If ε-GAP-MAX REP is in P then P = NP.*

2. *If there is an algorithm that on input an instance G of MIN REP returns a number $\geq \varepsilon OPT(G)$ then P = NP. (This follows from Part 1 and Lemma 9.14.)*

3. *If $\frac{1}{2^{\log^{1-\varepsilon}}}$-GAP-MAX REP is in P then NP $\subseteq$ DTIME$(n^{polylog(n)})$.*

4. *If there is an algorithm that on input an instance G of MIN REP returns a number $\geq \frac{1}{2^{\log^{1-\varepsilon}}} OPT(G)$ then NP $\subseteq$ DTIME$(n^{polylog(n)})$. (This follows from Part 3 and a variant of Lemma 9.14.)*

We define the dual problem, MIN REP.

ε-GAP-MIN REP.
Instance: A bipartite $G = (A, B, E)$ as in Definition 9.40, and ε.
Question: We only look at label covers which cover every superedge. We are promised that one of the following occurs. Let S be the number of superedges.

- There is label covering of size $2S$ that covers every superedge (this is optimal).
- Every label covering that covers every superedge has at least $\frac{1}{\varepsilon}$ nodes.

The question is to determine which case happens.
Note: Its called MIN REP since each superedge is covered, so each one has a representative. Or maybe just to look like the dual to MAX REP.

Theorem 9.42. *Let $0 < \varepsilon < 1$.*

1. *If ε-GAP-MIN REP is in P then P = NP.*
2. *If there is an algorithm that on input an instance G of MIN REP returns a number $\le \frac{1}{\varepsilon}OPT(G)$ then P = NP. (This follows from Part 1 and Lemma 9.16.)*
3. *If $\frac{1}{2^{\log^{1-\varepsilon}}}$-GAP-MIN REP is in P then $NP \subseteq DTIME(n^{polylog(n)})$.*
4. *If there is an algorithm that on input an instance G of MIN REP returns a number $\le 2^{\log^{1-\varepsilon}} OPT(G)$ then $NP \subseteq DTIME(n^{polylog(n)})$. (This follows from Part 3 and a variant of Lemma 9.16.)*

9.12 A Reduction from MIN REP

We show that the Directed Steiner Forest problem is hard by reducing MIN REP to it.

DIRECTED STEINER FOREST
Instance: A weighted directed graph G and a set of ordered pairs of vertices $\{(a_i, b_i)\}$.
Question: Find a subgraph G that has, for all i, a path from a_i to b_i of minimal weight. Note that this subset will be a forest.

Theorem 9.43. *Let $c \ge 1$. If there is an algorithm that, on input an instance of DIRECTED STEINER FOREST, outputs a number that is $\le c \times OPT(G)$, then P = NP.*

Proof. Let ε be such that $\frac{1}{\varepsilon} = c$. We give a gap reduction from ε-GAP-MIN REP to DIRECTED STEINER FOREST and then apply a variant of Lemma 9.16 from Exercise 9.38

1. Input is a bipartite graph $G = (A, B, E)$, $|A| = |B| = n$, a partition of A into sets $A_1, \ldots, A_k$ where $|A_i| = \frac{n}{k}$, and a partition of B into sets $B_1, \ldots, B_k$ where $|B_i| = \frac{n}{k}$. Let S be the number of superedges. We are told that either (1) there is a label cover of size $2S$ that covers all superedges, or (2) any label cover that covers all the superedges has $\ge \frac{1}{\varepsilon}2S$ vertices.
2. Create a directed graph G' as follows.

   (a) All of the vertices and edges of G are present. All of the edges have weight 0.
   (b) For each A_i there is a new vertex a_i. For all $v \in A_i$ there is an edge (a_i, v) of weight 1.
   (c) For each B_i there is a new vertex b_i. For all $v \in B_i$ there is an edge (v, b_i) of weight 1.
   (d) There is also an edge from a_i to b_i of weight w to be determined later.
   (e) The set of ordered pairs is all $\{(a_i, b_j) \mid (A_i, B_j) \text{ is a superedge}\}$.

3. (This is not part of the algorithm. This is commentary.) Let S be the number of superedges. It is easy to see that there is a label cover of size L if and only if there is a Directed Steiner Forest of weight L. Hence

   - If there is a label cover of size $2S$ that covers all the superedges then there is a Directed Steiner Forest of weight $2S$ which implies that there is a DSF of weight $2S$.
   - If the minimum sized label cover is of size $\ge \frac{1}{\varepsilon}2S$ then the optimal DSF has weight $\ge \frac{1}{\varepsilon}2S$.

4. Output the instance of Min Rep that you created.

$\square$

Chapter 10

Inapproximability

10.1 Introduction

In Chapter 9 we stated the following:

Theorem 10.1. *Assuming P ≠ NP:*

1. *TSP ∉ Poly-APX.*
2. *Clique ∈ Poly-APX − Log-APX.*
3. *Set Cover ∈ Log-APX − APX.*
4. *Max 3SAT ∈ APX − PTAS.*

We will use Max 3SAT and reductions to show many other problems are not in PTAS (unless P = NP). The other problems (TSP, Clique, Set Cover) do not seem to be useful to show that problems are not in some approximation classes (Poly-APX, Log-APX,APX).

Convention 10.2. For the rest of this chapter, "problem" means ***NPO problem***. When A and B are mentioned they are NPO problems. We will often say whether A is a min problem or a max problem.

Notation 10.3. Recall that if A is an NPO then the cost (if it's a min problem) or benefit (if it's a max problem) of a solution can be computed in polynomial time. If the problem is understood, x is an instance, and y is a solution, then benefit(x, y) is the benefit and cost(x, y) is the cost. For example, if the problem is Clique then the instance is a graph G, the solution is a clique C, and benefit(G, C) is the size of C.

10.2 Lower Bounds on Approximability

Everything in this section is due to Papadimitriou & Yannakakis [PY91], though with a very different outlook since Theorem 10.1 was not known when they wrote their paper.

They showed the following:

Theorem 10.4. *If* Max *3SAT $\in$ PTAS then every problem in APX has a PTAS.*

Note that Max 3SAT $\in$ APX. They called problems like Max 3SAT ***APX-complete***. They then showed many problems were APX-complete.

We contrast what they could conclude with what we can conclude. Let X be a problem in APX that we wish to show is likely not in PTAS.

1. They showed that there is an approximation preserving reduction (to be defined) from Max 3SAT to X. Hence, *if X has a PTAS then PTAS = APX*. Therefore X is unlikely to have a PTAS.

2. We use the same reductions they did; however, armed with Theorem 9.34, we can say *if X has a PTAS then P = NP*.

We will use their reductions; however, we will not need the notion of APX-complete since we have Theorem 9.34. We may still use the terminology for convenience.

We define the appropriate reductions, which are more complicated than the reduction $\le_p$.

Definition 10.5. Let A and B be 2 optimization problems. An ***approximation preserving reduction (APR)*** from A to B is a pair of polynomial-time functions (f, g) such that the following holds:

- If x is an instance of A then $f(x)$ is an instance of B.

- If y is a solution for $f(x)$ then $g(x, y)$ is a solution for x. We will later define types of reductions where a good solution for B maps to a good solution for A, for some notion of "good". Since we are only interested in good (whatever that might mean) solutions we may restrict g to solutions y that do not have an obvious improvement. For example, if the problem is MaxSAT (maximize the number of clauses satisfied), we may assume that any variable that appears without negations is set to TRUE. (Formally g is a function of x, x', and y' but we never specify it as such.)

- We usually do not use the notation "f and g". Instead (1) x will be an instance of A, x' will be the instance of B that x maps to ($f(x) = x'$), and (2) y' will be the solution to x', and y will be the solution of x that y' maps to ($g(y') = y$).

- Note that we have not specified how this reduction preserves approximation. And we won't. The name is not quite right. We will define reductions that begin "Blah is an approximation preserving reduction that also has property Blah Blah".

The idea here is that f transforms instances of A into instances of B while g transforms solutions for B instances into (in some sense equally good) solutions for the A instance.

Let's refine the above idea further:

Definition 10.6.

1. A ***PTAS reduction*** is an approximability preserving reduction satisfying the following additional constraint:

   for any $\varepsilon > 0$, there is a $\delta = \delta(\varepsilon) > 0$ such that, if y' is a $(1+\delta(\varepsilon))$-approximation to B, then y is a $(1+\varepsilon)$-approximation to A. Note that here we allow the f and g functions to depend on ε. We will assume that the function δ is monotone increasing and goes to infinity.

2. A ***strict reduction*** is an APX reduction where $\delta(\varepsilon) = \varepsilon$. This is a very strong concept of reduction.

Definition 10.7.

1. An ***APX reduction*** is a PTAS-reduction for which the δ function is linear in ε. This is a convenient reduction to use because if $B \in O(f)$-APX, then $A \in O(f)$-APX. For example, this stronger type of reduction would be useful for discussing log approximations.

Papadimitriou and Yannakakis defined APX-hard and APX-complete. We will give a different definition which is equivalent and makes more sense given that we have Theorem 9.34.

Their definition:

Definition 10.8.

1. A problem B is ***APX-hard*** if, for all $A \in$ APX, there is an APX-reduction from A to B.

2. A problem B is ***APX-complete*** if B is APX-hard and $B \in$ APX.

Our definition:

Definition 10.9.

1. A problem B is ***APX-hard*** if there is an APX-reduction from Max 3SAT to B.

2. A problem B is ***APX-complete*** if B is APX-hard and $B \in$ APX.

Henceforth we use our definition of APX-complete.

Note: The equivalence of these 2 definitions is not obvious. It depends on the theorem (due to Papadimitriou and Yannakakis) that Max 3SAT is APX-complete using Definition 10.8. We will not prove their theorem, nor do we need the equivalence.

The following follows from Theorem 9.34 and our definition of APX-complete.

Theorem 10.10. *If B is APX-complete then:*

1. *$B \in$ APX.*

2. *If $B \in$ PTAS then $P = NP$.*

Exercise 10.11. Assume there is a PTAS reduction from A to B.

1. Prove that if $B \in$ PTAS then $A \in$ PTAS. We will use the contrapositive form: if $A \notin$ PTAS then $B \notin$ PTAS.

2. Prove that if $B \in$ APX then $A \in$ APX. We will use the contrapositive form: if $A \notin$ APX then $B \notin$ APX.

3. Assume A is reducible to B with a strict reduction. Also assume that A and B are both MAX-problems (a similar theorem holds if they are Min-problems). Let $\varepsilon > 0$. Prove that if B has a $(1 + \varepsilon)$-approximation then A has a $(1 + \varepsilon)$-approximation. We will use the contrapositive form: If A does not have a $(1 + \varepsilon)$-approximation then B does not have a $(1 + \varepsilon)$-approximation.

PTAS reductions are all a bit awkward to work with directly because all of the approximability results use a multiplicative factor. In practice, people use L-reductions.

Definition 10.12. An L-reduction from A to B is an approximation preserving reduction which satisfies the following 2 properties:

- $\mathrm{OPT}_B(x') = O(\mathrm{OPT}_A(x))$
- $|\mathrm{cost}_A(y) - \mathrm{OPT}_A(x)| = O(|\mathrm{cost}_B(y') - \mathrm{OPT}_B(x')|)$

We denote this reduction by $A \leq_L B$. (The L stands for "Linear".)

L reductions are stronger than APX reductions (as we will show) so the existence of an L reduction implies the existence of an APX reduction which implies the existence of a PTAS reduction. Note that L reductions are not stronger than strict reductions.

Theorem 10.13. *If $A \leq_L B$ then that same reduction is an APX-reduction.*

Proof. Let's prove that this is an APX reduction. We will do this in the minimization case.

$\mathrm{OPT}_B(x') = O(\mathrm{OPT}_A(x))$ so there exists a constant $\alpha > 0$ such that

$$\mathrm{OPT}_B(x') \leq \alpha \mathrm{OPT}_A(x).$$

$|\mathrm{cost}_A(y) - \mathrm{OPT}_A(x)| = O(|\mathrm{cost}_B(y') - \mathrm{OPT}_B(x')|)$, so because this is a minimization problem, there exists a positive constant β such that

$$\mathrm{cost}_A(y) - \mathrm{OPT}_A(x) \leq \beta(\mathrm{cost}_B(y') - \mathrm{OPT}_B(x')).$$

To show that the L-reduction is an APX-reduction we need to show that there is a constant γ such that if

$$\mathrm{cost}(x') \leq (1 + \gamma\varepsilon)\mathrm{OPT}_B(x')$$

then

$$\mathrm{cost}_A(x) \leq (1 + \varepsilon)\mathrm{OPT}_A(x).$$

We will derive γ. Let $\delta = \gamma\varepsilon$.

Assume that $\mathrm{cost}_B(y') \leq (1 + \delta)\mathrm{OPT}_B(x')$. We want to obtain

$$\mathrm{cost}_A(y) \leq (1 + \varepsilon)\mathrm{OPT}_A(x).$$

We know that

$$\text{cost}_A(y) \le \text{OPT}_A(x) + \beta\text{cost}_B(y') - \beta\text{OPT}_B(x').$$

We factor out $\text{OPT}_A(x)$ to get

$$\text{cost}_A(y) \le \text{OPT}_A(x)\left(1 + \frac{\beta\text{cost}_B(y')}{\text{OPT}_A(x)} - \frac{\beta\text{OPT}_B(x')}{\text{OPT}_A(x)}\right).$$

Since

$$\frac{1}{\text{OPT}_A(x)} \le \frac{\alpha}{\text{OPT}_B(x')}$$

we have

$$\text{cost}_A(y) \le \text{OPT}_A(x)\left(1 + \frac{\alpha\beta\text{cost}_B(y')}{\text{OPT}_B(x')} - \frac{\alpha\beta\text{OPT}_B(x')}{\text{OPT}_B(x')}\right).$$

Hence

$$\text{cost}_A(y) \le \text{OPT}_A(x)\left(1 + \frac{\alpha\beta\text{cost}_B(y')}{\text{OPT}_B(x')} - \alpha\beta\right).$$

Since $\text{cost}_B(y') \le (1+\delta)\text{OPT}_B(x')$, we have $\frac{\text{cost}_B(y')}{\text{OPT}_B(x')} \le 1+\delta$, so

$$\text{cost}_A(y) \le OPT_A(x)(1 + \alpha\beta(1+\delta) - \alpha\beta) = \text{OPT}_A(x)(1 + \alpha\beta\delta).$$

Hence

$$\text{cost}_A(y) \le \text{OPT}_A(x)(1 + \alpha\beta\gamma\varepsilon).$$

It suffices to take $\gamma = \frac{1}{\alpha\beta}$. □

Exercise 10.14.

1. Show that if $A \le_L B$ and $B \le_L C$ then $A \le_L C$.

2. Show that if $A \le_L B$ with a strict reduction and $B \le_L C$ with a strict reduction, then $A \le_L C$ with a strict reduction.

10.3 Max 3SAT and Its Variants

The results in this section are essentially due to Papadimitriou & Yannakakis [PY91]. We define several SAT problems. We include Max 3SAT since the other SAT problems are variant of it.

Max 3SAT
Instance: A formula φ where every clause has ≤ 3 literals.
Question: What is the max number of clauses that can be satisfied by an assignment?

Max 3SAT-Ea
Instance: A formula φ where every clause has ≤ 3 literals and every variable occurs $\leq a$ times.
Question: What is the max number of clauses that can be satisfied by an assignment? (This problem is not interesting in its own right; however, it will be used to show several problems on graphs of bounded degree are hard to approximate.)

Max 2SAT
Instance: A formula φ where every clause has ≤ 2 literals.
Question: What is the max number of clauses that can be satisfied by an assignment?

Max NAE-3SAT
Instance: A formula φ where every clause has ≤ 3 literals.
Question: What is the max number of clauses that can be satisfied by an assignment with the extra condition that no clause has all of its literals TRUE. NAE stands for Not-All-Equal. (This problem is not interesting in its own right; however, it will be used to show MaxCut is hard to approximate.)

We will need the following exercise.

Exercise 10.15. Let φ be a formula with C clauses where every clause has ≤ 3 literals. Prove that the max number of clauses that can be satisfied is $\geq \Omega(C)$ (and hence clearly $\Theta(C)$).
Hint: Use Theorem 9.34.1.

10.4 Reductions from Max 3SAT and Its Variants

We first show a reduction from Max 3SAT to Max 3SAT-E3 which is *not* a PTAS reduction to motivate that we need a more sophisticated reduction.

1. Input $\varphi(x_1, \ldots, x_n)$. Assume φ has m clauses.

2. For each variable x that occurs ≥ 4 times do the following:

   (a) Let k be the number of times x occurs. Introduce new variables $z_1, \ldots, z_k$.

   (b) Replace the k occurrences of x with $z_1, \ldots, z_k$.

   (c) Add the clauses $(z_1 \rightarrow z_2)$, $(z_2 \rightarrow z_3)$, …, $(z_{L-1} \rightarrow z_k)$, $(z_k \rightarrow z_1)$. These clauses are an attempt to force all of the z_i to have the same truth value. If this was a decision-problem reduction then the attempt would succeed. (Formally we would use $z_1 \vee \neg z_2$ for $z_1 \rightarrow z_2$.)

Let φ' be the new formula.

Clearly $\varphi \in$ SAT if and only if $\varphi' \in$ SAT. But we need more. We need to be able to take an assignment that satisfies many clauses of φ' and map it to an assignment that satisfies many clauses of φ.

We give an example to show why the reduction does not work.

$$\varphi(x) = (x \vee x \vee x) \wedge \cdots \wedge (x \vee x \vee x) \wedge \cdots \wedge (\neg x \vee \neg x \vee \neg x) \wedge \cdots \wedge (\neg x \vee \neg x \vee \neg x)$$

where there are m $(x \vee x \vee x)$ clauses and m $(\neg x \vee \neg x \vee \neg x)$ clauses. Note that the Max 3SAT value is m.

φ' will be

$$(z_1 \vee z_2 \vee z_3) \wedge \cdots \wedge (z_{3m-2} \vee z_{3m-1} \vee z_{3m}) \wedge$$

$$(\neg z_{3m+1} \vee \neg z_{3m+2} \vee \neg z_{3m+3}) \wedge \cdots \wedge (\neg z_{6m-2} \vee \neg z_{6m-1} \vee \neg z_{6m}) \wedge$$

$$(z_1 \rightarrow z_2) \wedge \cdots \wedge (z_{6m-1} \rightarrow z_{6m}) \wedge (z_{6m} \rightarrow z_1)$$

If we set $z_1, \ldots, z_{3m}$ to TRUE and $z_{3m+1}, \ldots, z_{6m}$ to FALSE then we satisfy every single clause except $z_{3m} \rightarrow z_{3m+1}$. That's $2m - 1$ clauses. More to the point, there is no useful way to take that assignment and map it to an assignment for φ that satisfies many clauses. Hence if we want to reduce Max 3SAT to Max 3SAT-E3 we will need to use a more intricate reduction.

In the failed reduction we used a ***cycle*** to connect the different variables that are supposed to all have the same truth value. In the correct reduction we will use a more complicated graph.

Recall that the ***degree*** of a graph is the maximum degree of the vertices.

Definition 10.16. Let $d \in \mathbb{N}$. A ***d-expander graph*** is a graph $G = (V, E)$ where (1) every vertex has degree d and (2) for every partition $V = V_1 \cup V_2$ the number of edges from vertices in V_1 to vertices in V_2 is $\geq \min(|V_1|, |V_2|)$.

An expander graph is something like a sparse clique in the sense that there are a lot of connections across every partition, but there are only a bounded number of edges for each vertex.

Note: There are many different notions of expander graph that are *not* equivalent. Hence you may come across a different definition in the literature. They all involve graphs which somehow combine high connectivity with a small number of edges. We will only use the definition above.

The following is well known, and also in the paper by Papadimitriou & Yannakakis [PY91](Page 432).

Exercise 10.17. Prove that, for all $k \equiv 0 \pmod 2$, there exists a 3-expander graph on k vertices.

Theorem 10.18.

1. *Max 3SAT $\leq_L$ Max 3SAT-E7.*
2. *Max 3SAT-E7 $\leq_L$ Max 3SAT-E3.*
3. *Max 3SAT $\leq_L$ Max 3SAT-E3 (this follows from the parts 1 and 2, and Exercise 10.14).*

4. *Assume P $\neq$ NP. Let $a \geq 3$. Then* Max *3SAT-Ea $\notin$ PTAS (this follows from part 3 and Theorem 9.34.1).*

5. Max *3SAT-E3 is APX-complete (this follows from part 4 and Theorem 9.34.1).*

Proof.
1) Here is the reduction:

1. Input $\varphi(x_1, \ldots, x_n)$. We will assume every variable occurs an even number of times. The modifications needed if a variable occurs an odd number of times are left to the reader.

2. For each variable x that occurs ≥ 8 times do the following:

   (a) Let k be the number of times x occurs. Introduce new variables $z_1, \ldots, z_k$.

   (b) Replace the k occurrences of x with $z_1, \ldots, z_k$.

   (c) Let G be a 3-expander graph on k vertices $\{1, \ldots, k\}$ (such exists by Exercise 10.17). For every edge $\{i, j\}$ add the clauses $(z_i \rightarrow z_j)$ and $(z_j \rightarrow z_i)$. (Formally we would use $z_i \vee \neg z_j$ for $z_i \rightarrow z_j$.) Note that z_i will occur 7 times in φ': (1) once in the place it replaces x in the original formula, (2) 3 times in clauses of the form $z_i \vee \neg z_j$, (3) 3 times in clauses of the form $z_j \vee \neg z_i$. The last 2 come from G having degree 3.

   Let the new formula be φ'.

How many times does a variable z occur in φ'? If z occurs $k \leq 7$ times in φ then it will occur $k \leq 7$ times in φ'. If z occurred ≥ 8 times in φ then it will occur 7 times in φ' as noted above.

We show how to go from an assignment for φ' to an assignment for φ. Let $\vec{b}'$ be an assignment for φ'. Let x be a variable in φ that occurred ≥ 8 times in φ. We associated to x a set of variables that we want to all be assigned the same truth value. Let Z_{TRUE} be the subset of those variables that are assigned TRUE, and Z_{FALSE} be the subset of those variables that are assigned FALSE. We will show that *more* clauses of φ' can be satisfied by assigning all of them the same. Recall that we have many clauses relating the variables associated to x via an expander graph.

Assume $|Z_{\text{TRUE}}| > |Z_{\text{FALSE}}|$ (the other case is similar). The expander graph has $\geq |Z_{\text{TRUE}}|$ edges from Z_{TRUE} to Z_{false}. For every edge (i, j) where $i \in Z_{\text{TRUE}}$ and $j \in Z_{\text{FALSE}}$, there are *two* clauses, $z_i \rightarrow z_j$ and $z_j \rightarrow z_i$. Hence there are $\geq 2|Z_{\text{TRUE}}|$ clauses that connect a variable from Z_{TRUE} to a variable from Z_{FALSE}. If we change the assignment of everything in Z_{FALSE} to TRUE then we make $\geq 2|Z_{\text{TRUE}}|$ clauses TRUE. How many clauses are now FALSE? Each variable in Z_{TRUE} appeared at most once in the non-expander-part of the formula. Hence $\leq |Z_{\text{TRUE}}|$ of the clauses are now FALSE. So the net gain is $\geq |Z_{\text{TRUE}}| > 0$. Recall that we began with a variable x in φ which was associated to $Z_{\text{FALSE}} \cup Z_{\text{TRUE}}$ with $|Z_{\text{TRUE}}| > |Z_{\text{FALSE}}|$. We now know that to maximize the number of clauses satisfied, $Z_{\text{FALSE}} = \emptyset$. Hence all of the variables associated to x are assigned the same value. *This is the reason we use expander graphs rather than cycles!*

We recap and get back to our issue. We can assume that $\vec{b}'$ assigns variables as we intended. Hence it is easy to map to $\vec{b}$ which only assigns the original variables of φ in the obvious way. We now show that we have an L-reduction.

The expander graph for x_i has k_i vertices and $3k_i/2$ edges. Each edge corresponds to 2 clauses that will both be set to TRUE. Hence

$$\text{benefit}(\varphi', \vec{b}') = \text{benefit}(\varphi, \vec{b}) + 3\sum_{i=1}^{n} k_i.$$

Hence

$$\text{OPT}(\varphi') = \text{OPT}(\varphi) + 3\sum_{i=1}^{n} k_i.$$

If we subtract we get

$$|\text{OPT}(\varphi') - \text{benefit}(\varphi', \vec{b}')| = |\text{OPT}(\varphi) - \text{benefit}(\varphi, \vec{b})|$$

Hence we have the second condition for being an L-reduction. Note that φ' has $m+O(\sum_{i=1}^{n} k_i) = O(m)$ clauses and φ has $O(m)$ clauses. Hence, by Exercise 10.15, $\text{OPT}(\varphi) = O(\text{OPT}(\varphi'))$.

2) Given a formula φ where each clause has ≤ 3 literals, and each variable occurs ≤ 7 times, φ' is formed by using the cycle-construction presented earlier in this section (the construction that did not work). Details are left to the reader. □

Exercise 10.19. Complete the proof of Theorem 10.18.2.

A stronger result is known [Tov84], though we will not need it or prove it:

Theorem 10.20. *Let* Max *3SAT-E5 be (just for this theorem) the function that takes a formula where (1) every clause has* exactly *3 literals, and (2) every variable occurs* exactly *5 times, and returns (as usual) the assignment that maximizes the number of clauses satisfied. Then* Max *3SAT $\leq_L$* Max *3SAT-E5.*

10.5 Formulas and Graphs and Formulas and Graphs

In this section we start with Max 3SAT, which we already know has no PTAS (from Theorem 9.34) and form a chain of reductions through several graph and formula problems. It is of interest that we start with a formula problem, then get some graph problems, then get back to formulas, then graphs again.

Definition 10.21.

1. Independent Set-B-a is the function that, given a graph G with degree $\leq a$, returns the size of the maximum independent set of G.

2. Vertex Cov-a is the function that, given a graph G with degree $\leq a$, returns the size of the minimum vertex cover of G.

3. Dom Set-a is the function that, given a graph G with degree $\leq a$, returns the size of the minimum dominating set of G.

Theorem 10.22. *Within this theorem a statement of the form $X \notin$ PTAS is contingent on $P \neq NP$.*

1. *MAX 3SAT-E3 $\le_L$ INDEPENDENT SET-B-4 (the reduction is strict).*

2. *For all $\Delta \ge 4$, INDEPENDENT SET-B-Δ is APX-complete. This follows from Part 1 and the result of Halldórsson-Radhakrishnan [HR97] that INDEPENDENT SET-B-Δ has a $\frac{\Delta+2}{3}$-approximation via a greedy algorithm. (Recall that this means there is an algorithm that returns an independent set of size $\ge \frac{3}{\Delta+2}$OPT.)*

3. *INDEPENDENT SET-B-4 $\le_L$ VERTEX COV-4 (the reduction is strict).*

4. *For all $\Delta \ge 4$, VERTEX COV-Δ and VERTEX COVER are both in APX-complete. This follows from Part 3 and the result shown independently by Gavril & Yannakakis (see page 190 of Garey & Johnson [GJ79]) that VERTEX COVER (unbounded degree) has a 2-approximation.*

5. *VERTEX COV-4 $\le_L$ DOM SET-4 (the reduction is strict).*

6. *For all $\Delta \ge 4$, DOM SET-Δ is APX-complete. This follows from Part 5 and the result of Parekh [Par91] that DOM SET-Δ has a $O(\log \Delta)$-approximation by a greedy algorithm.*

7. *INDEPENDENT SET-B-4 $\le_L$ MAX 2SAT.*

8. *MAX 2SAT is APX-complete. This follows from Part 7 and a simple approximation algorithm for MAX 2SAT similar to the proof of Theorem 9.34.1.*

9. *MAX 2SAT $\le_L$ MAX NAE-3SAT (this reduction is strict).*

10. *MAX NAE-3SAT $\in$ APX $-$ PTAS. The lower bound follows from Part 9. We leave as an exercise to show that MAX NAE-3SAT $\in$ APX.*

11. *MAX NAE-3SAT $\le_L$ MAXCUT. (MAXCUT was defined in Section 2.13.)*

12. *MAXCUT $\in$ APX $-$ PTAS. Part 11 shows the lower bound. There is a simple randomized 2-approximation algorithm (recall that this means the algorithm returns a number $\ge \frac{1}{2}$OPT) that can be derandomized (see the Wikipedia page on Max Cut).*

Proof.
1) We give the reduction from MAX 3SAT-E3 to INDEPENDENT SET-B-4. This is the ordinary reduction from 3SAT to INDEPENDENT SET; however, we present it for completeness and to see how the bound on the number of occurrences of a variable gives a bound on the degree of the graph.

1. Input a formula φ where every clause has ≤ 3 literals and every variable appears ≤ 3 times.

2. We construct a graph G.

   (a) For every occurrence of a literal there is a vertex.

   (b) For every clause of the form $(L_1 \vee L_2 \vee L_3)$ where the L_i's are literals, put in edges between each pair of L_i's. For every clause of the form $(L_1 \vee L_2)$ where the L_i's are literals, put in edges between L_1 and L_2.

(c) For all variables x, if x is in one clause and $\overline{x}$ is in another clause, put an edge between them.

Let v be a vertex. It corresponds to variable x in clause C. There will be 2 edges from v to the other vertices in clause C. There will be ≤ 2 edges to other clauses since they goto vertices associated with occurrences of $\neg x$, and there are at most 2 of those. Hence the degree of v is ≤ 4. The same reasoning holds when v is associated to $\neg x$.

We leave it to the reader to prove this is a strict reduction.

3) If $G = (V, E)$ is a graph and U is an independent set, then $V - U$ is a vertex cover. Hence to show Independent Set-B-4 $\leq_L$ Vertex Cov-4 do the following: (1) just map G to G, and (2) map a vertex cover for G to its complement to get an independent set for G. The proof that this is a strict L-reduction is trivial.

5) Vertex Cov-4 $\leq_L$ Dom Set-4 by the following reduction: map a graph G to the graph obtained by, for every edge (u, v), adding a vertex x that has an edge to u and to v. No other edges use x. We map a dominating set of G' to a vertex cover of G as follows:

1. Input G' and a dominating set D' for G'.
2. If one of the new vertices x, adjacent to both u, v, is in D then either (1) one of u, v is in D so x can be removed, or (2) neither of u, v is in D, so remove x and add (say) u.
3. The new set (which might not be a dominating set for G') is a vertex cover U for G, so output it.

Clearly benefit(D) = benefit(U), so OPT(D) = OPT(U). So we have a strong reduction.

7) Independent Set-B-4 $\leq_L$ Max 2SAT by the following reduction from graphs to formulas.

1. Input $G = (V, E)$, a graph of degree ≤ 4.
2. Form a formula φ as follows:
   (a) For every vertex v we have clause $\{v\}$.
   (b) For every edge (u, v) we have clause $\{\overline{u} \vee \overline{v}\}$.

We show how to map an assignment for φ to an independent set of G. Recall from the definition of reduction that we need only consider assignments that cannot be improved in an obvious way.

If the assignment makes some 2-clause $\overline{u} \vee \overline{v}$ false, then change the assignment to make u false. The number of clauses satisfied will not decrease. Hence we will only consider assignments where all of the 2-clauses are satisfied. Given such an assignment, we map it to the set of vertices associated to variables that are set to true. Because of the change we made to the assignment, this will correspond to an independent set.

We now show that this is an L-reduction. Any assignment for φ can be improved (or at least not made worse) by the procedure described above: make every clause $(\overline{u} \vee \overline{u})$ true. The remaining 1-clauses that are set to true must correspond to a maximum independent set in G.

Hence we can start with an assignment $\vec{b}$ for φ that makes every 2-clause TRUE. Map it to the set of vertices U corresponding to the 1-clauses being TRUE. Clearly those vertices form an independent set. Hence benefit$(\vec{b})$ = benefit$(U) + |E|$. Hence the second condition for being an L-reduction is satisfied.

Since G has degree ≤ 4, $|E| = O(|V|)$ and $\text{OPT}(G) = \Theta(|V|)$ (by a greedy algorithm). Since φ has $|V|$ 1-clauses, $\text{OPT}(\varphi) = \Theta(|V|)$. Since $\text{OPT}(G)$ and $\text{OPT}(\varphi)$ are both $\Theta(|V|)$, the first condition of being an L-reduction, $\text{OPT}(\varphi) = O(\text{OPT}(G))$, is satisfied.

9) MAX 2SAT $\leq_L$ MAX NAE-3SAT by the following reduction from formulas to formulas.

1. Input φ, a formula where every clause has ≤ 2 literals.

2. Let z be a new variable.

3. We form a formula φ' by adding $\vee z$ to all 2-clauses, and $\vee z \vee z$ to all 1-clauses.

Let $\vec{b}'$ be an assignment for φ' where m of the clauses are satisfied but no clause has all 3 literals set the same. If z is set to TRUE then the m satisfied clauses all have at least one literal set to FALSE. Hence if we flip the truth value of all the variables (note z is now FALSE) we still get an assignment where there are m clauses set to TRUE, and none of them have all literals set the same. We need only consider such assignments.

Since z is set to FALSE, the assignment, not including z, is an assignment for φ that makes m clauses TRUE. Hence benefit$(\varphi', \vec{b}')$ = benefit$(\varphi, \vec{b})$, so we have a strong reduction.

11) MAX NAE-3SAT $\leq_L$ MAXCUT by the following reduction from formulas to graphs.

1. Input φ, a formula in 3-CNF form. It has n variables $x_1, \ldots, x_n$. x_i appears k_i times.

2. We form the graph G as follows (G will actually be a multigraph).

   (a) For every variable x, let both x and $\overline{x}$ be vertices, and put k edges between them where k is the number of occurrences of x (see Figure 10.1, left side).

   (b) For every 3-clause $L_1 \vee L_2 \vee L_3$ put edges between L_1, L_2, and L_1, L_3, and L_2, L_3. Similar for 2-clauses (see Figure 10.1, right side). For 1-clauses no edges are added.

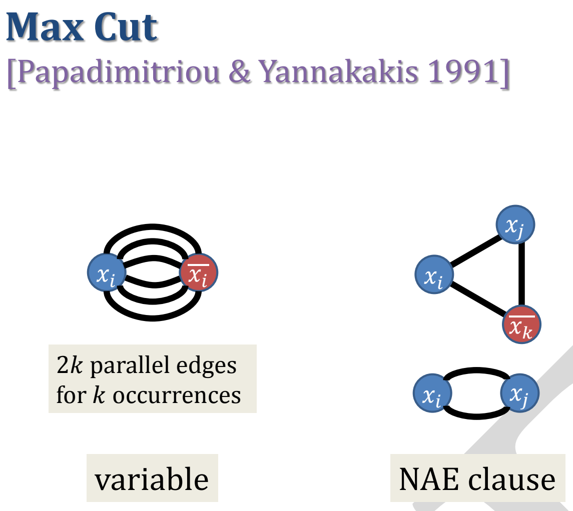


Figure 10.1: Max NAE-3SAT $\le_L$ MaxCut

We need to show how to map a partition of vertices (V_1, V_2) to an assignment $\vec{b}$. It is easy to see the only partitions worth considering have, for every variable x, x and $\neg x$ in different parts. Hence we can map (V_1, V_2) to the assignment that sets every literal in V_1 to TRUE. (We could have used V_2 but it won't matter which since Max NAE-3SAT treats TRUE and FALSE equally.)

Let (V_1, V_2) be a partition. The edges between x and $\neg x$ contribute $\sum_{i=1}^{n} k_i$ edges. Each other edge in the cut corresponds to a way to make a clause TRUE where not all of the literals have the same truth value. Hence benefit$(V_1, V_2) = \sum_{i=1}^{n} k_i + \text{benefit}(\vec{b})$, so $\text{OPT}(G) = \sum_{i=1}^{n} k_i + \text{OPT}(\varphi)$. From these 2 equations one can show that this is an L-reduction. □

Note: Theorem 10.22.2 Independent Set-B-4 has a 2-approximation, but (assuming P ≠ NP) does not have a PTAS. Does Independent Set-B-4 have a 1.9-approx? Berman & Karpinski [BK99] have more refined results on Independent Set-B-Δ, Vertex Cov-Δ, and Max 2SAT. Trevisan et al. [TSSW00], Hastad [Has01], and [KKMO07] have more refined results on MaxCut.

For the next exercise we will consider the following variant of Set Cover.

Unique Set Cover
Instance: n and Sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.
Question: What is the smallest size of a subset of S_i's that covers all of the elements in $\{1, \ldots, n\}$ with every element of $\{1, \ldots, n\}$ in exactly one of the chosen S_i's.
,

Exercise 10.23. Show that MaxCut $\le_L$ Unique Set Cover. What can you conclude from this in terms of approximations for Unique Set Cover?

10.6 Edge Matching Approximation

We define a function version of Edge Matching.

Edge Matching
Instance: A set of unit squares with the edges colored, and a target rectangle RECT.
Question: How many squares can you pack into the rectangle such that all tiles sharing an edge have matching colors. (The colors are unary numbers)

Our goal is to show that EMP is APX-complete. We will use the following problem.

Max Ind Set on 3-regular, 3-edge colorable graphs
Instance: A 3-regular graph and a 3-coloring of the edges (no 2 incident edges are the same color).
Question: Find the largest independent set.

By Vizing's theorem [Viz64] a graph of degree Δ is $\Delta + 1$-edge colorable. Hence the edge-chromatic number is either Δ or $\Delta + 1$. Cai & Ellis [CE91] showed that determining the edge-chromatic number of a 3-regular graph is NP-complete. Hence the Max Ind Set on 3-regular, 3-edge colorable graphs problem comes with a lot of information. As such the following results of Chlebik & Chlebikova [CC03] are surprising:

Theorem 10.24. *The Max Ind Set on 3-regular, 3-edge colorable graphs problem is APX-complete.*

We omit the proof.

We need one more problem, a variant of 3-dimensional matching which we discussed in Section 6.3.2.

3D-Matching-2
Instance: Disjoint sets A, B, C with $|A| = |B| = |C| = n$, and $M \subseteq A \times B \times C$ such that every element of $A \cup B \cup C$ appears exactly twice in M (hence the 2 in 3D-Matching-2).
Question: The intuition is that some alien species has 3 sexes and we are trying to arrange n 3-person-marriages. The output is an $M' \subseteq M$ such that (a) all the triples in M' are disjoint (no polygamy) and (b) every element of $A \cup B \cup C$ is in some triple of M' (no unmarried people). In the function version of this problem we are trying to maximize the size of M' that satisfies (a) and (b).

Theorem 10.25. *There is a strict reduction from Max Ind Set on 3-regular, 3-edge colorable graphs to 3D-Matching-2.*

Proof sketch:

Let $G = (V, E)$ be a 3-regular graph. We are also given a 3-edge coloring of it with colors 1,2,3. Let:

1. A be the set of edges colored 1.
2. B be the set of edges colored 2.
3. C be the set of edges colored 3.

For every $v \in V$ we put the triple of edges incident to it into M.

We leave it to the reader to both finish the proof and show that the reduction goes backwards, so we have a strict reduction in both directions. ∎

Exercise 10.26. Finish the proof of Theorem 10.25.

We now get to our actual goal which is Edge Matching.

The first part of the next theorem is easy. The second part is due to Antoniadis & Lingas [AL10].

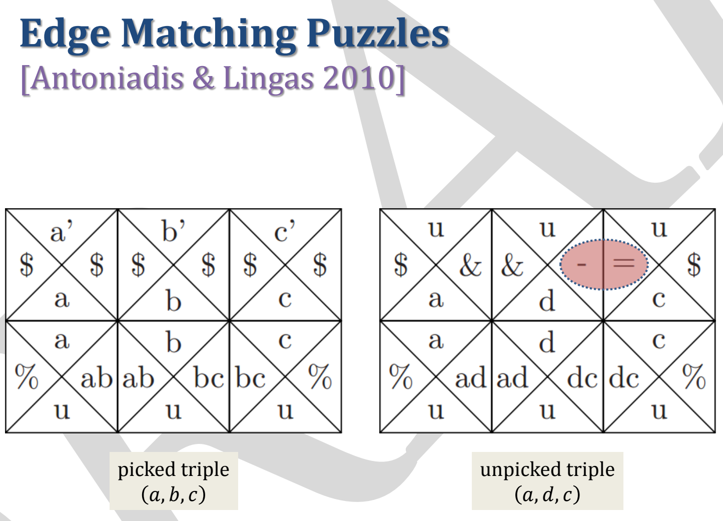
Theorem 10.27.

1. *EMP $\in$ APX. In particular there is an algorithm that returns $\geq \frac{1}{8}$OPT.*

2. *EMP is APX-complete.*

Proof sketch:
1) Here is the approximation algorithm: find the maximum number of pairs of tiles that share an edge. We leave it to the reader to show the number of tiles this algorithm returns is $\geq \frac{1}{8}$OPT.

2) We show the gadgets needed for the reduction EMP $\leq_L$ 3D-Matching-2.


Edge Matching Puzzles
[Antoniadis & Lingas 2010]

]

Figure 10.2: Gadgets for EMP $\leq_L$ 3D-Matching-2

Figure 10.2 shows gadgets for a picked triple and unpicked triple in an edge matching puzzle. The row of *u*s on the bottom is intended to match up against an un-pictured boundary structure in a way that forces the placing of the bottom row (in the sense that placing the bottom row in the intended way can only improve the number of matches).

Note that the tile marked with 2 dollar signs, a and a' is unique to the letter a, which only allows us to pick one triple containing a. We also need structures that enable us to dump the 'picked' top row if we do not pick the triple, or the 'unpicked' top row if we do pick the triple. The dump structures are listed, together with a full listing of tiles per 3DM element. We omit them here for brevity. ▮

Exercise 10.28. Complete the proof of Theorem 10.27.

What if the rectangle is $1 \times n$? Bosboom et al. [BDD+17] showed the following:

Theorem 10.29.

1. *There is an algorithm that packs* $\geq \frac{1}{2}OPT$.
2. *If there is an algorithm that packs* $\geq \frac{33519359}{33519360}OPT$ *then* $P = NP$ *(the faction is* ~ 0.9999999702*).*

The problem they reduce to the following gap version of Ham Cycle which was already known to be hard:

Given a graph G that you are promised either has a cycle on $n-1$ disjoint edges (so a Hamiltonian Cycle) or the max number of edges in a vertex-disjoint union of paths is at most $0.999999284(n-1)$ edges.

10.7 Other Complexity Classes for Approximations

APX-completeness is not the be-all and end-all of complexity in approximations. There exists a wealth of classes both above it and below it in complexity.

10.7.1 APX-Intermediate Problems

Let A be a min problem. Imagine that there is an algorithm that approximates A by outputting a number $\leq OPT_A(x) + 1$. Such a problem would be in APX since this is a constant approximation; however, this is actually better than APX. Does this approximation algorithm easily lead to a PTAS? We would need

$$\forall \varepsilon : \forall x : OPT_A(x) + 1 \leq (1+\varepsilon)OPT_A(x)$$

$$\forall \varepsilon : \forall x : 1 \leq \varepsilon \mathrm{OPT}_A(x)$$

which might not be the case.

So perhaps A is APX-complete.

We present three problems such that

1. There is a better-than-APX approximation (formally called an asymptotic PTAS but we won't be using that term).
2. If there currently no aPTAS.

For all three problems the third point is due to Crescenzi et al. [CKST99], the second point is folklore, and the first point we will provide a reference.

Bin Packing
Instance: A finite set of positive rationals U.
Question: Minimize the number of bins needed to partition U into sets whose sum is ≤ 1. Output the optimal number of bins. Rothvoß [Rot13b] showed there is an OPT+$O(\log(\text{OPT}) \log\log(\text{OPT})$-approximation. Karp & Karmarkar [KK82] had earlier results that were weaker but of the same flavor.

Minimize Max Degree of a spanning Tree
Instance: A weighted graph G.
Question: Find spanning tree with the smallest possible max degree. Only output the actual degree. Fürer & Raghavachari [FR92] have an OPT + 1 algorithm for this problem.

Edge Chromatic Number
Instance: A graph G.
Question: Find the minimum number of colors needed to color the edges so that no two incident edges are the same color. By Vizing's theorem [Viz64] a graph with maximum degree Δ has an $\Delta + 1$-edge coloring. Hence the we can always get an answer $\le \text{OPT}(G) + 1$. (Misra & Gries [MG92] obtained a constructive proof of Vizing's Theorem that yields a polynomial-time algorithm that finds the $\Delta + 1$-coloring. This is not needed for our purpose of just finding the edge-chromatic number.)

10.7.2 Log-APX-Completeness

Everything in this section is due to Escoffier & Paschos [EP06].

Combining Exercise 10.14 and 9.30.4 we easily have the following.

Theorem 10.30. *Let $A_1, \ldots, A_m$ be such that* Set Cover $\le_L A_1 \le_L \cdots \le_L A_m$. *Assume $P \neq NP$. There is a c such that A_m does not have a $c \ln n$-approximation.*

Theorem 10.31.

1. *There is a $(\ln \Delta + 2)$-approximation for* Dom Set *where Δ is the maximum degree.*

2. *Assume $P \neq NP$. For all $c < 1$ there is no $c \ln(\Delta)$-approximation for* Dom Set.

Proof.
1) A greedy algorithm where you always take the vertex of max degree yields a $(\ln \Delta + 2)$-approximation.

2) We show Set Cover $\le_L$ Dom Set.

1. Input n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$. Let $U = \{1, \ldots, n\}$.

2. Form a graph $G = (V, E)$ as follows:

   (a) There is a vertex for every element of $\{1, \ldots, n\}$ and for every S_i. Hence there are $n+m$ vertices. We call the vertices associated to $\{1, \ldots, n\}$ the U-vertices, and the vertices associated to the $S_1, \ldots, S_m$, the S-vertices.

   (b) All vertices representing the S_i are connected. This forms a clique of size n.

   (c) For all $i \in \{1, \ldots, n\}$ and all $j \in \{1, \ldots, m\}$ connect i to j if $i \in S_j$.

   (d) Note that $\Delta \le n$.

Let D be a dominating set. If there are any U-vertices in D then they can be replaced by the S-vertex they connect to. Hence we can assume that every dominating set consists only of S-vertices.

We map a dominating set to the S-sets that its S-vertices correspond to. The size of the dominating set is exactly the size of a covering. Hence this is a strict reduction.

Since the graph has $\Delta \le n$ it is easy to see that if there is a $c\ln(\Delta)$-approximation for Dom Set then there is a $c\ln(n)$-approximation for Set Cover, so P = NP. □

Exercise 10.32. Show that Dom Set $\le_L$ Set Cover.

We now show another inapproximability result. The ***15 puzzle*** is a classic puzzle from the 1870's (see the Wikipedia page on it). It can be generalized to the $n^2 - 1$ puzzle. Ratner & Warmuth [RW90] showed that solving the $n^2 - 1$ puzzle is NP-hard. It is not known if the function version (trying to minimize the number of moves) is in APX.

Călinescu et al. [CDP08] worked on a generalization of the $n^2 - 1$ puzzle:

Motion Planning
Instance: A graph $G = (V, E)$ with the vertex set split into 2 (possibly overlapping) sets V_1, V_2 of the same size. The elements of V_1 are called ***tokens*** and each one has a ***robot*** on it. The elements of V_2 are called ***targets***.
Question: A ***move*** is when a robot goes on a path with no other robots on it. Note that a robot may go quite far in one move. We want a final configuration where all the robots are on the vertices in V_2 (only one robot can fit on a vertex). We want to do this in the minimum number of moves.

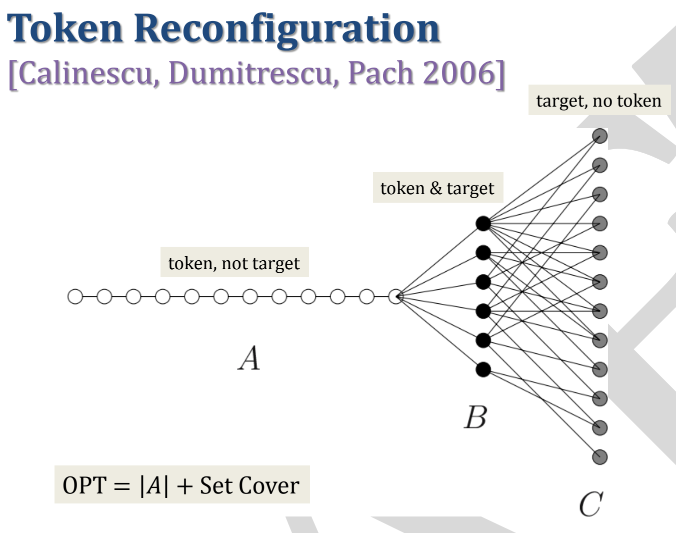
Theorem 10.33. *Set Cover $\le_L$ Motion Planning. Hence there exists a constant c such that, assuming $P \ne NP$, Motion Planning is not $c\ln n$-approximable.*

Proof. We give a reduction Set Cover $\le_L$ Motion Planning.

1. Input is an n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.
2. Form a graph (shown in Figure 10.3) with:
   (a) C is the set of m vertices, one for each S_j.
   (b) B is the set of n vertices, one for each element of $\{1, \ldots, n\}$. Put an edge between elements of B and C if the element associated to B is in the set associated to C.
   (c) A is a set of m vertices that form a line graph. The rightmost element of A has edges to all vertices in B.
3. Robots are put on the elements of $A \cup B$ and the target is $B \cup C$.

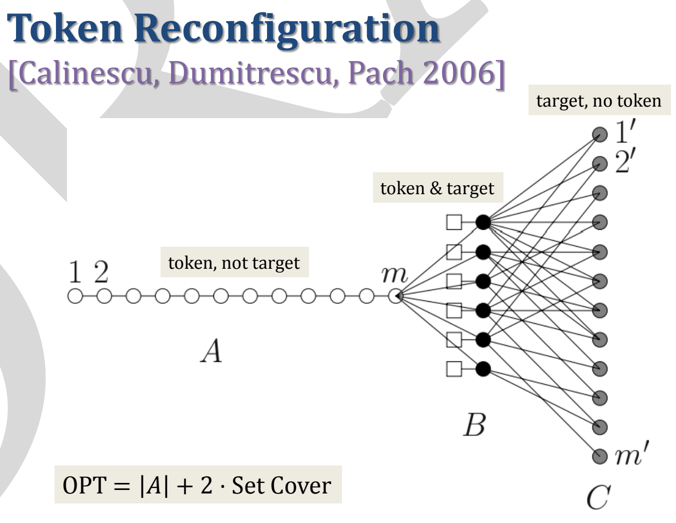
Given an instance of the token reconfiguration problem of this form, the optimal solution takes a number of moves equal to $|A|$ plus the size of a minimal set cover. This is because we need one move to fill every location in C, which requires exactly $|A|$ moves. Furthermore, we need one move to cover every location vacated by a robot in B. But the number of robots in B that must move is exactly the size of the minimum set cover.

Thus, the conditions of the L-reduction are satisfied, and this problem is NP-complete. If 2 robots are not allowed to occupy the same vertex at once, the second image shows a small modification that can be used to get around this problem. The optimal solution in this case is $|A|$ plus twice the size of a minimal set cover.



]

Figure 10.3: Set Cover $\leq_L$ Motion Planning



]

Figure 10.4: Set Cover $\leq_L$ Motion Planning Modified

□

Node-Weighted Steiner Tree (NWST
Instance: A graph $G = (V, E)$ with weights on its nodes, a set $T \subseteq V$ marked as terminal, and a node $r \in V$.
Question: We want a set of nodes S of such that (1) the graph induced by $T \cup S$ connects all the terminal nodes to r, and (2) the sum of the weights in S is minimal over all such S.

Theorem 10.34.

1. *There is an $O(\log n)$-approximation for NWST. This was proven by Moss & Ranbani [MR07]. We omit the proof.*

2. *There is an L-reduction from Set Cover to Node-Weighted Steiner Tree. Hence there is a constant c such that, assuming $P \neq NP$, there is no $c \ln n$-approx for NWST. (This follows from the reduction and Theorem 9.30.4).*

Proof. We give the reduction:

1. Input an instance of set cover: n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.

2. Construct a graph as follows:

   (a) For every $i \in \{1, \ldots, n\}$ there is a node of weight 0. These are the terminal nodes. For every set S_j there is a node of weight 1.

   (b) There is a node r that has an edge to each S_j.

   (c) If $i \in S_j$ then there is an edge between i and S_j.

See Figure 10.5 for an example.

It is easy to see that there is a cost d solution to Node-Weighted Steiner Tree if and only if there is a set cover of size d.

□

Note: One can define the ***Edge Weighted Steiner tree (EWST)*** problem. It is also NP-complete. Bykra et al. [BGRS13] showed there is a 1.39-approximation for EWST Hence, assuming P ≠ NP, there is no reduction of Set Cover to EWST.

Group Steiner Tree (GST)
Instance: A graph $G = (V, E)$ with weights on its edges, sets $V_1, \ldots, V_k \subseteq V$, and a node $r \in V$.
Question: We want a set of nodes S of such that (1) the graph induced by S connects some vertex of each V_i to r, and (2) the sum of the weights in S is minimal over all such S. Note that the graph induced will be a tree.

Theorem 10.35.

1. *The Group Steiner Tree problem has (1) a randomized polynomial-time algorithm that gives an $O((\log^2 n \log\log n \log k)$-approximation for general graphs, and (2) a $O(\log n \log\log k)$-approximation for trees. These were obtained by Garg et al. [GKR00]. We omit the proofs.*

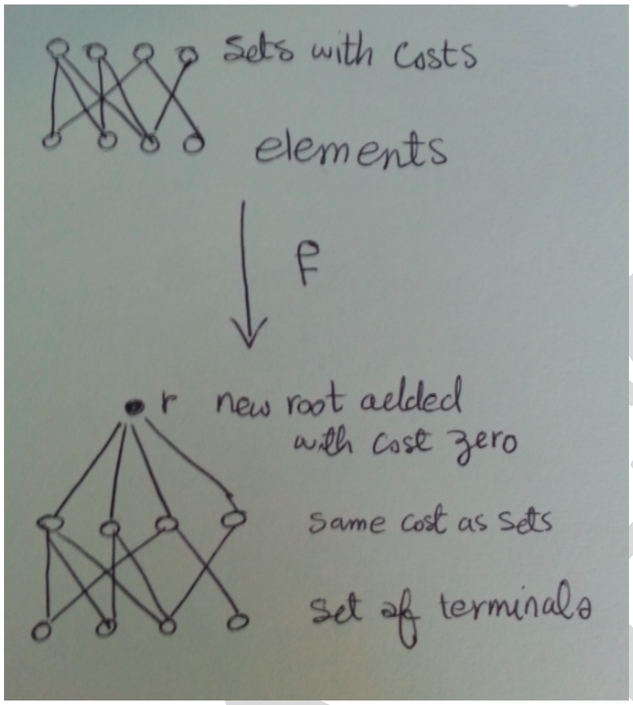


Figure 10.5: APX Reduction from Set Cover to NWST

2. *Set Cover $\le_L$ Group Steiner Tree. We can restrict Group Steiner Tree to trees where every edge has weight 0 or 1.*

3. *If there is a $(1 - o(1)) \ln n$-approx for Group Steiner Tree then P = NP. (This follows from Part 1 and Theorem 9.30.4).*

Proof. We prove part 2 by giving the reduction.

1. Input an instance of set cover: n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.

2. Construct a graph as follows:

   (a) The root is r. It has children $S_1, \ldots, S_m$. The weights of the edges from r to each S_j is 1.

   (b) Each S_j has children labeled with the elements of S_j. These will be the leaves. (So you will have several leaves with the same label). The edges from the i to S_j have weight 0.

   (c) For $1 \le i \le n$, V_i is the set of all leaves labelled i.

We leave it to the reader to show this is an L-reduction.

We do an example. Let $n = 8$ and
$S_1 = \{1, 3, 8\}$.
$S_2 = \{1, 2, 3, 4\}$.
$S_3 = \{5, 6, 8\}$.
$S_4 = \{2, 7\}$.

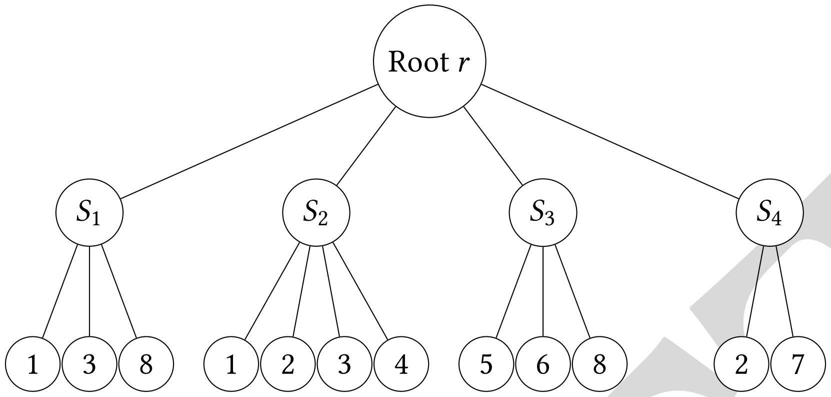


Figure 10.6: Reduction of Set Cover to GST

The resulting graph is in Figure 10.6. The weights on the edges are as follows (1) all of the edges from r to an S_i have weight 1, (2) all of the edges from S_i to one of its children have weight 0. □

10.7.3 Poly-APX-Complete

Bazgan et al. [BEP05] defined Poly-APX which was a framework for saying that problems cannot be approximated any better than within a polynomial factor. Two problems in this class (each of which can easily be reduced to the other) are finding the largest clique and the largest independent set.

10.7.4 EXPTIME-APX-Complete

Escoffier & Paschos [EP06] defined EXPTIME-APX. Encountering this class is rather uncommon. EXPTIME-APX-complete problems occur when there are numbers in the problem that can have large size, increasing the difficulty of approximation. The most well-known problem in this class is the non-metric traveling salesman problem, where edges can be exponential in the size of the input.

10.7.5 NPO-Complete

Crescenzi et al. [CKST99] defined the class NPO and NPO-complete. This class contains solutions of polynomial size, whose decision problems are NP-complete. Examples of problems in this class include:

- maximum/minimum weighted Ones (maximize number of true variables in clauses)
- maximum/minimum 0-1 linear programming

Jonsson [Jon98] defined the class NPOPB and NPOPB-complete. This class contains solutions that are recognizable in polynomial time, whose costs are also computable in polynomial time. (The PB in the title stands for "polynomially bounded".)

However, computing even a valid solution is NP-complete. The distinction between *NPOPB*-complete and NPO-complete is similar to that between Poly-APX-complete and EXPTIME-APX-complete respectively; it's usually due to the presence of large numbers.

NPOPB-complete problems are difficult to approximate polynomially even when the solutions are trivial. Examples of problems in this class include:

- minimum independent dominating set
- shortest Turing-machine computation
- longest induced path
- longest path with "forbidden pairs" (pairs of vertices such that the path cannot cross both)

10.8 Further Results

10.8.1 Maximum Feasible Linear System (Max FLS)

Maximum Feasible Linear System (Max FLS)
Instance: A system of linear equations with coefficient in $\mathbb{Z}$: $A \cdot \vec{x} = \vec{b}$.
Question: A maximum subset of the equations that has a solution over $\mathbb{Q}$.

Max FLS *look* easy since, given matrix A and vector $\vec{b}$ one can, in polynomial time, do the following (by Gaussian elimination):

- Determine whether there is a solution, and if so then find one, and if not then produce a certificate of infeasibility.
- If there is no solution then find a $\vec{x}$ such that $A\vec{x}$ is close to $\vec{b}$. More precisely $\vec{x}$ is such that, $A\vec{x} - \vec{b}$ has the *least mean squared error.*

Nevertheless, Amaldi & Kann [AK95] showed the following:

- The natural decision formulation of Max FLS is NP-hard.
- Many variants and restrictions of Max FLS are NP-hard.
- Assume P ≠ NP. Many variants of Max FLS are hard to approximate. The hardness varies with the variant. Some are in APX but not PTAS, and some are harder to approximate than that.

10.8.2 MaxCut

Recall that there exists an approximation algorithm for MaxCut that returns a number $\geq \frac{1}{2}$OPT. We state better algorithms and some lower bounds.

Theorem 10.36.

1. *(Goemans & Williamson [GW95]) There is an approximation algorithm for* MaxCut *that returns a number* $\geq 0.878\ldots$ *OPT. (The number* $0.878\ldots$ *is actually* $\frac{2}{\pi}\min_{0\leq\theta\leq\pi}\frac{\theta}{1-\cos(\theta)}$*).*

2. *(Hastad [Has01] building on work of Trevisan et al [TSSW00]). Assume P* $\neq$ *NP. Let* $\varepsilon > 0$*. If there no algorithm for* MaxCut *that returns a value* $\geq (\frac{16}{17}+\varepsilon)$*OPT. Note that* $\frac{16}{17} \sim 0.941$*.*

Note that if our hardness assumption is P $\neq$ NP then we do not get matching upper and lower bounds. In Chapter 11 we will see that, assuming the Unique Game Conjecture, the algorithm of Goemans & Williamson is optimal.

10.8.3 Shortest Vector Problem and Its Variants

Definition 10.37.

1. A lattice $\mathcal{L}$ in $\mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$.

2. Let $p \in [1,\infty)$. The ***p*-norm** of a vector $\vec{x} = (x_1,\ldots,x_n) \in \mathbb{R}^n$ is

$$\|\vec{x}\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}.$$

   Note that $p = 2$ yields the standard Euclidean distance.

3. If $p = \infty$ then

$$\|\vec{x}\|_p = \max_{1\leq i\leq n}|x_i|.$$

4. The distance between $\vec{x}$ and $\vec{y}$ in norm p is $\|\vec{x}-\vec{y}\|_p$.

In the next problem let $p \in [1,\infty]$.

Shortest Vector Problem$_p$ (ShVecProb$_p$)
Instance: A lattice L specified by a basis.
Question: Output the shortest vector in that basis using the p-norm.

Lenstra et al. [LLL82] showed that there is a $2^{n/2}$-approximation to ShVecProb$_2$. They used it to obtain an algorithm to factor polynomials. Schnorr [Sch87] improved this to a $2^{n(\log\log n)^2/\log n}$-approximation. There are many non-approx results which indicate that the results of the type Lenstra and Schnorr obtained are the best possible. We state some of them and an also refer the reader to the papers cited for earlier results on this topic.

Theorem 10.38.

1. *(Boas [vEB81])* ShVecProb$_\infty$ *is NP-hard.*

2. *(Ajtai [Ajt98])* ShVecProb$_2$ *is NP-hard under randomized reductions.*

3. *(Khot [Kho05]) Assume NP* $\not\subseteq$ *RP. Let* $p \in (1,\infty)$*. There is no poly-time constant-approx for* ShVecProb$_p$*.*

4. *(Khot [Kho05]) Assume NP* $\not\subseteq$ *RTIME*$(2^{polylog(n)})$*. Let* $p \in (1,\infty)$*. Let* $\varepsilon > 0$*. There is no poly-time* $2^{(\log n)^{1/2-\varepsilon}}$*-approx for* ShVecProb$_p$*.*

5. *(Aggarwal et al. [ABGS21]) Let $p \in [1,\infty) - 2\mathbb{Z}$. Assume SETH. ShVecProb$_p$ cannot be solved in time $O(2^{(1-\varepsilon)n})$ for any $\varepsilon > 0$.*

6. *(Haviv & Regev [HR12] Assume NP $\not\subseteq$ RTIME$(2^{\text{polylog}(n)})$. Let $p \in [1,\infty)$. Let $\varepsilon > 0$. There is no poly-time $2^{(\log n)^{1-\varepsilon}}$-approx for ShVecProb$_p$.*

7. *(Bennett & Peikert [BP22]) Let $p \in [1,\infty)$. ShVecProb$_p$ is NP-hard under randomized reductions. This proof has some aspects to it that make derandomizing it plausible. If this is shown then the hardness assumption of NP $\not\subseteq$ RP can be changed to P $\neq$ NP.*

Micciancio [Mic12] presented new proofs of the results of Khot [Kho05] and Haviv & Regev [HR12] that, while still using random reductions, seem likely to be able to derandomize. This gives evidence that (1) Khot's result can be improved to use the hardness assumption P $\neq$ NP, and (2) Haviv & Regev's result can be improved to use the hardness assumption NP $\not\subseteq$ DTIME$(2^{\text{polylog}(n)})$.

For most of the results in Theorem 10.38 there are similar, but not identical, upper and lower bounds for Closet Vector Problem, the closest vector problem.

10.8.4 Online Nearest Neighbor

A useful problem in data structures is to store a set of points A (in some space) so that, given a point x (not in A), you can determine the point in A that is closest to x. You may be allowed to prepossess the points.

Online Nearest Neighbor (OnNN$_p$ and γ-OnNN$_p$):
Instance: (To Preprocess) A set of points A in $\mathbb{R}^d$. We will assume there are n points.
Instance: A query point x.
Question: (OnNN$_p$) Which point $y \in A$ is closest to x in the p-norm?
Question: (γ-OnNN$_p$ where $\gamma > 1$) We will call the distance to the closest point OPT. Obtain a $y \in A$ such that $\|x - y\|_p \le \gamma$OPT. (We will also allow distances other than p-norms such as edit distance and Hamming distance.)

1. If you do no preprocessing and, given x, compute its distance to every point in A, this takes $O(n)$ time (assuming that distances takes $O(1)$ time). This is considered a lot of time for data structures since (1) n is large and (2) computing a distance is costly even if it $O(1)$.

2. Assume you knew ahead of time the set of query points. You could, in the preprocessing stage, determine for each query point which point of A it is closest to. This would yield query time $O(1)$ but an absurd (1) time for preprocessing, and (2) and space for the data structure.

Is there a way to get both quick preprocessing and quick query times? What if you settle for an approximation? Assuming SETH the answer is no:

Theorem 10.39.

1. *(Rubinstein [Rub18]) Let $p \in \{1,2\}$ Assume SETH. Let $\delta, c > 0$. There exists $\varepsilon = \varepsilon(\delta, c)$ such that no algorithm for OnNN$_p$ has (1) preprocessing time $O(n^c)$, (2) query ties $O(n^{1-\delta})$ and solves $(1+\varepsilon)$-OnNN. (The result also holds for edit-distance and Hamming-distance.)*

2. *(Ko & Song [KS20]) Assume SETH. Let $\delta, c > 0$. There exists $\varepsilon \in \{0, 1)$ such that no algorithm for* OnNN$_p$ *has (1) preprocessing time polynomial in n, (2) query time $O(n^{1-\delta})$ and solves $(1+\varepsilon)$-*OnNN$_p$.

10.8.5 Minimum Bisection

Minimum Bisection (Min Bis)
Instance: A graph $G = (V, E)$ on an even number of vertices.
Question: A partition $V = V_1 \cup V_2$ such that the number of edges from V_1 to V_2 is minimized.

Theorem 10.40.

1. *(Feige & Krautghamer [FK02]) There is a $O(\log n^2)$-approximation for* Min Bis.

2. *(Khot [Kho06]) Let $\varepsilon > 0$. There exists $\delta > 0$ (which depends on ε such that the following is true: Assume 3SAT cannot be solved in $2^{O(n^{\varepsilon})}$ time. Then there is no $(1+\delta)$-approximation for* Min Bis.

Max 3SATMaxCut

Chapter 11

Unique Games Conjecture

11.1 Introduction

As the title indicates, this is a chapter on *The Unique Games Conjecture*. This is a deep field of study that interacts with many branches of mathematics. Hence we will only be able to scratch the surface. We encourage you to, after you read this chapter, read more on our own. One of our sources, that we highly recommend, is Khot's Survey [Kho10a].

In Chapters 9 and 10 we stated (and in some cases proved) upper and lower bounds on many approximation problems. The lower bounds assumed the usual hardness assumption $P \neq NP$ and used the PCP machinery, or reductions. We recall two sets of results to make a point.

Max 3SAT. Let OPT be the max number of clauses that can be satisfied.

1. (Folklore) There is a polynomial-time algorithm that will, given a 3CNF formula φ, find an assignment that will satisfy $\frac{7}{8}\mathrm{OPT}(\varphi)$. We presented this in Theorem 9.34. It was easy.

2. (Arora et al. [ALM+98]) Assume $P \neq NP$. There exists $\varepsilon > 0$ such that there is no polynomial-time algorithm that will, given a 3CNF formula φ, find an assignment that will satisfy $(1-\varepsilon)\mathrm{OPT}(\varphi)$. We presented this in Theorem 9.34. It was easy *given* the PCP Theorem (Theorem 9.25).

3. (Hastad [Has01]) Assume $P \neq NP$. Let $\delta > 0$. There is no polynomial-time algorithm that will, given a 3CNF formula φ, find an assignment that will satisfy $(\frac{7}{8}+\delta)\mathrm{OPT}(\varphi)$. We stated this but did not prove it. The proof is difficult and requires (a) a different kind of proof system which we will discuss in Section 11.2 and (b) hard math as evidenced by the terms "Fourier Transforms" and "Long Codes". We presented (but did not prove) this result in Theorem 9.35.

To summarize: assuming $P \neq NP$, the upper and lower bounds on approximating Max 3SAT *match*.

Vertex Cover. Let OPT be the min size of a vertex cover.

1. (Folklore) There is a polynomial-time algorithm that will, given a graph G, find a vertex cover of size $\leq 2\mathrm{OPT}(G)$. We did not present this; however, it is fairly easy.

2. (Arora et al. [ALM+98]) Assume P $\neq$ NP. There is no polynomial-time algorithm that will, given a graph G, find a vertex cover of size $\leq$ 1.07OPT(G). We presented this in Theorem 9.37. It was a reduction from Max 3SAT. So, much like the lower bound on Max 3SAT, it was easy *given* the PCP Theorem (Theorem 9.25).

3. (Hastad [Has01]) Assume P $\neq$ NP. Let $\delta > 0$. There is no polynomial-time algorithm that will, given a graph G, find a vertex cover of size $\leq (\frac{7}{6} + \delta)$OPT($G$) ($\frac{7}{6} = 1.1666\ldots$). We stated this but did not prove it. The proof is difficult and requires (a) a different kind of proof system that we will discuss in Section 11.2 and (b) hard math as evidenced by the terms "Fourier Transforms" and "Long Codes".

4. (Dinur & Safra [DS05]) Assume P $\neq$ NP. Let $\delta > 0$. There is no polynomial-time algorithm that will, given a graph G, find a vertex cover of size $\leq (10\sqrt{5}-21+\delta)$OPT($G$) ($10\sqrt{5}-21 \sim 1.3606$). We stated this but did not prove it. The proof is difficult and requires (a) a different kind of proof system that we will discuss in Section 11.2 and (b) hard math as evidenced by the terms "Harmonic Analysis", "Extremal Set Theory", and "Long Codes".

To summarize: even assuming P $\neq$ NP, and using hard math, the upper and lower bounds on approximating Vertex Cover *do not match.*

So… what to do?

- Try to get a better approximation algorithm for Vertex Cover. The result that gives $\leq$ 2OPT(G) is very old, and there have been no improvements on it. So this is unlikely. The consensus of the theory community is that $\leq$ 2OPT(G) is the best polynomial-time approximation possible.

- Try to use even harder math or more finely tuned prover-systems to get better lower bounds. Khot [Kho10b] gives reasons why this may not be possible.

So *now* what to do? In Section 11.2 we describe a game that was used to obtain better lower bounds than PCP for many problems; however, most of those lower bounds did not match the upper bounds. Then, in Section 11.3, we will tweak that game to come up with another one that *seems* NP-hard. The assumption that it *is* NP-hard will be dubbed

The Unique Games Conjecture.

From that conjecture one can obtain (1) *matching* upper and lower bounds for approximating Vertex Cover and several other problems for which assuming P $\neq$ NP did not seem to suffice, (2) *better* lower bounds for some problems than can be obtained from assuming P $\neq$ NP, and (3) *better* lower bounds for some problems if you assume variants of the Unique Games conjecture.

Warning: Math games are not fun games. See two blogs of Gasarch [Gas09b, Gas09c] for more on this idea.

Most of this chapter will be about the Unique Games Conjecture and what it implies for lower bounds on approximation. However, we will also discuss a surprising application it has to integrality gaps.

11.2 The 2-Prover-1-Round Game

Arora et al. [ABSS97] defined the following problem as a tool to prove better lower bounds on approximation.

Label Cover (Approximate Label Cover)
Instance:

1. A bipartite graph (V, W, E). (For approximate version also $0 < \delta < 1$.)
2. $a, b \in \mathbb{N}$ with $a \geq b$. (Note that the only restriction on a, b is $a \geq b$. This is one of the things we will tweak.)
3. For each edge $e \in E$ a surjection $f_e : [a] \longrightarrow [b]$. (Note that the only restriction on f_e is that it be surjective. This is one of the things we will tweak.)

Question: Is there a way to label every $v \in V$ with an element $\ell(v) \in [a]$, and every $w \in W$ with an element $\ell(w) \in [b]$, such that, for every $e = (v, w)$, $f_e(\ell(v)) = \ell(w)$? (For the approximate version we want to know whether we can satisfy the fraction δ of the constraints.)
Note: This problem is also called a "2-Prover-1-Round Game". We will see why in Definition 11.2.
Note: There are several variants of the Label Cover; however, they are all equivalent to the one we defined. We discussed one of them in Chapter 9.

Exercise 11.1. Show that Label Cover is NP-complete.

We now reinterpret the problem as a game where 2 provers are trying to convince 1 verifier that a graph has a label covering.

Definition 11.2. The ***2-Prover-1-Round Game*** goes as follows. The board consists of the following:

1. A bipartite graph (V, W, E).
2. $a, b \in \mathbb{N}$ with $a \geq b$.
3. For each edge $e \in E$ a surjection $f_e : [a] \longrightarrow [b]$.

The players are (1) the verifier, and (2) a pair of provers P_1, P_2. The game is played as follows

- The Verifier randomly picks a $(u, w) \in E$ and sends u to P_1 and w to P_2. P_1 and P_2 cannot communicate.
- P_1 outputs an $\ell(u) \in [a]$, P_2 outputs a $\ell(w) \in [b]$.
- If $f_e(\ell(u)) = \ell(w)$ then the provers win. If not then the verifier wins.

Clearly if there is a label covering then the provers can win. Also note that if δ of the edges can be satisfied then there is a strategy for the provers to win δ of the time.

You can view the game as the provers wanting to convince the verifier that there is a label covering. The more rounds of the game they play the harder it will be to fool the verifier. How fast does the error decrease? Raz [Raz98] showed that the error decreases exponentially. This is a very hard and deep theorem that is the key to many other results.

The following can be proven from the PCP Theorem (Theorem 9.25) and Raz's Parallel Repetition Theorem.

Definition 11.3. Let U be an instance of the label cover problem. If δ is the largest fraction of edges that can be satisfied then $\mathrm{OPT}(U) = \delta$.

Theorem 11.4. *Let $0 < \delta < 1$. Then there exists a constant C such that the following happens. Restrict the inputs U to label cover where $a = \Theta((1/\delta)^C)$. It is NP-hard to distinguish between the following cases:*

- *$OPT(U) = 1$ (so there is a label covering)*
- *$OPT(U) \leq \delta$ (so no label covering can be that good).*

This theorem was a key ingredient in getting lower bounds for approximation by Bellare et al. [BGS98] (Max 3SAT, MaxCut, Vertex Cover, Clique, Chromatic Number), Dinur et al. [DGKR05] (Hypergraph Vertex Cover), Dinur & Safra [DS02] (Vertex Cover), Dinur & Steurer [DS13] (Set Cover), Guruswami et al. [GHS02] (c-coloring a 2-colorable 4-uniform hypergraph), Hastad [Has99] (Clique), Hastad [Has01] (Max 3SAT, Vertex Cover), Khanna et al. [KLS00], (4-coloring 3-colorable graphs), Khot [Kho01] (Chromatic Number, Clique) and others. These results offer some good news and some bad news:

- *Good News.* In all cases the lower bounds obtained involved concrete numbers (e.g., Vertex Cover cannot be approximated any better than $1.36\mathrm{OPT}(G)$), in contrast to proofs where the constant was buried in the details of the PCP theorem (e.g. Theorems 9.28 and 9.34).
- *Bad News.* In most cases these improved lower bounds still were not tight. (Exceptions: Dinur & Steurer's lower bound on set cover is tight, and Hastad's lower bound on Max 3SAT is tight.)
- *Worse News.* Khot [Kho10b] gives reasons why the lower bounds obtained by using Theorem 11.4 cannot be improved.

So again... what to do? We will tweak the definition of 2-Prover-1-Round Games in the next section.

11.3 The Unique Games Conjecture

Definition 11.5. We define the ***Unique 2-Prover-1-Round Game***. The board consists of the following:

1. A bipartite graph (V, W, E).

2. $a \in \mathbb{N}$.

3. For each edge $e \in E$ a surjection $f_e : [a] \to [a]$.

The players are (1) the verifier, and (2) a pair of provers P_1, P_2. The game is played as follows

- The Verifier randomly picks a $(u, w) \in E$ and sends u to P_1 and w to P_2. P_1 and P_2 cannot communicate.
- P_1 outputs an $\ell(u) \in [a]$, P_2 outputs an $\ell(w) \in [a]$.
- If $f_e(\ell(u)) = \ell(w)$ then the provers win. If not then the verifier wins.

Clearly if there is a unique label covering then the provers can win. Also note that if δ of the edges can be satisfied then there is a strategy for the provers to win δ of the time.

You can view the game as the provers wanting to convince the verifier that there is a unique label covering.

Definition 11.6. Let U be an instance of the unique label cover problem. If δ is the largest fraction of edges that can be satisfied then $\mathrm{OPT}(U) = \delta$.

The following exercise may surprise you.

Exercise 11.7. Let $0 < \delta < 1$. The following two cases can be distinguished in polynomial time:

- $\mathrm{OPT}(U) = 1$ (so there is a unique label covering)
- $\mathrm{OPT}(U) \le \delta$ (so no unique label covering can be that good).

The only possible way to get a hard problem out of Unique Games is to allow 2-sided error.

UNIQUE GAMES CONJECTURE (UGC)

Conjecture 11.8. *Let $0 < \varepsilon, \delta < 1$. Then there exists a constant C such that the following happens. Restrict the inputs U to unique label cover to those with $a = C$. It is NP-hard to distinguish between the following cases:*

- *$OPT(U) \ge 1 - \varepsilon$ (so there is a fairly good approximate unique label covering)*
- *$OPT(U) \le \delta$ (so no unique label covering can be that good).*

In the next section we will list many consequences of the Unique Games Conjecture.

11.4 Assuming UGC...

In this section we list consequences of the Unique Games Conjecture for lower bounds on approximation. In each subsection we define the problems in our list that have not been defined elsewhere in this book. We sometimes even define a problem that we have defined earlier since we need to say something about it that was not mentioned earlier.

The tables in this chapter abbreviate *Approximation Factor* by *AF*, and *Folklore* by *FL*.

The following notation comes up in several definitions.

Definition 11.9. Let $G = (V, E)$ be a graph. Let $V_1, V_2 \subseteq V$. Then $E(V_1, V_2)$ is the set of edges that have one endpoint in V_1 and the other in V_2.

We divide the problems we look at into three categories. All three categories (1) have to do with lower bounds on approximation, and (2) yield better lower bounds when using Unique Games Conjecture as a hardness assumption than just assuming P ≠ NP.

The three categories are as follows:

- Problems where Unique Games Conjecture yields matching upper and lower bounds on approximations.
- Problems where Unique Games Conjecture does not (yet) yield matching upper and lower bounds.
- Problems where hardness assumption extensions of Unique Games Conjecture yield better lower bounds than hardness assumption P ≠ NP. We denote these extensions Unique Games Conjecture+.

In the tables below we allow randomized algorithms, with high probability of success, in the column *Best Approx*.

Note: In the next three sections we will use Definition 9.5 of α-approximation. Reread that definition and the following paragraph about why it is awkward for Max problems.

11.4.1 Problems Where UGC Implies Optimal Lower Bounds

Vertex Cover on k-Hypergraphs
Instance: k-Hypergraph $G = (V, E)$. Note that $E \subseteq \binom{V}{k}$.
Question: Find the size of the smallest set $V' \subseteq V$ such that every edge $E = \{v_1, \ldots, v_k\}$ has some element of V' in it.

MaxCut
Instance: A graph $G = (V, E)$.
Question: Find the largest value $|E(V_1, V_2)|$ can have where (V_1, V_2) ranges over all partitions of V.
Note: Goemans & Williamson [GW95] have a randomized polynomial-time algorithm for MaxCut that returns a value $\geq \beta$OPT where

$$\beta = \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} \sim 0.87856.$$

Let $\alpha_{MC} = \frac{1}{\beta}$. Using Definition 9.5, α_{MC} is the approximation ratio.

MAX 2SAT
Instance: A 2CNF formula $\varphi = C_1 \wedge \cdots \wedge C_k$.
Question: What is the largest fraction of clauses that an assignment can satisfy?
Note: Lewin et al. [LLZ02] have a randomized polynomial-time algorithm for MAX 2SAT that returns a value $\geq \beta\mathrm{OPT}$ where β is hard to describe but is around 0.94. Let $\alpha_{LLZ} = \frac{1}{\beta}$. Using Definition 9.5, α_{LLZ} is the approximation ratio.

MAX ACYCLIC SUBGRAPH (MAS)
Instance: A directed graph $G = (V, E)$.
Question: What is the size of the largest acyclic subgraph?

Problem	Best Approx	UGC	P $\neq$ NP
VERTEX COVER	2	$2-\varepsilon$ [KR08, BK09]	1.36 [DS02]
VERTEX COVER on k-uniform hypergraphs ($k \geq 3$)	k	$k-\varepsilon$ [KR08, BK10]	$k-1-\varepsilon$ [DGKR05]
MAXCUT	α_{MC} [GW95]	$\alpha_{MC}-\varepsilon$ [KKMO07] [OW08, KO09]	$17/16-\varepsilon$ [Has01]
MAX 2SAT	α_{LLZ} [LLZ02]	$\alpha_{LLZ}-\varepsilon$ [Aus07]	APX-hard [PY91]
MAS	2 [New00]	$2-\varepsilon$ [GHM+11]	$66/65-\varepsilon$ [New00]

11.4.2 Problems Where UGC Implies Good but Not-Optimal Lower Bounds

FEEDBACK ARC SET
Instance: A directed graph $G = (V, E)$.
Question: What is the size of the smallest set of edges that contains at least one edge from every directed cycle?

Definition 11.10. Let $G = (V, E)$ be a graph and $V' \subseteq V$. The ***sparsity of V'***, denoted $S(V')$, is

$$\frac{|E(V', V - V')|}{\min\{|V'|, |V - V'|\}}.$$

SPARSEST CUT (SC). Also called MIN CUT.
Instance: A graph $G = (V, E)$.
Question: What is the minimum possible sparsity of a set of vertices of G?

MIN-2SAT-DELETION
Instance: A 2CNF formula $\varphi = C_1 \wedge \cdots \wedge C_k$ where no clause is of the form $\overline{x} \vee \overline{y}$.
Question: What is the max number of clauses that can be satisfied? (We give results for this version.)
Note: An equivalent formulation, which is where the name comes from, is as follows: Given a 2CNF formula φ (no restrictions) what is the min number of clauses that need to be deleted from φ such that the remaining formula is satisfiable? It is equivalent in that the answer to one of them is k minus the answer to the other one.

MinUncut
Instance: A Graph $G = (V, E)$.
Question: Find $V' \subseteq V$ that minimizes $|E(V', V')| + |E(V - V', V - V')|$.

Problem	Best App	UGC	P ≠ NP
Fback Arc Set	$O(\log n \log\log n)$ [Sey95] [ENSS98]	$\omega(1)$ [GHM+11]	1.36 (Red from VC)
Sparsest Cut	$O(\sqrt{\log n})$ [ALN05] [ARV09]	$\omega(1)$ [CKK+06]	NONE
Min-2SAT-Del	$O(\sqrt{\log n})$ [ACMM05]	$\omega(1)$ [Kho02] $\omega(1)$ [CKK+06]	APX-hard (FL)
MinUncut	$O(\sqrt{\log n})$ [ACMM05]	$\omega(1)$ [Kho02]	APX-complete (FL)

11.4.3 Problems Where UGC+ is Used to Obtain Lower Bounds

We present several problems where, by assuming an extension of Unique Games Conjecture, better lower bounds on approximation can be found. For details on those extensions see the survey by Khot [Kho10b].

Coloring a k-colorable Graph
Instance: A graph G that you are promised is k-colorable.
Question: Find a k' coloring of G. The idea is to make k' as small as possible.
Note: We will abuse terminology in the table below by calling a lower bound on k' the Approx Factor. As with the usual approx factors, these are obtained using assumptions such at P ≠ NP or Unique Games Conjecture. Here is an example: If I say

Assuming P ≠ NP the approx factor for coloring 3-colorable graphs is 4.

that means that

If every 3-colorable graph could be 4-colored in polynomial time then P = NP.

Scheduling with Precedence Constraints
Instance: An acyclic graph G of jobs. The idea is that if there is an edge (i, j) then job i must be completed before job j. Each job has associated to it a processing time p_i and a weight w_i. The weight indicates how important it is to get this job done earlier.
Question: Output a linear ordering of the jobs. This ordering will also give a set of completion times. Let the ith job have completion time c_i. Minimize $\sum_{i=1}^{n} c_i w_i$.

Problem	Best App	UGC+	P ≠ NP
Coloring 3-COL graph	$N^{0.211}$ [AC06]	$\omega(1)$ [DMR09]	4 [KLS00]
Coloring $2d$-COL graph	$N^{1-\frac{3}{2d+1}}$ [KMS98]	$\omega(1)$ [DMR09]	$2d + 2\lfloor \frac{2d}{3} \rfloor - 1$ [KLS00] $d^{\Omega(\log d)}$ [Kho01]
Sch with Prec. Const.	2 (FL)	2ε [BK09]	NONE

11.5 Unique Games Conjecture Implies Integrality Bounds

Usually we aim for theorems like

Assuming P ≠ NP there is no polynomial-time approximation algorithm for Set Cover that returns $(\ln n - \varepsilon)OPT$.

Note that this is a statement about *all* polynomial-time algorithms.

There are times when you have a *particular* polynomial-time approximation algorithm and want to know the limits of how well it can do. In this section we look at algorithms based on relaxation methods and formulate the question of finding limits on how well they can do. We present the following without proof.

1. A relaxed linear programming approach to approximate Set Cover, and the limit to how good it can do. This example has nothing to do with the Unique Games Conjecture; however, it is a good example of integrality gaps.

2. A relaxed Semi Definite Programming approach to approximate MaxCut, and the limit to how good it can do. This is particularly interesting since the limit to how well this particular algorithm can do is exactly the limit you get by assuming Unique Games Conjecture. Hence the Semi Definite Programming algorithm may be optimal.

3. A relaxed Semi Definite Programming approach to approximate Sparsest Cut, and the limit to how good it can do. This is fascinating since (a) the Unique Games Conjecture inspired the proof (actually more than inspired); however, the proof stands without assumption, and (b) the proof involved a lot of hard math, particular geometric embeddings.

11.5.1 Linear Programming and Set Cover

We give an example of how 0-1 programming can be relaxed to linear programming, and then used to approximate Set Cover.

Linear Programming (LP) and 0-1 Programming (ZOP)
Instance: An Integer matrix A and integer vectors $\vec{b}$ and $\vec{c}$.
Question: Find the max value $\vec{c} \cdot \vec{x}$ can have relative to the constraint $A\vec{x} \leq \vec{b}$. The Linear Programming problem restricts $x_i \in \mathbb{Q}$. The 0-1 Programming problem restricts $x_i \in \{0, 1\}$.

The Linear Programming problem is in P, whereas the 0-1 Programming problem is NP-hard.

One way to deal with NP-hard problems is to do the following:

1. Formulate them as a 0-1 Programming problem.

2. Relax this to the Linear Programming problem.

3. Solve the Linear Programming problem.

4. Use the answer to get an approximation for the original 0-1 Programming problem (thats the clever part).

We give an example that seems to be folklore; however, Trevisan [Tre11] has a writeup.

Algorithm to approximate Set Cover

1. Input n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$. We denote the instance of Set Cover X.
2. Formulate the following 0-1 Programming which we denote ZOP(X).
   - The variables are $x_1, \ldots, x_m$. The idea is that $x_i = 1$ means S_i is chosen and $x_i = 0$ means S_i is not chosen. Since this is 0-1 programming, $x_1, \ldots, x_m \in \{0, 1\}$.
   - Constraints. For $j \in \{1, \ldots, n\}$ we need that at least one of the S_i's that has j is chosen. Hence we have the constraint:
   $$\forall j \in \{1, \ldots, n\} : \sum_{j \in S_i} x_i \geq 1.$$
   (Note that the sum is over all i such that $j \in S_i$.)
   - Objective function: we want to minimize
   $$x_1 + \cdots + x_m$$

   Note that the ZOP(X) solves Set Cover exactly.
3. Relax ZOP(X) to an Linear Programming, denoted LP(X). The only difference is that $x_i \in [0, 1]$ instead of $x_i \in \{0, 1\}$.
4. Solve LP(X) to obtain $x_1^*, \ldots, x_n^* \in [0, 1]$ (the x_i^* will be rational by a well known theorem about linear programming).
5. Set x_i to 1 with probability x_i^*.

Trevisan's writeup shows that, with probability ≥ 0.45, the algorithm returns a set cover that is $\leq 2(\ln n + 6)$OPT.

To prove the analysis of the algorithm is roughly tight we would need an instance X of Set Cover which the algorithm above returns a set cover that is $\geq \ln(n)$.

Definition 11.11. The ***integrality gap*** for the above algorithm is

$$\alpha = \inf_X \frac{\text{LP}(X)}{\text{ZOP}(X)} = \frac{\text{LP}(X)}{\text{OPT}(X)}.$$

(We use inf instead of min since it may be a limit.)

It is easy to see that the algorithm cannot do any better than $\leq \alpha$OPT(X).

Lovász [Lov75] showed that the integrality gap is H_n, the nth harmonic number, which is $\ln n$ in the limit. Hence the Linear Programming approach cannot do any better than the simple greedy algorithm of Chvatal [Chv79]. This is worth knowing!

Takeaway: If we find the integrality gap α then it is evidence that the algorithm cannot give an approximation that is any better than α. This shows a limitation on a particular algorithm, not on the problem itself.

11.5.2 Semi Definite Programming and MaxCut

Semi Definite Programming (SDP)
Instance:

- Integers $\{c_{i,j} \mid 1 \le i, j \le n\}$.
- Integers $\{a_{i,j,k} \mid 1 \le i, j, \le n, 1 \le k \le m\}$.

Question: Find vectors $x^1, \ldots, x^n \in \mathbb{R}^n$ such that

- $$\sum_{1 \le i,j \le n} c_{i,j}(x^i \cdot x^j)$$
  is minimized.
- Subject to
  $$(\forall k)[\sum_{1 \le i,j \le n} a_{i,j,k}(x^i \cdots x^j) \le b_k.$$

We will consider Semi Definite Programming to be in polynomial time, though there are some issues about approximation and the reals (that we will not consider).

Goemans & Williamson obtained an approximation algorithm for MaxCut using an Semi Definite Programming. The rest of this section is about their work.

They first formulate MaxCut as an optimization problem (the type does not have a nice name like 0-1 Programming, however, it is NP-complete). They then formulate an Semi Definite Programming which is similar and use it to get an approximation.

Algorithm to approximate MaxCut

1. Input Graph $G = (V, E)$. Let $V = \{1, \ldots, n\}$.
2. Formulate the following problem, denoted $MC(G)$.
   - The variables are $x_1, \ldots, x_n$. The idea is that $x_i = 1$ means i is chosen to be in the cut, and $x_i = -1$ means i is not chosen to be in the cut.
   - Constraints. For $1 \le i \le n$, $x_i \in \{-1, 1\}$.
   - Objective function: we want to maximize:
     $\frac{1}{2}\sum_{(i,j)\in E}(1 - x_i x_j)$.

   Note that the $MC(G)$ solves MaxCut exactly.
3. The analogous Semi Definite Programming (called a ***relaxation*** in the literature) is as follows.
   - The variables are ***vectors*** $x_1, \ldots, x_n$ in $\mathbb{R}^n$.
   - Constraints: For $1 \le i \le n$, $\|x_i\| = 1$.

- Objective function: we want to maximize

$$\frac{1}{|E|}\sum_{(i,j)\in E}\frac{1-x_ix_j}{2}.$$

  We call this SDP(G).

4. Solve SDP(X) to obtain $\vec{x}_1,\ldots,\vec{x}_n$. (The entries might be irrational. There are ways to deal with this; however, we omit this discussion.)

5. Pick a random vector r on the unit sphere.

6. Let $V = V_1 \cup V_2$ where $V_1 = \{i \mid x_i \cdot r \geq 0\}$ (This is called ***randomized rounding*** in the literature.)

Definition 11.12. The ***integrality gap*** for the above algorithm is

$$\alpha = \sup_G \frac{\text{SDP}(G)}{\text{PROBLEM}(G)} = \frac{\text{SDP}(X)}{\text{OPT}(X)}.$$

(We use sup instead of max since it may be a limit.)

Note that this approach to MaxCut cannot do any better then αOPT(G).

Goemans and Williamson [GW95] showed that α is the α_{MC} we defined in Section 11.4.1. Hence their algorithm cannot do better than α_{MX}OPT(G). This constant became more interesting when from Unique Games Conjecture one obtains α_{MC} for the lower bound on how well *any* algorithm can do.

11.5.3 Semi Definite Programming and Sparsest Cut

Leighton & Rao [LR99] used a relaxation of a Linear Programming to obtain an $O(\log n)$-approximation for Sparsest Cut. They also showed that the integrality gap was $\Omega(\log n)$ so the bound was tight. Note again that this is just for their Linear Programming. Aumann & Rabani [AR98] and Linial et al. [LLR95] independently used a geometric theorem of Bourgain [Bou85] to obtain another approach to a relaxation of a Linear Programming for Sparsest Cut. This Linear Programming is looking for a metric with certain properties. They also obtained matching upper and lower bounds of $\Theta(\log n)$.

Even though no better algorithms came out of using Linear Programming's based on metrics, this was a breakthrough since it introduced deep geometrical theorems to the field. Goemans [Goe97] and Linial [Lin02] made a conjecture that metrics exist which would imply an $O(1)$-approximation for Sparsest Cut (so Sparsest Cut would be in APX).

Arora et al. [ARV09] used a relaxation of a Semi Definite Programming to obtain an $O(\sqrt{\log n})$-approximation for Sparsest Cut. So that is progress! But then Khot & Vishnoi [KV15] showed that the integrality gap for *any*
Semi Definite Programming that is based on geometry (and that is probably any Semi Definite Programming) has integrality gap $\omega(1)$. This proof was inspired (actually more than inspired) by the Unique Games Conjecture; however, the result is absolute and needs no assumption.

Realize that this just rules out Semi Definite Programming approaches. While we believe that Unique Games Conjecture is true and therefore Sparsest Cut is not in APX, this has not been proven.

11.6 Is Unique Games Conjecture True?

We are not going to answer the question "Is Unique Games Conjecture True?" since neither we, nor anyone else, knows. That's why its a *conjecture*. However, we will give some arguments for it, together with counters to those arguments.

First we restate it:

Conjecture 11.13. *Let $0 < \varepsilon, \delta < 1$. Then there exists a constant C such that the following happens. Restrict the inputs U to unique label cover to those with $a = C$. It is NP-hard to distinguish between the following cases:*

- *$OPT(U) \geq 1 - \varepsilon$ (so there is a fairly good approximate unique label covering)*
- *$OPT(U) \leq \delta$ (so no unique label covering can be that good).*

1. The Unique Games Conjecture begins $\forall \varepsilon, \delta$ with $0 < \varepsilon, \delta < 1$. What if we allow ε to depend on δ? Formally we ask if the following is true:

   There exists a function $D\colon (0,1) \longrightarrow (1,\infty)$ such that, given δ there exists a constant C such that the following happens. Restrict the inputs U to unique label cover to those with $a = C$. It is NP-hard to distinguish between the following cases:

   - $\mathrm{OPT}(U) \geq D(\delta)\delta$ (so there is a fairly good approximate unique label covering)
   - $\mathrm{OPT}(U) \leq \delta$ (so no unique label covering can be that good).

   Feige & Reichman [FR04] showed this is true. Great! But then $D(\delta) \longrightarrow \infty$ as $\delta \longrightarrow 0$. And $D(\delta)\delta \longrightarrow 0$ as $\delta \longrightarrow 0$. Hence it is not clear if this result really indicates Unique Games Conjecture is true.

2. If ε, δ are both close to $\frac{1}{2}$ then we expect the problem to be hard, and hence the conjecture to be true. Indeed, O'Donnell & Wright [OW12] showed that if $\varepsilon = \frac{1}{2}$ and $\frac{3}{8} < \delta < \frac{1}{2}$ then the problem is hard. Khot et al. [KMS18] (building on the work of Dinur et al. [DKK+18b, DKK+18a] and Khot et al. [KMS17]) showed that if $\varepsilon = \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$ then the problem is hard. Even though this is not the full Unique Games Conjecture, the result already yielded some lower bounds on approximation. See Barak's [Bar18] blog post on these results for a good overview. Are these real indications that Unique Games Conjecture is true? As usual, not clear. However, the lower bounds gotten from the results show that Unique Games Conjecture is worth studying.

3. There is an Semi Definite Programming approach to solving The Unique Games Problem. Khot & Vishnoi have shown, by integrality gap methods, that this approach will not get Unique Games Conjecture in polynomial time. Yeah. But is there some other algorithmic paradigm that has not been ruled out? There is! Lasserre [Las01a, Las01b] devised a way

to iterate Semi Definite Programming programs to obtain better approximations (also see Rothvob's notes [Rot13a] and the paper of Karlin et al. [KMN11]). Khot et al. [KPS10] have shown results that indicate (though do not prove) that Lasserre's approach with a constant number of rounds cannot solve The Unique Games Problem. This still leaves open the possibility of non-constant number of rounds. What makes the Lasserre method so hopeful (or non-hopeful if you think UGC is true) is that it has not been ruled out by integrality gaps or other paradigms.

4. Arora et al. [ABS15] have a subexponential algorithm for the problem. This tends to indicate that Unique Games Conjecture is false. However, we note that there are some NP-complete problems with subexponential time algorithms, though they are contrived and involve padding the input.

5. (This is what we consider the most compelling reason.) UGC has great explanatory power. Take Vertex Cover as an example. It seemed like it was very hard to get a $(2-\varepsilon)$OPT approximation for Vertex Cover. And the community believed that it was impossible. But we couldn't prove it. AH- but with Unique Games Conjecture we can! Vertex Cover is not an isolated example. The table in Section 11.4 give many cases where we get either matching bounds or better bounds. And these bounds match our intuition. Counterargument: We may one day prove all of these better lower bounds from $P \neq NP$. That is possible and has not been ruled out.

11.7 Open Problems

1. Prove or disprove Unique Games Conjecture.

2. For the tables in Section 11.4 for which we don't have matching bounds, close the gap. This may be done by either better approximation algorithms or better reductions.

3. There may come a time when we think, for lower bounds on approximation, that Unique Games Conjecture has gone as far as it can go (we may already be there with $P \neq NP$). When this happens find a new conjecture that is reasonable and whose assumption will help close the gaps.

Part II

Above NP

Chapter 12

Counting Problems

12.1 Introduction

Recall that $\varphi \in \text{SAT}$ if there exists *at least one* satisfying assignment for φ. For SAT all we care about is the distinction between 0 and ≥ 1 satisfying assignments. What if we want to know *how many* satisfying assignments there are? This problem is clearly at least as hard as SAT but is it harder? (Spoiler Alert: Probably Yes.) Valiant [Val79c, Val79a, Val79b] defined the class of functions that capture this notion.

In this chapter we will mostly look at counting versions of problems in P and NP. We will also look at two other topics about number-of-solutions: (1) ANOTHER SOLUTION: If you have, say, one satisfying assignment, will it help you find another?, and (2) FEWEST CLUES: What is the least number of (say) assignments to variables do you need so that the remaining formula has only one satisfying assignment?

Definition 12.1.

1. #SAT is the following function: on input a Boolean formula φ, output the number of satisfying assignments.

2. Let $A \in \text{NP}$. Hence there exists a polynomial p and a set $B \in \text{P}$ such that
$$A = \{x \mid \exists y : |y| = p(|x|) \wedge (x, y) \in B\}.$$
$\#A$ is the function that, on input x, outputs the number of y of length $p(|x|)$ such that $(x, y) \in B$.

3. #P is the set of all functions of the form $\#A$ where $A \in \text{NP}$.

Note: The above definition of $\#A$ is ambiguous as written. There may be many different p, B for a set $A \in \text{NP}$. Here is an example:
$$\text{SAT} = \{\varphi(x_1, \ldots, x_n) \mid \exists x = yz : |y| = |z| = n \wedge \varphi(y) = \text{TRUE} \wedge\ z \text{ is a square}\}.$$
Using this definition of SAT, $\#\text{SAT}(\varphi(x_1, \ldots, x_n))$ would be
$$(\text{Number of Satisfying Assignments}) \times (\text{Number of squares of length } n).$$

This is clearly not what we want. We will ignore this issue and assume that the B, p used to define A are natural. For example, #3COL is the number of 3-colorings of a graph, #Ham Cycle is the number of Hamiltonian cycles in a graph, etc.

Is #SAT much harder than SAT? Likely yes. Toda [Tod91] showed the following.

Theorem 12.2. *If A is in the Polynomial Hierarchy (so $A \in \Sigma_i$ or $A \in \Pi_i$) then A is reducible to #3SAT. The reduction may use many evaluations of #3SAT. Hence, if #3SAT is in FP, then every set in the polynomial hierarchy is in P.*

Our motivation for studying counting problems comes from puzzle design. We usually want to know whether there is a unique solution for a puzzle. Hence we want to know when there is exactly one solution.

12.2 Reductions and Completeness

The statement $A \le_p B$ can be viewed as saying that we can solve the question $x \in A$ by making one call to B, and the answer to the call is the answer. We define reductions that allow many calls and can use the information any way you want. Formally this would require the definition of an oracle Turing machine. We are not going to define the concept formally. All you need to know is that an oracle Turing machine $M^{()}$ can make queries to whatever is in those parentheses (a set or a function). In the calculation of $M^g(x)$, if the machine wants to know $g(y)$ it only has to write down y on a special tape.

Definition 12.3. Let f, g be functions.

1. $f \le_{p,o} g$ if there is an oracle Turing machine $M^{()}$ such that $f(x) = M^g(x)$ and the calculation of $M^g(x)$ takes polynomial time.
2. g is #P-hard if, for all $f \in$ #P, $f \le_{p,o} g$.
3. g is #P-complete if $g \in$ #P and g is #P-hard.

We will often want to show that, for some set $A \in$ NP, #A is #P-complete. We will do this by presenting a certain type of reduction from A to B. We define two such reductions.

Definition 12.4. Let A, B be sets in NP. A ***parsimonious reduction*** from A to B is a reduction f such that the following hold.

1. $x \in A$ if and only if $f(x) \in B$.
2. The number of solutions for $x \in A$ (e.g., the number of satisfying assignments) equals the number of solutions for $f(x) \in B$ (e.g., the number of cliques of size k).

The next terminology is due to Erik Demaine, though the idea of consistent expansion of the number of solutions is well-used in the original #P papers (e.g., [Val79a]).

Definition 12.5. Let A, B be sets in NP. A ***c-monious reduction*** from A to B is a reduction f such that there exists a constant a such that the following hold.

1. $x \in A$ if and only if $f(x) \in B$.

2. The number of solutions for $x \in A$ (e.g., the number of satisfying assignments) equals a times the number of solutions for $f(x) \in B$ (e.g., the number of cliques of size k).

Exercise 12.6.

1. Show that #SAT is #P-complete.
   Hint: Modify the proof of the Cook-Levin Theorem.

2. Show that if there is a parsimonious or c-monious reduction from SAT to A then $\#A$ is #P-hard.

3. Show that if $\#A$ is #P-complete and $\#A \in$ FP then #P = FP.

12.3 Thoughts on "For which $A \in$ NP is $\#A$ #P-Complete?"

Valiant [Val79b] showed that for many $A \in$ NP, there is a parsimonious reduction from SAT to A, and hence $\#A$ is #P-complete. problems are #P-complete. We will present some of those proofs, as well as other proofs that problems are #P-complete or #P-hard.

Empirically it seems that, for every natural problem A that is NP-complete, $\#A$ is #P-complete. This is not a theorem, and it probably cannot be a theorem since it is hard to define "natural". Are there sets $A \in$ P for which $\#A$ is #P-complete? Yes. We state four of them.

Theorem 12.7.

1. *Brightwell et al. [BW05] showed that counting the number of Eulerian Circuits in a graph is #P-complete. Note that detecting whether a graph has an Euler Circuit is in P.*

2. *Jerrum [Jer94] showed that counting the number of labeled trees in a graph is #P-complete. Note that detecting whether some subgraph of a graph is a tree is trivial, the answer is always yes.*

3. *Valiant [Val79b] showed that counting the number of bipartite matching is #P-complete. We will see this as an easy corollary of* Perm *being #P-complete. Note that finding a matching in a bipartite graph (even a general graph) is in P. Vadhan [Vad01] showed that this problem is still #P-complete when G is restricted to bipartite graphs (1) with degree 4, (2) planar of degree 6, or (3) k-regular for any $k \geq 5$. That paper has many other problems in P whose counting version is #P-complete.*

4. *Valiant [Val79b] showed that counting the number of Satisfying assignments in a monotone 2-SAT formula is #P-complete. Note that the problem of determining whether a monotone 2-SAT formula is satisfied is trivial, the answer is always yes. Provan and Ball [PB83] extended this result.*

Exercise 12.8.

1. Find several sets $A \in$ NP that are also in P such that $\#A$ is in polynomial time.

2. (Vague Open Problem) Find a difference between the problems you found in Part 1 and the problems from Theorem 12.7

12.4 For Many $A \in$ NP, #A is #P-Complete

We show many functions in #P are #P-complete using parsimonious reductions.

12.4.1 #3SAT, #Clique, and #3SAT-3

Theorem 12.9.

1. *#SAT is #P-complete. This is Exercise 12.6, hence we omit the proof.*
2. *There is a parsimonious reduction from SAT to 3SAT. Hence #3SAT is #P-complete.*
3. *There is a parsimonious reduction from 3SAT to 3SAT-3. Hence #3SAT-3 is #P-complete.*
4. *There is a parsimonious reduction from 3SAT to Clique. Hence #Clique is #P-complete.*

Proof.
2) Here is the parsimonious reduction:

1. Input $\varphi = C_1 \wedge \cdots \wedge C_k$ where each C_i is an OR of literals.
2. Introduce three new variables x, y, z and the clauses
   $(x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge$
   $(\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z)$
   Hence in any satisfying assignment x, y, z are all set to false.
3. For all C_i that are 1-clauses, $C_i = w$, replace it with $w \vee x \vee y$.
4. For all C_i that are 2-clauses, $C_i = w_1 \vee w_2$, replace it with $w_1 \vee w_2 \vee x$.
5. For all C_i that are 3-clauses, no change needed.
6. We now talk about C_i that have ≥ 4 literals. We do an example of what to do with a 5-clause. From our example the general case will be clear.
   Let the clause be $L_1 \vee L_2 \vee L_3 \vee L_4 \vee L_5$.
   Note that all the $L's$ are literals, so they can be variables or their negations.
   Introduce a new variables W. Consider
   $$(L_1 \vee L_2 \vee W) \wedge (\neg L_1 \vee L_2 \vee W) \wedge (L_1 \vee \neg L_2 \vee W) \wedge (\neg L_1 \vee \neg L_2 \vee \neg W) \wedge$$
   $$(\neg W \vee L_3 \vee L_4 \vee L_5)$$
   If $W =$ FALSE then we get all solutions where $(L_1, L_2) =$ (TRUE, TRUE).
   If $W =$ TRUE then we get all solutions where $(L_1, L_2 \in \{$(TRUE, FALSE), (FALSE, TRUE), (FALSE, FALSE)$\}$.
   Hence the number of satisfying assignments is the same. But we still have a 4-clause.

Do the same procedure on the 4-clause that we did on the 5-clause.

We leave it to the reader to formalize the case where there are ≥ 4 literals and then to show that the reduction is parsimonious.

3) The reduction from 3SAT to 3SAT-3 in Theorem 1.9 is parsimonious.

4) The reduction from 3SAT to Clique in Theorem 2.3 is parsimonious.

□

Exercise 12.10.

1. Give the reduction for Theorem 12.9.2 in the case where the formula has ≥ 4 literals.
2. Prove that the reduction in Theorem 12.9.2 is parsimonious.
3. Prove that the reduction in Theorem 12.9.3 is parsimonious.

12.5 #Planar 3SAT and Variants

The reader is advised to read Chapter 2 since in this section we discuss the reductions in that chapter with an eye towards either observing they already are parsimonious or modifying them so they are parsimonious.

Exercise 12.11.

1. #Planar 3SAT is #P-complete.
   Hint: The reduction in Theorem 2.9 is parsimonious, though this needs to be proven.
2. In Chapter 2 we proved that for several problems X, Planar 3SAT $\leq_p X$. Modify these reductions (if needed) to show that #X is #P-hard. Note which ones are actually #P-complete.

Theorem 12.12. *#Planar Rectilinear 3SAT is #P-complete.*

Proof sketch: The reduction in the proof of Theorem 2.24 will suffice. ▮

Exercise 12.13. Prove Theorem 12.12 rigorously.

We would like to show that #Planar 1-in-3SAT is #P-complete. The reduction from 3SAT to Planar 1-in-3SAT shown in Figure 12.1 is not parsimonious. There is exactly one case when we can set the variables of the original instance and cannot force the internal variables to an specific value. The figure shows that there are two possible solutions of the Planar 1-in-3SAT instance that correspond to the same solution of the 3SAT instance. If every solution of the 3SAT instance had 2 corresponding Planar 1-in-3SAT solutions we would have a c-monious reduction, but this irregularity makes this reduction not parsimonious.

Luckily, there is a stronger reduction from Planar Rectilinear 3SAT to Planar Positive Rectilinear 1-in-3SAT that does work. In this proof, all the variables are forced. The number of solutions is conserved and we have a parsimonious reduction.

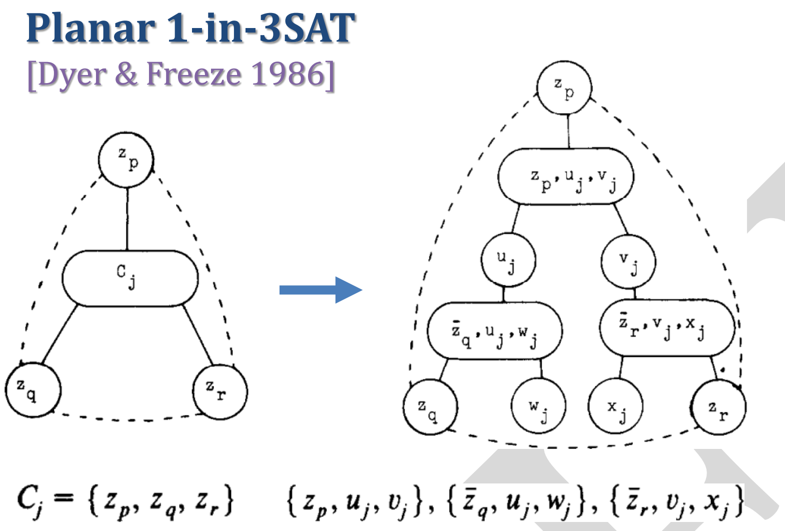


Figure 12.1: Planar 1-in-3SAT Reduction Transformation

12.6 #Planar Directed Ham Path is #P-Complete

In the proof of Theorem 4.2 we showed that Planar 3SAT reduces to Planar Directed Ham Path restricted to graphs of degree 3. See Figures 12.2 and 12.3 for all of the gadgets used. The reduction was not parsimonious. When multiple variables satisfy a clause any of them can grab the vertices in the clause gadget. Since we can sometimes have 1, 2 or 3 possible solutions that correspond to the same 3SAT solution, we don't have a parsimonious (or c-monious) reduction. We sketch how to modify the reduction to make it parsimonious.

Theorem 12.14.

1. *#Ham Cycle restricted to directed planar max-degree-3 graphs is #P-complete.*

2. *#Ham Cycle is #P-complete.*

Proof sketch: We can use the XOR-gadget from Yato & Seta [YS03] to prove

$$\text{Planar 1-in-3SAT} \le_p \text{Planar X3C}.$$

This gadget admits only one solution and forces that only (and exactly) one edge connected by the gadget is used in the cycle.

Three new gadgets were created for this reduction. They are shown in Figure 12.4 and are: OR (that requires that one or two of the edges are used in the cycle), implication (that forces an edge to be used if the other edge is used) and a 3-way OR gate (that is used in the clause). The difference between this clause gadget with a 3-way OR, and the previous reduction, is that the new gadget has all its edges forced admitting only one cycle per solution of the original 3SAT instance. Consequently, this new reduction is parsimonious and proves that finding the number of Hamiltonian cycles in a planar max degree-3 graph is #P-hard.

The reduction of the restricted case to Ham Cycle is trivial.

▮

Planar Directed Max-Degree-3

[Plesńik 1979]

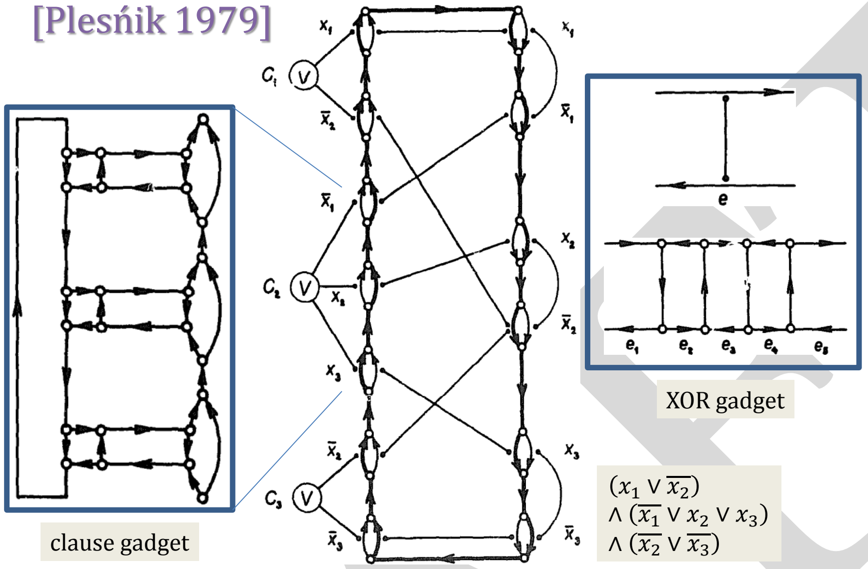


Figure 12.2: Gadget for Proving Ham Cycle on Planar Max-Degree-3 Graphs is NP-complete

Planar Directed Max-Degree-3

[Plesńik 1979]

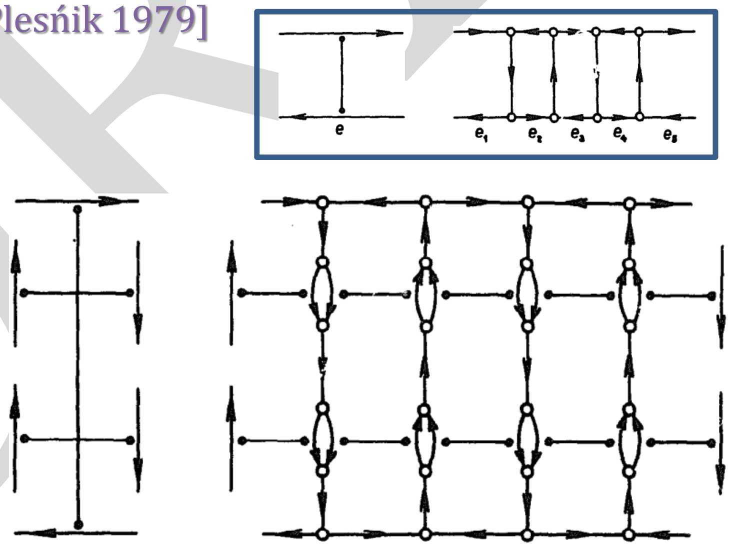


Figure 12.3: Another Gadget for Proving . . .

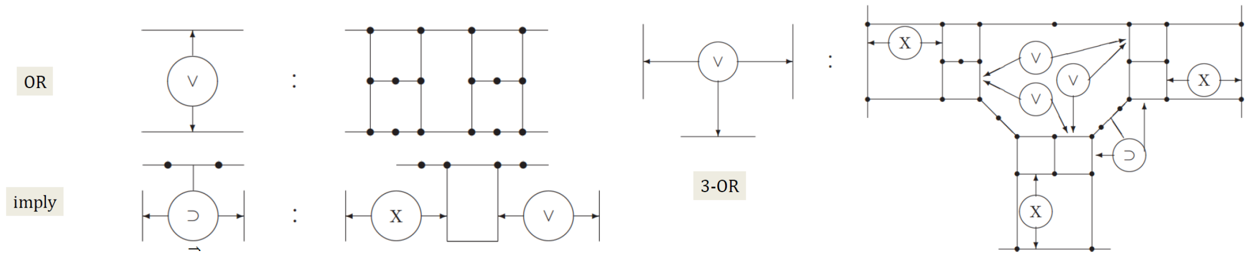


Figure 12.4: New Gadgets for the Reduction to Hamiltonian Cycle

12.7 Slitherlink

Recall that in Section 4.7.2 we defined the game of Slitherlink and showed that it was NP-complete. We are now interested in counting how many ways someone can win, which is of course a counting problem.

The proof that Slitherlink is NP-complete used a reduction from Hamiltonian Cycle restricted to grid graphs. Alas, we do not have the counting version of this variant of Hamiltonian cycle #P-complete. However, we can use the restriction of Hamiltonian cycle to directed planar max-degree 3 graphs which was shown #P-complete in Theorem 12.14.

Theorem 12.15. *#Slitherlink is #P-complete.*

Proof sketch: We show a reduction from directed planar max-degree 3 graphs.

Observe the figure below and notice that for any Hamiltonian cycle (i.e. order in which we visit the vertices) in the planar graph, there is a unique way to solve the puzzle by visiting the 'required' vertices in the corresponding order and given this order there is exactly one way to use the optional vertices to complete the cycle. Hence the reduction is parsimonious and we get #P-hardness for Slitherlink.

∎

12.8 Permanent

All results in this section are due to Valiant [Val79a].

Definition 12.16. Let $M = (m_{i,j})$ be an $n \times n$ matrix.

1. The ***determinant***, denoted $\textsc{Det}(M)$, is $\sum_{\pi}(-1)^{sign(\pi)}\Pi_{i=1}^{n} m_{i,\pi(i)}$ where the sum is over all permutations π of the set $\{1, 2, \ldots, n\}$ and $sign(.)$ of a permutation is a 0/1 value given by the parity of the number of inversions mod 2. Note that even though this definition has an exponential (in n) number of terms, computing the determinant is in FP.

2. The ***permanent*** of M, denoted $\textsc{Perm}(M)$, is $\sum_{\pi}\Pi_{i=1}^{n} m_{i,\pi(i)}$ i.e. the unsigned sum over permutations.

It has been known for a long time that computing the determinant is easy. People wondered if computing the permanent was also easy. Before the modern notions of complexity were established people wondered if, in our terminology, Perm $\le_{p,o}$ Det.

The problem of showing Perm was hard motivated Valiant [Val79a] to define #P. He proved that computing the permanent is #P-complete. To present his proof we need the following definition and exercise that give an alternative definition of Perm.

Definition 12.17. A ***cycle cover*** of a graph G is a set of (not necessarily disjoint) cycles $C_1, \ldots, C_L$ such that every vertex of G is in at least one of the C_i. The ***weight*** of one cycle is the product of the weights of the edges (this is unusual—usually weights are sums). The ***weight*** of the cycle cover is the sum of the weights of the cycles.

Exercise 12.18.

1. Let M be a matrix. View M as the adjacency matrix of a weighted directed graph G. Show that the permanent of M is the sum of all of weights of the cycle covers of G.

2. Use Part 1 of this exercise to show that Perm $\in$ #P. (You might want to use the definition of NP that uses nondeterminism.)

Theorem 12.19. *Perm is #P-complete and remains so when the matrix is restricted to having elements in* $\{-1, 0, 1, 2, 3\}$.

Proof. Perm $\in$ #P by Exercise 12.18.

We now show that Perm is #P-hard. We show there is a c-monious reduction from 3SAT to Perm.

All edges shown in the gadgets are weight one. The shaded structure used in the gadgets is a graph with weight matrix (X) as shown in Figure TO BE ADDED LATER. Notice that the permanent of the matrix is 0 hence in any nonzero weight cycle cover for the whole graph all such structures must be a part of a bigger cycle that enters and leaves at some point (else there would be a 0 in the product).

Also, although not explicitly shown, a cycle can enter/exit these structures only at the structures' nodes that correspond to row/column 1 and row/column 4. If a cycle enters and leaves at the same node (1 or 4) then we again get a 0 in the product due to the remaining part of the structure because the permanent of X-row/column 1 and X-row/column 4 is also 0. Hence cycles must enter every node and leave at distinct points for all structures.

Also, X-row/columns 1,4 has permanent 0 as well, hence the cycle entering and exiting the structure at nodes 1,4 must go through at least one more internal node as well to have a non-zero product weight.

Now the permanent of matrix X-row 1, column 4 and of X-row 4, column 1 is 4 hence we get a factor 4 in the weight product of a cycle cover due to each structure.

In summary the structures act as forced edges in variable and clause gadgets. In the variable gadget this forces the choice of either x or $\bar{x}$ and in the clause gadget if none of the literal structures have been covered by external edges (T_2, T_5, T_7) then we must traverse all 5 structures in the clause in a consecutive manner but having done that there is no way to finish a cycle, hence at least one of the structures has to be covered by an external cycle, which corresponds to that literal being 1.

Now note that for every solution to the 3SAT instance there are exactly 4^5 possible cycle covers for every clause (because of 5 structures in a clause gadget each coverable 4 different ways), given the variable assignment of the 3SAT solution. Hence the reduction is a c-monious reduction with $c = 4^{5(\#clauses)}$. □

We will need the definition of PermMod in the proof of our next theorem.

Definition 12.20. PermMod is the problem of, given a matrix M and a number r, determine Perm$(M) \pmod{r}$.

Theorem 12.21. *Perm is #P-complete and remains so when the matrix is restricted to 0-1 matrices.*

Proof. We take Perm to mean the cycle cover problem in Exercise 12.18. To prove the result it suffices to give a parsimonious reduction from the Weighted cycle-cover problem for directed weighted graphs (with weights in $\{-1, 0, 1, 2, 3\}$) by Theorem 12.19 to the cycle-cover problem for directed unweighted graphs.

We replace each edge of weight k with a gadget with k loops. Figure TO BE ADDED LATER below shows the gadget for an edge of weight 5.

If (x, y) is not covered by a cycle in the instance of PermMod there is exactly one way to cover the edges. If (x, y) is covered in a cycle there are exactly k solutions (each using a different self-loop). □

What if we just want the permanent mod r? Is that also hard? Note that this problem does not appear to be in #P. Hence we prove a hardness result, not a completeness result.

Theorem 12.22. *PermMod is #P-hard, even when restricted to 0-1 matrices.*

Proof. We show that Perm $\leq_{p,o}$ PermMod. By Theorem 12.21 we can assume that Perm is a 0-1 matrix.

1. Input(M), and $n \times n$ 0-1 matrix.
2. Compute $n!$ which bounds the size of the Perm(M).
3. Find the least prime p such that the product of all the primes up to and including p is larger than $n!$.
4. For $r = 2, 3, 5, 7, 11, \ldots, p$ (so r goes through all the primes $\leq p$) calling PermMod on (M, r). This requires $O(n \log n)$ calls which is polynomial in the size of the input.
5. Using the results of these calls and the Chinese Remainder Theorem we can determine Perm(M) in polynomial time.

□

What if we fix the mod? Then the problem is in P:

Theorem 12.23. *Fix r. The problem of finding the permanent of an $n \times n$ matrix mod r can be done with $O(n^{4r-3})$ arithmetic operations. Hence if there is a bound on the entries (e.g., a 0-1 matrix) then the problem is in polynomial time.*

12.9 Counting Bipartite Matchings: Perfect and Maximal

Definition 12.24. (Figure 12.5 illustrates the concepts in this definition.) Let G be a graph.

1. A ***matching for G*** is a set of edges that share no vertices.
2. MAT is the set of graphs that have a matching. Bip-MAT is the set of bipartite graphs that have a matching. These are silly definitions since all graphs that have a a matching, namely the empty set. However, we will see that #MAT and #Bip-MAT are #P-complete.
3. A ***perfect matching for G*** is a matching M so that every vertex is the endpoint of some edge in M. Note that a ***perfect matching for a bipartite graph G*** is a bijection from the left vertices to the right vertices. This follows from the definition of a perfect matching on a graph.
4. PerfMat is the set of graphs that have a perfect matching. Bip-PM is the set of bipartite graphs that have a perfect matching. It is known that both PerfMat $\in$ P and Bip-PM $\in$ P; however, we will see that #PerfMat and #Bip-PM are #P-complete.
5. A ***maximal matching of G*** is a matching M such that, for any edge e that is not in the matching, $M \cup \{e\}$ is not a matching. Note that a graph with no edges has $\emptyset$ as a maximal matching.
6. MaximalMat is the set of all graphs that have a maximal matching. Bip-MM is the set of bipartite graphs that have a maximal matching. These are silly definitions since, by a greedy algorithm, *every* graph has a maximal matching and it can be found quickly; however, we will see that #MaximalMat and #Bip-MM are #P-complete.

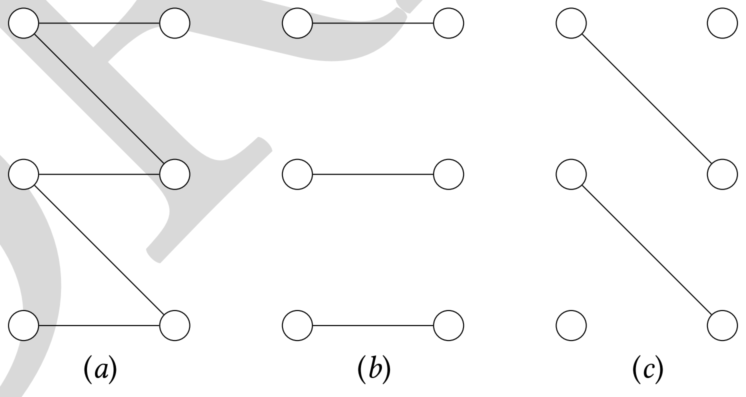


Figure 12.5: (a) Bipartite Graph (b) Perfect Matching (c) Maximal Matching

Exercise 12.25. Let M be an $n \times n$ a 0-1 matrix. We denote the ijth entry by a_{ij}. Let G be the bipartite graph with both left and right sets of vertices be $\{1, \ldots, n\}$, and there is an edge from i (Left) to j (Right) if and only if $a_{ij} = 1$. Show that the permanent of M is #Bip-PM(G).

Theorem 12.26. *#Bip-PM is #P-complete. Hence #PerfMat is #P-complete.*

Proof. This follows from Exercise 12.25 and Theorem 12.21. □

Theorem 12.26 uses that the graph is bipartite. None of the remaining results in this section need that assumption. Even so, we state the results for bipartite graphs.

Theorem 12.27. *#Bip-MAT and #Bip-MM are #P-complete. Hence #MAT and #MaximalMat are #P-complete.*

Proof. We show #Bip-PM $\le_{p,o}$ #Bip-MM and #Bip-PM $\le_{p,o}$ $\#BIP-MAT$ at the same time since the reductions are so similar. They are both one-call reductions.

1. Input $G = (X, Y, E)$, a bipartite graph with $|X| = |Y| = n$ (if $|X| \neq |Y|$ then there are no perfect matchings). We want to know #Bip-PM(G).

2. Create a bipartite graph by doing the following:

   (a) Replace every node v in $X \cup Y$ with n^2 nodes $v_1, \ldots, v_{n^2}$.

   (b) If (v, w) is an edge in G then put an edge between every v_i and w_j.

   (c) Call this new bipartite graph G'.

   (d) Make the query #Bip-MM(G'). We now give commentary before saying the next step. Let M be a (maximal) matching of G with i edges. Then it will correspond to $(n^2!)^i$ (maximal) matchings in G'. For $1 \le i \le n$ let m_i be the number of (maximal) matchings with i edges. Note that m_n = #Bip-PM(G) (in either the matching or maximal matching case). Hence

   $$\#\text{Bip-MM}(G') = \sum_{i=1}^{n} m_i (n^2!)^i.$$

   Since $m_i \le \binom{n^2}{n} < n^2!$ we can view #Bip-MM(G') as the base-n^2! number $m_n m_1 \cdots m_0$.

   (e) Express #Bip-MM(G') in base n^2! to obtain $m_n \cdots m_0$. Output m_n.

□

12.10 Counting Versions of 2SAT and Several Graph Problems

Recall that 2SAT ∈ P. What about #2SAT? Is it #P-complete? It is. We do not prove this. We prove a variant of #2SAT is #P-complete, and use this variant to show several other problems #P-complete or #P-hard.

Definition 12.28. Th-Pos-2SAT is the following problem: given a Boolean formula in 2-CNF form with all literals positive, and a number t (called the ***threshold***), does it have a satisfying assignment with at least t variables set to false.

Theorem 12.29. *#Th-Pos-2SAT is #P-complete.*

Proof. We describe a parsimonious reduction from PerfMat to Th-Pos-2SAT.

1. Input $G = (V, E)$, a graph on $2k$ vertices. Note that any perfect matching will have k edges.
2. Form a formula as follows:
   (a) For every edge e we have a Boolean variable v_e. Our intent is that if e is in the matching then v_e is set to FALSE (you read that right).
   (b) For every edges e and e' that are incident we have the clause $(v_e \vee v_{e'})$.
   (c) Let the formula be φ.
3. Output (φ, t).

We show a bijection between the perfect matchings of G and the satisfying assignments of φ that have $\geq k$ variables set FALSE.

Given a perfect matching, every variable corresponding to an edge in the matching is set to FALSE, and all of the other variables are set to TRUE. Since the edges in the matching are not incident to each other, there is no clause that has two variables false. Hence this is a satisfying assignment.

We leave it to the reader to show that every satisfying assignment is in the image of this map. □

We can use Theorem 12.29 to prove several more problems are #P-complete or hard.

Theorem 12.30.

1. *#Independent Set is #P-complete.*
2. *#Clique is #P-complete (this will be a different proof then the one given in Theorem 12.9).*
3. *Counting the number of maximal independent sets is #P-hard.*
4. *Counting the number of maximal cliques is #P-hard.*

Proof. Recall that Independent Set is the problem of, given (G, k) determine whether there is an independent set of size k. So #Independent Set(G, k) is the number of independent sets of size k.

1. Input (φ, k), where φ is a 2CNF formula with all positive literals.
2. Form the following graph:
   (a) For every variable x we have a vertex x.
   (b) For every clause $(x \vee y)$ we have an edge (x, y).
3. We take the desired size of clique to be k.

We form the needed bijection: Let $\vec{b}$ be a satisfying assignment of φ that has $\geq k$ variables set to FALSE. We map $\vec{b}$ to the set of vertices x such that variable x was set to FALSE. Since every edge (x, y) has a corresponding clause $(x \vee y)$, we cannot have both x, y in the set. Hence we have mapped to an independent set of size $\geq k$.

We leave it to the reader to show that every independent set of size $\geq k$ is in the image of the map.

We omit the remaining three parts of this theorem and leave them to the reader. □

Exercise 12.31. We state five #P-complete problems and give references. Either try to prove they are #P-complete or look up the papers, read them, understand the reductions, and put it in your own words.

- Counting the number of vertex covers of size $\leq k$ in a bipartite Vertex Cover. This was shown by Provan and Ball [PB83]. They also showed several variants of this problem are #P-complete.

- Minimum Cardinality (s, t) Cut: Given (G, s, t) where G is a graph and s, t are vertices, find how many minimum-size edge cuts separating s and t. This was shown by Pravan and Ball [PB83].

- Antichain. Given partial order $(X, \preceq)$ find the number of antichains (sets with all elements pairwise incomparable). This was shown by Provan and Ball [PB83]. They also showed several variants of the problem are #P-complete.

- EXT is the problem of, given a partially ordered set (via the elements and the relations) determine whether there is an extension of it (s consistent way to order some pair that is not ordered). This problem is easily in P since a partial order is in EXT if and only if it does not compare every pair. But what about the problem of *counting* the number of extensions. Brightwell & Winkler [BW91] showed that this problem is #P-complete.

- Recall that 3COL is the problem of, given a graph G, determine whether it is 3-colorable. Creignou & Hermann [CH99] have shown that #3COL is #P-complete.

For more #P-complete problems see the papers of Valiant [Val79c, Val79a, Val79b] and Simon [Sim77].

12.11 Another Solution Problems (ASP-A)

Definition 12.32. Let $A \in \mathrm{NP}$. Hence there exists a polynomial p and a set $B \in \mathrm{P}$ such that

$$A = \{x \mid \exists y : |y| = p(|x|) \wedge (x, y) \in B.$$

Then ASP-A is the following problem: Given x and given a y with $|y| = p(|x|)$ and $B(x, y)$, determine whether another solution exists (ASP stands for Another Solution Problem). Note that you need not find that other solution.

This concept is useful in puzzle design since we want to be able to show that there is a unique solution.

There are cases where A is hard but ASP-A is easy. ASP-3COL is easily seen to be in P: given a 3-coloring of a G, just permute the colors to get another 3-coloring of G.

Open Problem 12.33. *Is the following problem NP-complete: Given a graph G and a 3-coloring ρ, find another 3-coloring that is not a permutation of ρ.*

We look at a more interesting example.

Let Cubic Ham Cycle be Ham Cycle restricted to cubic graphs (this is known to be NP-complete). Then we will see that ASP-Cubic Ham Cycle ∈ P.

1. Tutte [Tut46] proved that any cubic graph has an even number of Hamiltonian cycles. Hence if a cubic graph G has a Hamiltonian cycle then it has another one. The proof was nonconstructive and hence did not yield an algorithm to actually find the second Hamiltonian cycle. (Tutte credits Smith with an earlier and different proof of the theorem).

2. Thomason [Tho78] came up with an algorithm that will, given a graph G and a Hamiltonian cycle H, output another Hamiltonian cycle. He did not analyze it.

3. Krawczyk [Kra99] showed that Thomason's algorithm, in the worst case, takes exponential time.

Since we only want to know whether there is another Hamiltonian cycle, and do not need to find it, we have the following theorem.

Theorem 12.34.

1. *ASP-3COL is trivially in P since we can permute the colors.*

2. *Let Cubic Ham Cycle be the problem of finding a Hamiltonian cycle in a cubic graph. By the statements above ASP-Cubic Ham Cycle ∈ P trivially since the answer is always yes.*

Let $A, B \in$ NP. We define a reduction so that if $A \le_p B$ and ASP-$A \notin P$ then ASP-$B \notin$ P.

Definition 12.35. Let $A, B \in$ NP.

1. A is ***ASP reducible to B*** if there is a parsimonious reduction from A to B with the additional property that, given a solution for an instance of A, a solution for an instance of B can be computed in polynomial time.

2. B is ASP-complete if, for every $A \in$ NP, A is ASP reducible to B. (We do not have a notion of ASP-hardness since these notions only make sense if $B \in$ NP.)

Exercise 12.36. Prove that if A is ASP reducible to B then the following hold:

- ASP-$B \in P \implies$ ASP-$A \in$ P.
- ASP-A is NP-hard $\implies$ ASP-B is NP-hard.

Exercise 12.37. Show that all of the parsimonious reductions in this chapter were ASP-reductions. This leads to many problems being ASP-complete.

Exercise 12.38. Fillmat is a logic puzzle published by Nikoli. It was shown to be both NP-complete and ASP-complete by Uejima and Suzuki [US15]. Read their paper and either try to prove their theorems or read how they did and put it in your own words.

12.12 Fewest Clues Problem

Imagine that you are designing a hard Sudoku puzzle. You want to give as few numbers as possible yet still have a unique solution. Demaine et al. [DMS+18] have formalized this notion.

Let

Definition 12.39. Let $A = \{x \mid \exists^p y : (x, y) \in B\}$ where $B \in \mathrm{P}$. Let $q(|x|)$ be the size of y. Let $x \in \Sigma^*$. A ***clue for x*** is a string in $(\Sigma \cup \perp)^{q(|x|)}$ such that there is exactly one way to fill in the $\perp$-spots to get a y such that $(x, y) \in B(x, y)$.

> FCP(A)
> *Note:* $A = \{x \mid \exists^p y : (x, y) \in B\}$ where $B \in \mathrm{P}$.
> *Instance:* $x \in \Sigma^*$ and $k \in \mathbb{N}$.
> *Question:* Does there exist a clue for x with k non-$\perp$ characters?
> *Note:* For the function version of the problem you are given x and want to find the least k such that $(x, k) \in$ FCP(A).

Demaine et al. [DMS+18] showed that FCP versions of Planar 1-in-3SAT, Planar 3SAT, 1-in-3SAT are Σ_2-complete. They then used these results to get that the FCP version of several games problems are Σ_2-complete: Sudoku, Akari, Shakashaka (these are Nikoli games), Latin Squares.

12.13 Open Problems

Here is an ill-posed conjecture.

Conjecture 12.40. *If A is a natural NP-complete problem then #A is #P-complete.*

The conjecture is ill-posed because *natural* is not well defined. Even so, it seems to be true. So the open question here is to find some rigorous way to state it, and then prove it.

What if we drop the requirement that the problem A is natural. Since there can be rather contrived sets A we pose this as an open question instead of a conjecture:

Open Question Prove or disprove the following: If A is an NP-complete problem then $\#A$ is NP-complete.

We have also seen that there are sets $A \in \mathrm{P}$ such that $\#A$ is #P-complete. There are also sets $A \in \mathrm{P}$ such that $\#A \in \mathrm{FP}$.

Research Project For many sets $A \in \mathrm{P}$ determine if $\#A$ is #P-complete. Try to formulate (1) criteria on A that makes $\#A$ #P-complete. (2) criteria on A that makes $\#A \in \mathrm{FP}$. One must be careful here since there may not be a natural or a unique function to call $\#A$.

We have seen that if A os NP-complete then ASP-A might be in P, NP-complete, ASP-complete, or even both NP-complete and ASP-complete.

Research Project For many sets A that are NP-complete determine if ASP-A is in P or ASP-complete or NP-complete or both ASP-complete and NP-complete. Try to formulate criteria on A that puts in each of those classes.

Chapter 13

PSPACE-Hardness via QSAT

13.1 Overview

In Chapter 4, we discussed two metatheorems, due to Viglietta [Vig14], that allowed us to guide us to proofs that some problems were NP-complete. In this chapter we discuss more metatheorems due to Viglietta (from the same paper). These metatheorems will be used as general techniques for proving hardness. We will cover 2 main metatheorems (and some different versions of them): one for NP-hardness and one for PSPACE-hardness. They can be applied to a lot of games. We will use the term metatheorem in somewhat vague sense and not a formal theorem, because it's hard to state all the assumptions for all games. It will give a general set up for the proof.

13.2 Definitions

Viglietta defines an "avatar" in his proofs, but for our case, we will use the term "player." The player is the character in the game that we can control and move around to complete objectives. One basic assumption we make about the player is that we can choose, at any time, to change the player's direction of movement.

13.3 PSPACE

PSPACE is the set of problems solvable in polynomial space.

Exercise 13.1. Prove the following.

1. NP ⊆ PSPACE.

2. PSPACE ⊆ EXPTIME.

3. PSPACE = NPSPACE.

13.3.1 PSPACE-Complete Problems

In order to show that a problem is PSPACE-hard, we need a set of problems known to be PSPACE-complete (to reduce from). The classical problem is to simulate a polynomial space algorithm (or simulate a linear space Turing Machine). This problem is not very useful for hardness proofs.

We present a set of simpler problems.

Quantified Boolean Formulas (QBF) and Variants
Instance: For QBF a quantified Boolean Formula. It may begin with either a $\exists$ or a $\forall$. We will take it to be of the form

$$(Q_1 x_1) \cdots (Q_k x_k)[\varphi(x_1, \ldots, x_k)]$$

where the Q_i's alternate between the two types of quantifiers. For Q2SAT, φ is in 2-CNF form. For Q3SAT, φ is in 3-CNF form. Other variants can be defined and we will freely use them.
Question: Is the quantified Boolean formula true?
Note: We will also use the abbreviation QSAT.

Exercise 13.2.

1. Show that Q2SAT $\in$ P.

2. Show that Q3SAT is PSPACE-complete.
   Hint: Given a polynomial-space-bounded Turing machine M and an input x you need a QBF φ such that $M(x)$ accepts if and only if φ is true. Construct a sequence of QBF's $\varphi_0(a,b)$, $\varphi_1(a,b), \ldots, \varphi_L(a,b)$ (we leave it to you to figure out L) such that φ_i is satisfiable if and only if there is a way for M to go from configuration a to configuration b in time $\le 2^i$.

13.3.2 Schaefer-Style Dichotomy Theorem

Recall that Schaefer [Sch78] (our Theorem 1.19) proved a dichotomy theorem which states exactly which types of SAT problems are in P and which are NP-complete. Schaefer stated a dichotomy theorem for QSAT but did not provide a proof. Twenty three years later Creignou et al. [CKS01] proved the theorem Schaefer stated. For other interesting variants on the issue of dichotomy for QSAT see the papers of Hemaspaandra [Hem04] and Dalmau [Dal99].

We now present the dichotomy theorem for QSAT. Note that QSAT problems are to determine whether

$$(Q_1 x_1) \cdots (Q_k x_k)[\varphi(x_1, \ldots, x_k)]$$

is true. The different QSAT formulas are based on the different forms that φ can have.

Theorem 13.3.

1. *QSAT $\in$ P if and only if φ is Horn, Dual Horn, Q2SAT, or X(N)OR*

2. *QSAT is PSPACE-complete otherwise.*

As with regular kSAT, the planar versions of QSAT are also hard: we can start with the same crossover gadget as seen in the proof of Theorem 2.9 that forces some variables to be the same, and then also have it create new variables that must also be quantified. We can do this by adding a $\exists x_i$ for every newly created variable x_i.

Notably, Planar 1-in-3QSAT is still hard, and Planar NAE-3QSAT is still easy.

13.4 Metatheorem 3

(Metatheorems 1 and 2 are in Sections 4.8.1 and 4.8.2 respectively.)

Convention 13.4. When we say that a game is PSPACE-complete (or any other complexity) we mean that determining who wins is PSPACE-complete. This also applies to 1-player games (puzzles) where the question is can the one player complete the task.

The third metatheorem from Viglietta's paper [Vig14] states that a game where you have a player who needs to traverse a planar graph from start to finish, and has door and pressure plate objects, is also PSPACE-complete. A door can be considered an edge that exists only if a certain condition is met. There are two types of pressure plates — an "open" pressure plate which satisfies the condition of the door, and a "close" pressure plate which causes the door condition to not be satisfied.

(Keep in mind that these pressure plates simply cause the door to open or close, but do not require constant pressure to keep them in that state, i.e., unlike Portal's door mechanics.)

13.4.1 Reduction from Q3SAT

The idea is as follows:

- We start at the position labeled "start," and continue along the path.
- Wherever there is an $\exists$ quantifier, we fix a value for that variable.
- Wherever there is a $\forall$ quantifier, we will check both possible values by:
  1. Traversing towards the clauses (top branch in the slide 9 diagram), we remember that we have seen this variable once, and set its value to true.
  2. Traversing back from the clauses (bottom branch in the diagram), we check to see if this variable is true: if true, set to false and loop again; if false, continue along the branch.

Remember that we will continue along the bottom only if the quantifier is satisfied; e.g. if one value fails to satisfy the formula for a $\forall$ quantifier, the second loop is no longer necessary, as we already know that the $\forall$ quantifier cannot be satisfied.

Now, we let a door's state determine the value of its *literal*: an open door indicates that its represented literal is selected. Thus, for a variable x, we need two doors for it: x and $\neg x$. Note that, when we say an x door, we actually mean all doors labeled by x such that an x-open switch opens all the x doors, and so on for closing and $\neg x$.

Then, a clause is a hall with three doors possible to the next hall, one for each literal; if any of the three doors is open, the clause is considered to be satisfied.

Finally, we need to come up with our quantifiers:

- Existential quantifier: two branching paths, that mutually close each other, and turn the variable on or off

- Universal quantifier: snake-like path, with middle gate that "counts" how many times we've gone through

The clause gadget and the quantifier gadgets are both diagrammed in slide 10. One key point to note about the quantifier gadgets is that they prevent deadlock by requiring the path to open one door contain pressure plates to close the other doors such that the progression must move forward.

Note that, by construction, a solution will take exponential time: at least $O(2^U)$ time for U universal quantifiers. However, the reduction process should still take polynomial time.

13.4.2 First Player Shooter (FPS) Games

Metatheorem 3 allows many FPS games such as Quake to be easily proved PSPACE-complete, by simply designing the maps and puzzles using pressure plates and doors as described in the reduction above.

13.4.3 Role Playing Games (RPG)

Additionally, Metatheorem 3 allows one to prove many RPG games (such as Eye of the Beholder) to be PSPACE-complete, using the same idea of designing the maps and puzzles correctly.

13.4.4 Script Creation Utility for Maniac Mansion (SCUMM) Engine

Many adventure games can also be proven PSPACE-complete in this way. One rather large categorization of such games – the SCUMM engine – can be shown to be PSPACE-complete allowing all games that use it to be shown to be as well. Some SCUMM engine-based games are Secret of Monkey Island, Maniac Mansion, Space Quest IV.

13.4.5 Prince of Persia

In this game, the player is given the ability to jump. So, in order to ensure that pressure plates are always pressured when the player passes over that tile, we put the pressure plate on top of a very high wall, such that the player can only jump that high, so the player must touch the pressure plate (which is important to ensure that the player must make progress in the game).

With this adjustment, one can show that Prince of Persia is PSPACE-complete.

13.5 Metatheorem 4

Metatheorem 4 by Viglietta [Vig14] expands upon Metatheorem 3 to allow the use of buttons instead of pressure plates. These buttons do not need to be pressed and open/close 3 doors at once (and in fact it was recently shown that 2 doors at once will work, but this has not yet been published).

The reduction for 3 doors at once involves treating a button as 3 pressure plates put together.

13.5.1 Examples

Some examples of games that fall under this pattern are Sonic the Hedgehog, The Lost Vikings, and Tomb Raider. With this adjustment, they can be shown PSPACE-complete.

13.6 Doors and Crossovers

Another metatheorem, by Aloupis et al. [ADG14, ADGV15], is a further generalization that relies on doors and crossovers to show PSPACE-hardness.

To do this, we show that we can use doors and crossovers to create an environment that provides the following three types of paths: a traverse path (whereby the player can only pass if the door is opened), an open path (which allows the player to open the door) and a close path (which forces the player to close the door).

This result can be found in the paper of Aloupis et al. [ADG14, ADGV15] which uses it to analyze various Nintendo games.

13.6.1 Legend of Zelda: A Link to the Past

In Legend of Zelda, we create a traverse path with just a labeled door: if the labeled door is open, then we can traverse; if not, then we can't.

One interesting caveat with Legend of Zelda is that the game has only toggles: it opens all the connected doors if they're closed, and closes them if they were open. This means that the open and close paths are somewhat trickier: the general idea is for the open and close paths to lead to directed teleporters to the appropriate halls (seen at the bottom of slide 19). Then, to toggle the door, we have to traverse from the direct that we want to toggle to, and then hit the toggle in a secluded room, for which the only way out is through the directed teleporter right next to it, which takes us back to a hall.

Since we want to initialize all doors to the closed state, we need to create an initializer for this purpose; basically, we just create a chain of gadgets that will toggle all the doors to be closed. At the very end, we have a crystal that can be broken, which will active the inactive toggles, and deactive the active toggles. There will be only one crystal, and it serves the purposes of destroying our initializing paths and enabling a one-way gadget. To make sure that it doesn't appear at the wrong place, we ensure that it appears only between the initializer and final traversal gadgets.

13.6.2 Donkey Kong Country 1, 2 and 3

Donkey Kong Country is another Nintendo series of games that can be shown to be PSPACE-complete using doors and crossovers. Unfortunately, DKC 1, 2, and 3 have mutually exclusive "features," so we will have to address each one individually.

Donkey Kong Country 1

In the Donkey Kong Country games, there are bees. If you touch a bee, you die, so the goal is to not die. In Donkey Kong Country 1, there is a tire object: if you land on the tire, you bounce back up. For a traversal, we use a barrel shot straight down, so landing on a tire would cause us to be stuck in the game (and thus not win).

Furthermore, we will introduce stationary bees as obstacles, and moving bees (highlighted in red on slide 21 – not to be confused with actual red bees in the game), which move in a deterministic pattern, demonstrated by the arrows.

To open the door, we come in from the open path, jump across the ledge, and push the tire just up the hill so that it doesn't roll back down. Note that the giant box of bees that are labeled "open" move in a left-down-right-up pattern, so that after pushing the tire up the hill, we can still run back down through the traversal path.

To close the door, we come in from the close path, and push the tire down the hill with a slight nudge, and scramble up a rope to exit. One caveat here is that, in the real game, we'd either have to make the bees slower, or make the climbing faster.

With these adjustments one can show that Donkey Kong Country 1 is PSPACE-complete.

Donkey Kong Country 2

In Donkey Kong Country 2, instead of tires, we have balloons and air currents (the gray steam near the bottom of the map). The balloons float on top of the air currents, and in order to traverse, we have to move it out of the way of the traversal path.

To do this, from the open end, we can jump on the balloon and move it away from the air currents in the middle so that it drops down to point A. We then escape through the entrance of the open path.

To close, we enter from the close path, drop onto the balloon, and move it back into the current before using the barrels to propel ourselves out of the region.

With these adjustments one can show that Donkey Kong Country 2 is PSPACE-complete.

Donkey Kong Country 3

In Donkey Kong Country 3, instead of tires and balloons, we have tracking barrels, which the player can slide around sideways once the player land in one, and it always shoots up. The caveat is that the barrel will follow the player if the player tries to jump out of it, so the tracking barrel effective prevents the player from going down.

Thus, the open state is for the barrel in the big gap of slide 23 to be on the left: if the player enters to traverse, the player can drop into the barrel, get shot up, and exit through the traversal path.

To close, we enter from the close path, jump into the barrel, and slide it all the way to the right. Since the barrel tracks us wherever we go after we land in it, we are guaranteed that, if we leave through the close path, the barrel must be on the right.

To open, then we just drop into the barrel from the open end, slide to the left, and shoot ourselves out through the open path.

With these adjustments one can show that Donkey Kong Country 3 is PSPACE-complete.

13.6.3 Super Mario Bros

Super Mario Bros is also PSPACE-hard. Using the diagram on slide 24, we see that the traversal path is on the left, the open path is on the bottom, and the close path is the entire right side.

The intuition behind the setup is: we need something that the player can't pass through, but an obstacle can. So, we break two rules in the game itself and have (1) a length-1 firebar (a.k.a. a fireball) to separate the traversal and close paths, and a spiny to block a path.

To close, we enter from the close side, and if the spiny is on the right, then we go under the spiny, and at the right time, we knock up up and over the fireball to the other side, so that we can traverse the close path.

To open, we enter from the open side, knock the spiny over to the right, and leave again.

Then, when we traverse, we can traverse if and only if the spiny isn't on the traverse side.

13.6.4 Lemmings

Lemmings is an old real-time strategy game where the player is in control of a bunch of Lemmings, and can imbue any Lemming with jobs. For our purposes, we will have two jobs: a builder (to build bridges) and a basher (to cut through dirt, but not steel).

Although many results for Lemmings exist, we will show PSPACE-completeness using an exponential amount of builders, bashers, and time.

The standard Lemming is a Walker, and its mechanics are documented on slide 28: in brief, Walkers can climb up out of ditches but only if they're not too tall.

One kind of special Lemming that we will use is a Basher, who can cut through dirt but not steel. The mechanics are documented on slide 29, and a Basher will stop bashing once the steel check finds steel, or a solidarity check finds no more dirt.

What is interesting to note is that disparity between the steel and the solidarity checks.

The last kind of special Lemming that we will use is a Builder, who can build bridges over gaps (otherwise the Lemmings will die from falling between the gaps). The mechanics are documented on slide 30, and again, there are solidarity checks for where to lay the bridge pieces, and whether the Lemming can keep going forward and up, by using a solidarity check near the head.

In order to apply the doors and crossovers metatheorem, we need to show the primitive paths that we will use: a rising path, a falling path, and rise-and-double-back path, and fall-and-double-back path, and a crossing path. To prevent Lemmings from effectively running away, we place steel pretty much everywhere that isn't marked with a red crossed-box or a gray platform on slide 31.

Now, we will need a fork (a.k.a. crossover) gadget, to allow for traversals of two different paths. This is achieved by having a gap for lemmings to fall through: if they fall through, they'll go through one paths; if they build a bridge instead, they'll go through another path (which also requires a Basher to clear some obstacles, which are placed to prevent builders from going crazy and building a stairway to heaven).

Finally, our door gadget will be represented as such on slide 33: an open door will have a gap in the middle, while a closed door will have a bridge in the middle that will prevent traversal from the top to the bottom. To open the door, we bash the bridge apart using a Basher; to close the door, we layer a bridge on top of the gap using a Builder. Since we have a limited (but exponential) number of Builder and Basher upgrades available to us, we must use each one wisely.

Thus, with our door and crossover gadgets, we have shown that Lemmings is PSPACE-hard.

13.7 Stochastic Games

In addition to standard 2-player games, we can also consider Stochastic games, where one of the players plays randomly. For Bounded 2-player stochastic games, we ask whether one player can force a win with a probability greater than $\frac{1}{2}$. This question is PSPACE-complete.

More formally, we have Stochastic SAT which asks

$$\exists x_1 : \mathcal{R}x_2 : \exists x_3 : \cdots : \Pr(Win) > \frac{1}{2}$$

where we use $\mathcal{R}x_2$ to denote a randomly chosen x_2. Interestingly replacing random choice with a for all choice seen in standard 2 player games doesn't change the hardness.

13.8 Other Kinds of Games

So far we have been focused on 1 player games, or puzzles. Instead, we will be focusing on 0-player games, or simulations, and 2-player games. In each case, if the game is on a bounded board, then it will either take an exponentially bounded number of moves (unbounded) or a polynomially bounded number of moves (bounded). Generally speaking, these games are in the complexity classes given by Figure 13.1. We will look at hardness results for several games in particular, including

- Various SAT games, all 2-player games with a bounded number of moves
- Reversi/Othello, a 2-player game with a bounded number of moves,

All of these games are in fact PSPACE-complete.

13.9 Open Problems

1. Define and analyze generalizations of Nine-Men-Morris, Quoridor, and other games (check to see if they have already been analyzed). Then determine the complexity of these versions. You should also look at variants of these games.

2. (This is a research program.) Most of the results on the hardness of games do not use the game as it is actually played. (See Biderman [Bid20] for aa possible exception.) For example,

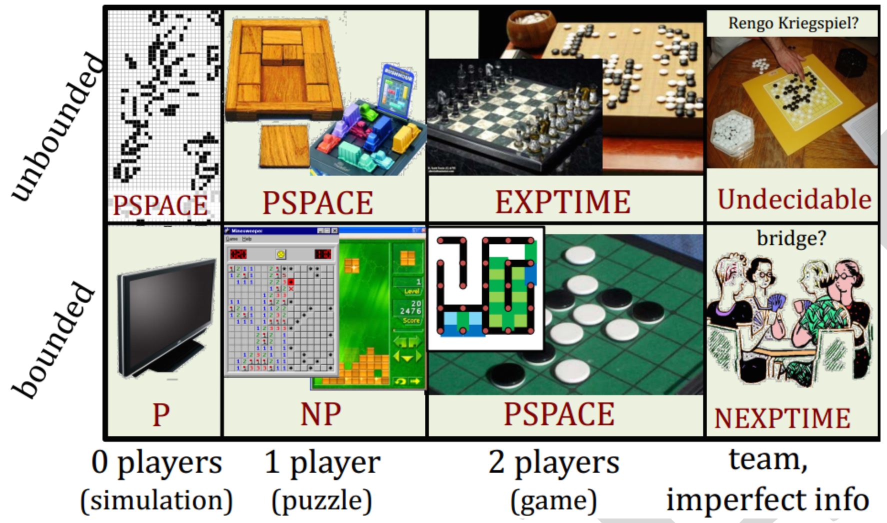

unbounded
bounded
PSPACE
PSPACE
EXPTIME
Rengo Kriegspiel?
Undecidable
P
NP
PSPACE
bridge?
NEXPTIME
0 players
(simulation)
1 player
(puzzle)
2 players
(game)
team,
imperfect info

Figure 13.1: A table showing the complexity classes for each type of game, with bounded and unbounded referring to whether the number of moves is polynomially bounded or not.

Chess and Checkers are played on an 8×8 board, not an $n \times n$ board. Develop a framework for the complexity of games that can be used to show that a game, as it is actually played, is hard.

Chapter 14

Beyond PSPACE But Decidable

14.1 Introduction

In this chapter we will examine problems whose complexity is beyond PSPACE. They will all be games: given a game position, which player will win (if they both play perfectly).

We will not define "game" formally, however we will define parameters of games and give examples which are also summarized in Figure 14.1.

Definition 14.1.

1. A game is ***bounded*** if the number of moves is bounded by a polynomial in the board size. A game is ***unbounded*** otherwise.

2. A game has ***perfect information*** if there is no hidden information. A game has ***imperfect information*** otherwise.

3. A 0-player game is a simulation. The ***game of life*** which we will study in Section 15.2 is such a game. These always have perfect information. Bounded 0-player games tend to be in P where as unbounded ones tend to be NP-complete. The game of life provides examples of both bounded and unbounded problems.

4. A 1-player game is a puzzle. Jigsaw puzzles, packing puzzles, and all of the other puzzles in Chapter 6 are 1-player games with perfect information. The bounded versions of 1-player games with perfect information tend to be NP-complete whereas the unbounded versions tend to be PSPACE-complete.

5. A 2-player game is just what you think it is. Chess is an example of a 2-player game with perfect information.

6. A multiplayer game is a game of 3 or more players. Bridge is a multiplayer game of imperfect information. Bounded multiplayer games tend to be NEXPTIME-complete whereas unbounded multiplayer games tend to be undecidable. Rengo Go is a team version of Go (1) where white has 2 players (2) black has 2 players, (3) the player on each team alternate play, but (3) the players on a team cannot communicate. Rengo Go is believed to be undecidable.

	0 players	1 player	2 players	Team, imperfect information
Unbounded	PSPACE	PSPACE	EXPTIME	CE (Undecidable)
Bounded	P	NP	PSPACE	NEXPTIME

Figure 14.1: Natural complexity classes for computations with varying numbers of players and length of computation. (Based on [HD09, Figure 1.1].)

Team games with imperfect information fall into one of two categories – games that have a bounded number of moves can be shown to be in NEXPTIME, while games with an unbounded number of moves are undecidable in general. In most situations above, an "unbounded" number of moves is still bounded by the number of board states, which is exponential. In this setting, however, we exploit that board states might repeat, which might make for a superexponential number of moves.

There are no known "natural games" in either category, but Bridge is team game with imperfect information and a bounded number of moves (all players must play their cards at a turn until there are no cards remaining), and is conjectured to be in NEXPTIME. Similarly, Rengo Kriegspiel (blind team Go) is a team game with imperfect information and an unbounded number of moves and is conjectured to be undecidable.

Finally, we consider bounded 0-player games, such as Game of Life run for a polynomial number of moves. These simulations are in P, but also provide a notion of P-completeness, and many games can be reduced to the Game of Life simulation. This is interesting because there is no way to parallelize a simulation of Game of Life.

Figure 14.1 shows which complexity classes these different types of games tend to reside.

14.2 SAT Games

Consider the following game:

Definition 14.2. The ***original SAT game*** is as follows.

1. The board is a Boolean formula $\varphi(x_1, \dots, x_{2n})$.
2. For $i = 0$ to $n - 1$
   (a) Player I sets x_{2i+1} to $b_{2i+1} \in \{\textsc{true}, \textsc{false}\}$
   (b) Player II sets x_{2i+2} to $b_{2i+2} \in \{\textsc{true}, \textsc{false}\}$
3. If $\varphi(b_1, \dots, b_n) = \textsc{true}$ then I wins, else II wins.

This is an example of a "SAT game". Determining who wins is PSPACE-complete (this follows from QBF being PSPACE-complete). Stockmeyer and Chandra [SC79] introduced many variants of this game and showed they are EXPTIME-complete. They used these EXPTIME-complete problems to prove that other problems are EXPTIME-complete. Since P ≠ EXPTIME (by a simple diagonalization argument) any EXPTIME-complete problem is *provably (!)* not in P. We do not need to condition on P ≠ NP or any other assumption.

The games they consider are unbounded in that they could go on forever. This happens since (1) in some games you can change the value of a variable, and (2) in some games a player can pass.

All of the games have the following:

1. The game begins with one or two Boolean formula and a partial assignment (which might not set any variables).
2. The players are RED (who goes first) and BLUE (who goes second).
3. Some of the variable are RED, some of the variables are BLUE, and some of the variables are neither. Only the RED (BLUE) player can set the RED (BLUE) variables.
4. These variables can then be set to TRUE or FALSE. In some cases they can be reset.
5. A player may be allowed to pass.

Stockmeyer and Chandra [SC79] proved the following theorem.

Theorem 14.3. *For the following games, determining which player will win is EXPTIME-complete.*

1. *G1:*

   Board: *A 4CNF formula with variables RED, BLUE, and one x which is neither, and a truth assignment for the RED and BLUE variables, but not x.*

   Move: *A move of RED (BLUE) consists of setting all the RED (BLUE) variables and also setting x to TRUE (FALSE).*

   Passing: *Not allowed.*

   Win Condition: *If after a player's move the formula is FALSE then that player loses.*

2. *G2:*

   Board: *Two 12DNF formulas with all variables RED or BLUE, and a truth assignment for all the variables. One formula is itself called RED, the other BLUE.*

   Move: *A move of RED (BLUE) consists of changing ≤ 1 RED (BLUE) variable*

   Passing: *Allowed since you can opt to change 0 variables.*

   Win Condition: *If, after RED (BLUE) moves, the RED (BLUE) formula is TRUE, then RED (BLUE) wins.*

3. *G3:*

   Board: *Two 12DNF formulas with all variables RED or BLUE, and a truth assignment for all the variables. One formula is itself called RED, the other BLUE.*

   Move: *A move of RED (BLUE) consists of changing one RED (BLUE) variable*

   Passing: *Not allowed.*

   Win Condition: *If, after RED (BLUE) moves, the RED (BLUE) formula is TRUE, then RED (BLUE) loses.*

4. *G4:*

   Board: *A 13DNF formula with all variables RED or BLUE, and a truth assignment for all the variables.*

   Move: *A move of RED (BLUE) consists of changing* ≤ 1 *RED (BLUE) variable*

   Passing: *Allowed since you can opt to change 0 variables.*

   Win Condition: *If, after RED (BLUE) moves, the formula is* TRUE*, then RED (BLUE) wins.*

5. *G5*

   Board: *An (unrestricted) formulas with all variables RED or BLUE, and a truth assignment for all the variables.*

   Move: *A move of RED (BLUE) consists of changing* ≤ 1 *RED (BLUE) variable*

   Passing: *Allowed since you can opt to change 0 variables.*

   Win Condition: *RED wins if the formula* ever *becomes true.*

6. *G6: Identical to G5 except that the formula has to be in CNF form.*

We mention one more type of game, due to Papadimitriou [Pap85]

Definition 14.4. ***Stochastic SAT*** is the following 1-player game.

1. The board is a Boolean formula $\varphi(x_1, \ldots, x_{2n})$.

2. For $i = 0$ to $n - 1$

   (a) The Player sets x_{2i+1} to $b_{2i+1} \in \{\text{TRUE}, \text{FALSE}\}$.

   (b) x_{2i+2} is set to $b_{2i+2} \in \{\text{TRUE}, \text{FALSE}\}$ randomly.

3. If $\varphi(b_1, \ldots, b_n) = \text{TRUE}$ then the Player wins, else he loses.

Theorem 14.5. *Given a Boolean formula, determining whether the Player has a strategy so that their probability of winning is over* $\frac{1}{2}$ *is PSPACE-complete.*

The stochastic SAT game can be viewed as the original SAT game (Definition 14.2) but with Player 2 replaced by a random player. As such it is interesting that the original SAT game and the Stochastic SAT game are both PSPACE-complete.

14.3 Games People Don't Play

All of the results in this section are due to Stockmeyer and Chandra [SC79]. In each of the games one player is RED and the other is BLUE. RED goes first. The complexity of a game is the complexity of, given a position in it, determine which player wins.

14.3.1 Peek

Definition 14.6. Peek is the following game: The setup is a stack of plates with holes, where one hole is fixed. Each plate is either RED or BLUE and can be either in or out. Each turn, a player can move a plate of his color or pass. (see Figure 14.2). The player who gets a hole all the way through the plates wins.

Theorem 14.7. *G4 $\leq_p$ Peek, so Peek is EXPTIME-complete.*

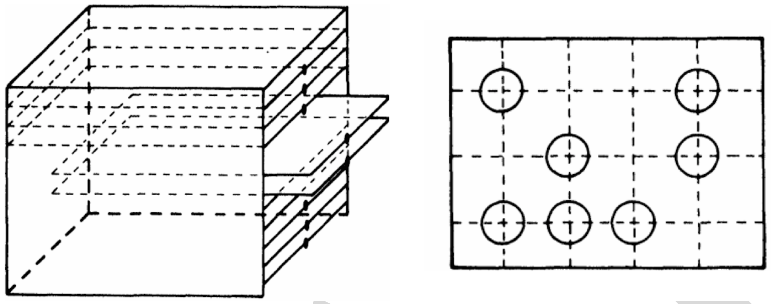


Figure 14.2: Left: A Stack of Plates. Right: An Example of a Plate.

14.3.2 HAM

Definition 14.8. Ham Games is a game where we start with a simple, undirected graph where each edge is colored either RED or BLUE. In addition to a color, each edge also has a state - in or out. Each turn, a player must toggle the state of an edge of his color. Red wins if at any point in the game, the edges that are IN form a Hamiltonian cycle.

Theorem 14.9. *G6 $\leq_p$ Ham Games, hence Ham Games is EXPTIME-complete.*

14.3.3 Block

Block is another graph game.

Definition 14.10. Block starts with 3 graphs over the same vertices. Some of these vertices contain tokens that are either RED or BLUE. A vertex can have at most one token. Each turn, a player must slide a token of his color along any path in one of the 3 graphs, as long as the target vertex and all intermediate vertices on the path don't have any tokens. Each player i has some set of "victory vertices" W_i. If a player can move one of his tokens to a vertex in his set of victory vertices, he wins.

Theorem 14.11. *G3 $\leq_p$ Block, so Block is EXPTIME-complete.*

Proof sketch:

Figures 14.3 shows the variable gadgets for both players (white left, black right). Stars are winning vertices, and the dashed, dotted, and solid lines represent edges in each of the graphs. If either player deviates from setting variables, the other can instantly win.

Figure 14.4 shows the gadget for the formula $\overline{x_3} \wedge y_5$. Once white activates by moving up, both black and white are forced to move up one at a time - or else the other player wins instantly. If x_3 and $\overline{y_5}$ are blocked, then white can't move up, in which case black wins and white should not have activated the formula.

∎

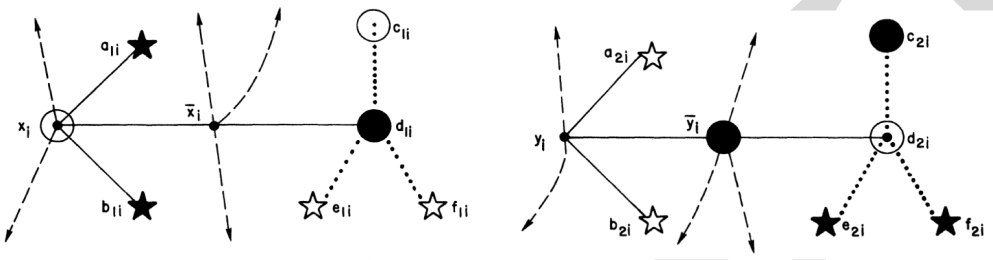


Figure 14.3: Variable Gadgets

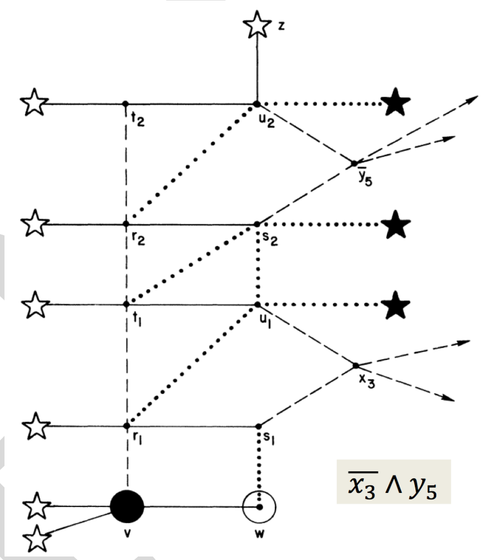


Figure 14.4: Gadget for the formula $\overline{x_3} \wedge y_5$

14.4 Games People Play

We look at the complexity of Checkers, Chess, and Go on an $n \times n$ board. A statement of the form "CHECKERS is BLAH-complete" means that the problem of, given a position, which player wins (the one whose turn it is, or the other one) is BLAH-complete. We allow the board to be $n \times n$. CHECKERS mate-in-1 is the problem of determining whether the player who is about to go and make one move that wins the game. Similar for other problems of the form GAME mate-in-one.

We will also look at slight variations on the ruleset that make these games even harder than EXPTIME.

14.4.1 Checkers

Questions about Checkers run the gamut between P and EXPTIME-time complete.

1. Checkers mate-in-1 seems to involve a large number of jumps; however, Fraenkel et al. [FGJ+78] showed that deciding whether a Checkers position is a mate-in-1 is in P. They reduced it to that of finding an Eulerian path on a graph, which is in P.

2. Bosboom et al. [BCD+18] showed that (1) deciding whether there is a move that force the other players to win in one move is NP-complete, (2) Checkers where jumping is mandatory is PSPACE-complete, and (3) cooperative versions of (1) and (2) are NP-complete.

Robson [Rob84] showed the following:

Theorem 14.12. *G3 $\leq_p$ Checkers, hence Checkers is EXPTIME-time complete.*

Proof sketch:
In the proof we exploit the fact that a player must capture a piece when it is possible to. The board is split into 2 regions - an inner region where the clauses and variables are represented, and an outer region which can be triggered by a player to win (Figure 14.5).

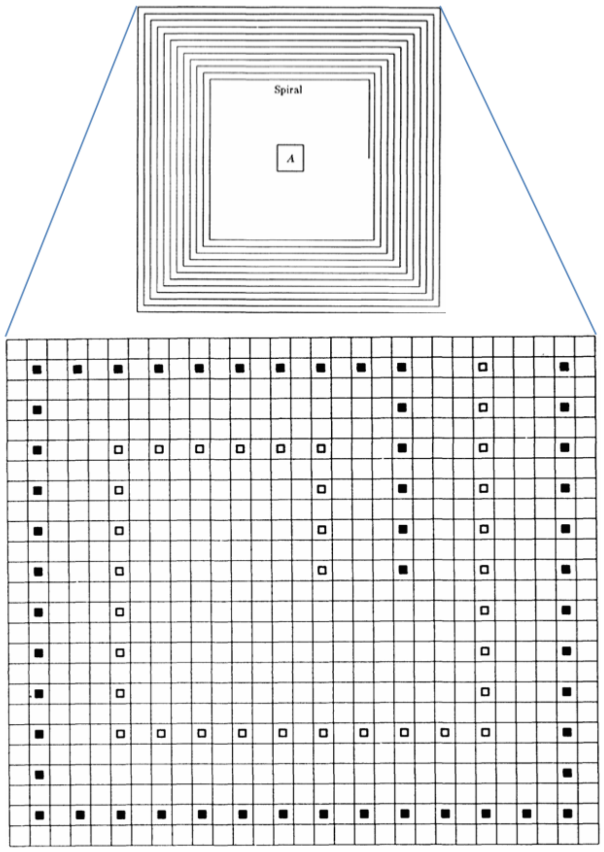


Figure 14.5: The Spiral Set Up Around the Variables and Clauses

Initially, players start by moving their own Kings between either a true or a false position. If at any point, a player satisfies their own DNF clause represented by pieces in the middle, the other player can activate an attack. A successful attack results in "free moves" - where the opponent has a configuration that requires a capture, but the other player does not. After accumulating enough free moves, the player can maneuver his pieces in the outer spiral where the opponent is forced to jump into a position where all of his pieces in the spiral can be taken in one jump in the next turn. This gives enough of a material advantage to guarantee a win. ∎

14.4.2 Chess

In traditional Chess there are some limits on how long the game can go. For example, if the same position is repeated three times then the game is a draw (it's more complicated than that and depends on which chess federation you are playing in, but we ignore that). In this bounded version, Chess is PSPACE-complete — but without this rule, we obtain an unbounded game and Fraenkel and Lichtenstein [FL81] showed EXPTIME-completeness:

Theorem 14.13. *Let* Chess *be chess with no rule to limit the length of a game, and only using pawns, bishops, and kings. Then G3 $\le_p$* Chess*. Hence* Chess *is EXPTIME-complete.*

14.4.3 Go

In the Japanese ruleset positions are only not allowed to immediately repeat. Robson [Rob83] showed the following:

Theorem 14.14. *G6 $\le_p$* Go*, if* Go *uses Japanese rules. Hence this version of* Go *is EXPTIME-complete.*

Note that, with Japanese rules, Go is in EXPTIME since we only need to compare two states at a time.

14.4.4 Phutball (Philosopher's Football)

We discuss a game that may or may not be Beyond PSPACE. We suspect that it is, which is why its in this chapter.

Berlekamp et al. [BCG82] devised the game Phutball which is short for ***Philosopher's Football***. Phutball is a 2-player game played with black and white stones on a square lattice (or Go board). Players move by either adding another black stone to the board or moving the white stone by jumping over contiguous groups of black stones and removing them. The goal is to move the white stone to one of the goals at the end of the board.

We discuss both the mate-in-one question, and the who-wins question. It is somewhat surprising that, in contrast to Checkers, the mate-in-1 question is hard.

Phutball
Instance: A position in the game Phutball.
Question: There are two Questions.

1. Is there a mate-in-1? We denote this problem Mate-In-1 which, in this section, will only refer to mate-in-1 for Phutball.
2. Can the player whose move it is win? We denote this problem Phutball.

Demaine et al. [DDE00] showed the following:

Theorem 14.15. *Mate-In-1 is NP-hard.*

Proof sketch: The reduction lays out variables and clauses on a large 2-dimensional board, with variable choices along the left and right edge and clauses along the top and bottom. The player makes a series of long horizontal jumps to choose which variables to set to true, and is able to traverse top to bottom if and only if none of the crucial squares have been cleared. ▮

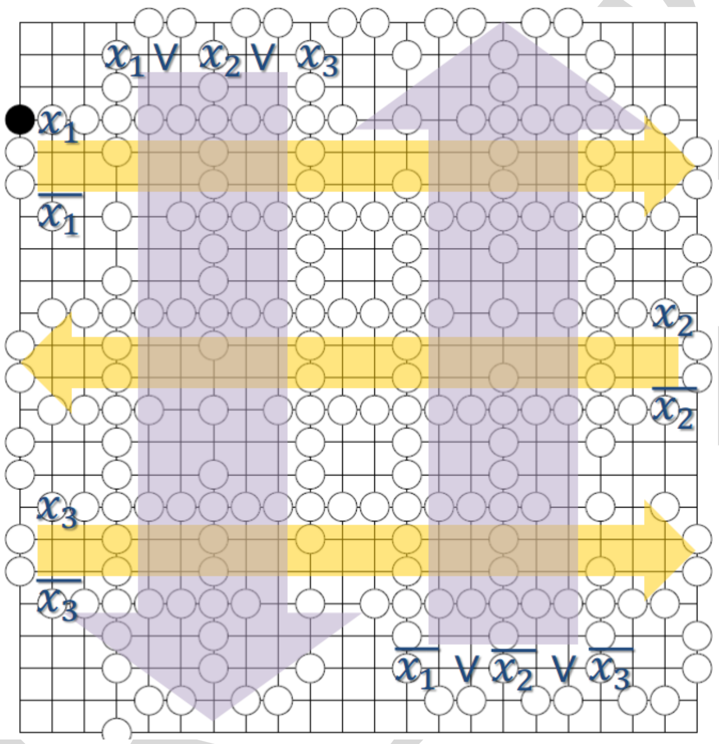


Figure 14.6: Here clauses are traversed by going top to bottom or bottom to top. Variable values are set by going left to right and vice versa.

Why is Mate-In-1 NP-hard for Phutball and in P for Checkers? The key is that one move in Phutball can comprise a large number of individual jumps.

What about determining who wins? Dereniowski [Der10] showed the following

Theorem 14.16. *Phutballis PSPACE-hard.*

The problem Phutballis in EXPTIME; however, the question of whether Phutballis EXPTIME-complete is still open.

14.4.5 Hard Variants of Games People Play

The results on Checkers, Chess, and Go were for the versions of the game where the rules allow the game to go on forever. If we add some additional rules that prohibit this then the games get harder.

The No-Repeat Rule. This rule makes a player lose if they ever repeat a past game configuration. With the addition of this rule, G1, G2, G3 become EXPSPACE-complete. So, Chess and Checkers also become EXPSPACE-complete. However, it's still an open problem whether Go with no-repeat (known as the superko rules) is EXPSPACE-complete. The intuition here is that we have to track and compare against all past game states.

The Conditional No-Repeat Rule.

Definition 14.17. A problem is in 2EXPTIME if it is in time $O(2^{2^{n^k}})$ for some k.

We introduce two special variables x and y. A player now loses if they repeat a past game configuration and at most 1 of x and y have changed since that configuration was played. Adding this rule makes G1 2EXPTIME-complete. Here, the intuition is that we have to track x and y temporarily, in addition to all past game states.

Private Information Games. In this variation, you can see some, but not all of an opponent's state. This makes G1 with 5DNF and G2 with DNF, 2EXPTIME-complete. An example of this game is Peek with a partial barrier obscuring the state of some of each players plates to the other player.

Blind Games. Here, Player 1's entire state is hidden from Player 2. This makes G2 with DNF EXPSPACE-complete. An example of this kind of game is Peek with a full barrier.

14.5 Further Results

We list problems that are complete in classes that are likely above PSPACE. For games listed the problem is

given a position in the game, which player wins?

1. Chinese Checkers, and other pebble games, were proven EXPTIME-complete by Kasai, Adachi, and Iwata [KAI79]. Variants of pebble games were proven EXPTIME-complete by Kolaitis & Panttaja [KP03].

2. Shogi, also known as Japanese Chess, is a 2-player strategy game. It does share some properties of chess, though the games are not that much alike. It was proven EXPTIME-complete by Adachi et al. [AKI87]. The complexity of variants of Shogi was studied by Yato et al. [YSI05].

3. Quixo is a complicated variant of tic-tac-toe. It was shown EXPTIME-complete by Mishiba and Takenaga [MT20].

4. Cops and Robbers is a game played on a graph where a robber is trying to escape a group of cops trying to encircle them. It was shown EXPTIME-complete by Kinnersley [Kin15].

5. The Custodian Capture game is a game where pieces move like rooks and capture by being on either side of a piece. It was shown EXPTIME-complete by Ito et al. [INKT21].

6. Reachability-Time Games on Timed Automata were showed EXPTIME-complete by Jurdzínski and Trivedi [JT09].

7. Different versions of Angry Birds were shown NP-hard or PSPACE-hard or EXPTIME-hard by Stephenson et al. [SRG20].

8. A Graph Request-Response games is a 2-player graph game where the objectives are ANDs of conditions like "if a RED vertex is visited, then later on a BLUE vertex must be visited". Such games where shown EXPTIME-complete by Chatterjee, et. al [CHH11]. A variant of these, called "Streett games", were shown EXPTIME complete by Fijalkow & Zimmermann [FZ14].

Chapter 15

Undecidable Problems

As we have seen throughout this book, many problems and are NP-hard, or PSPACE-hard, EXPTIME-hard, or EXPSPACE-hard. (Many of them are also complete in those classes.)

1. Are there any problems that are undecidable? Yes. The Halting problem (defined below) is undecidable. Rice [Ric53] showed that any non trivial problem about Turing machines is undecidable. For example

$$\{M \mid \text{Turing Machine } M \text{ halts on all the primes}\}.$$

2. Are there any problems that do not refer to Turing machines that are undecidable? In this section we will give some examples of such.

15.1 Basic Undecidable Problems

To show that a set A is undecidable we need to already have some basic undecidable problem X and then show $A \le X$ (the reduction need not be polynomial time, just computable). We present some of these basic undecidable problems.

Halt
Instance: A Turing Machine M.
Question: Does M halt on 0?
Note: There are many equivalent formulations of Halt that are all efficiently reducible to each other.
Note: This is the problem that one first proves is undecidable from first principles. Almost all proves that a problem is undecidable use either Halt or some other problem known to be undecidable. This is similar to how we view SAT except that we actually *know* that Halt is undecidable, whereas we need to assume SAT is not in P.

Post Correspondence Problem (PostCorrProb)
Instance: An finite alphabet Σ and two vectors of words over Σ, $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\beta = (\beta_1, \ldots, \beta_n)$.
Question: Does there exists $1 \le i_1, \ldots, i_k \le N$ such that $\alpha_{i_1} \cdots \alpha_{i_k} = \beta_{i_1} \cdots \beta_P i_k$?
Note: k might be much larger than n so the naive algorithm of trying all possibilities to see if one works will go on forever if the answer is no.

A ***counter machine*** is another model of computation. We will not define if formally.

Counter Machines (CounterMach)
Instance: A Counter Machine M.
Question: Does M halt on 0?

Theorem 15.1.

1. *(Post [Pos46]) Halt $\le$ PostCorrProb with an efficient reduction.*

2. *If PostCorrProb $\le A$ with an efficient reduction then Halt $\le A$ with an efficient reduction. This will be important when we claim that a reduction of PostCorrProb to a game problem will produce reasonably small, playable, initial settings for the game.*

3. *(Minsky [Min67], but probably also folklore)) Halt $\le$ CounterMach; however the reduction takes a Turing machine of length n and returns a Counter machine of length $2^{O(n)}$. This will be important when we claim that a reduction of CounterMach to a game problem do not produce reasonably small, playable, initial settings for the game.*

15.2 Conway's Game of Life

In Conway's game of life, we have a grid of cells that are either living or dead. A cell's neighbors are defined to be the eight cells that are horizontally, vertically, or diagonally adjacent to it. Each iteration of the simulation, the cells update as follows:

1. A living cell stays alive if it has 2 or 3 living neighbors. Otherwise it dies.

2. A dead cell becomes living if it has exactly 3 living neighbors.

Questions about Life
Instance: An initial configuration for the game of Life. This is a grid where some cells are blank and some have a living cell.
Question: What will happen when the game is run. This is an informal question; however, we will state rigorous theorems.

With these rules, we can end up with periodic patterns, e.g. pulsars, or static patterns, known as still life. Some of these can be seen in Figure 15.1. For more examples of how various patterns behave, there are many interactive simulations online.

There are a couple different decision problems we can ask. The first, whether it is an example of still life is easy; we just compute the next step. We can also ask whether the configuration

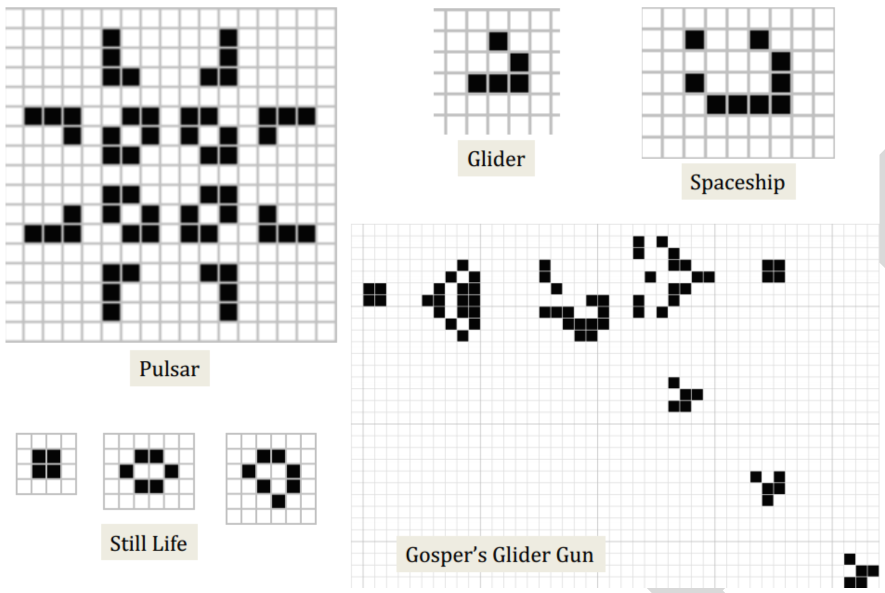


Figure 15.1: A couple of common patterns in Life. Black squares are living, and white squares are dead

is periodic and whether all the cells will die out, which are a bit harder. These are unbounded simulations, and with polynomial space, they are PSPACE-complete, whereas with infinite space but a finite starting area for live starting area, they are undecidable.

1. Given a configuration, is it a still life? This is trivial: just do one step of the game and see if anything changed.

2. Given a configuration and a box that it is inside of, so the game does not affect anything outside of that box, will it become periodic? Will all of the cells die? Randell [Ren00a, Ren00b] showed that both of these are PSPACE-complete.

3. Given a configuration will it become periodic? Will all of the cells die? Randell [Ren00a, Ren00b] showed that both of these are undecidable. The statement "The Game of Life is Undecidable" refers to this statement.

These results are generally proven by showing that Conway's Game of Life can simulate Turing machines. This works for both bounded games (e.g., inside a box) for PSPACE-hardness, and unbounded games for undecidability.

The Game of life is undecidable. Perhaps a lesson for us all.

15.3 The Video Game Recursed

Recursed is a 2D puzzle platform game involving chests, pink flames, green glows, crystals, ledges, jars, rings, and other objects. You start in a room. If you open a chest in that room you do not get an any object or treasure. Instead you are *in another room!* Jars also lead to rooms, but in a different way. Suffice to say, the game is complicated. We hasten to point out that *this is a fun game that people actually play.*

Recursed is a 1-player game where the goal is for the player to get the crystal. RECURSED is the decision problem where you are given an initial set up for the game Recursed and need to determine whether the player can win.

Demaine et al. [DKL20] showed the following.

Theorem 15.2. *RECURSED is undecidable.*

Some notes about the result and the proof:

1. Theorem 15.2 was proven by showing POSTCORRPROB $\leq$ RECURSED and using Theorem 15.1. Since the basic problem used is POSTCORRPROB, and the reduction is efficient, the instances of RECURSED that are produced are fairly small.

2. Most complexity-of-games results such as those about CHESS and GO rely on (1) the *board* getting bigger and bigger, and (2) such large constants that, even for short inputs, the resulting games are not playable. The result for recurse is novel in that (1) the board stays the same size at 15×20, and (2) the games for short inputs are playable. A caveat: the number of recursions from the chests gets bigger and bigger.

15.4 Other Undecidable Games

1. The video game Braid is playable and fun. We refer the reader to Wikipedia entry on the game, for details of the rules. We denote the problem of determining whether the player can win Braid by BRAID. Hamilton [Ham14] showed that BRAID is undecidable. Hamilton's proof uses COUNTERMACH. As a result, the instance of Braid that are produced are rather large. BRAID and RECURSED are the only two video game problems that we know of that are undecidable.

2. The game Magic is well known. We refer the reader to Wikipedia entry on the game, for details of the rules (or ask someone at random since many people play it). Note that Magic is 2-player, as opposed to Recursed of Braid which are 1-player. We denote the problem of determining whether the player can win Magic by MAGIC. Churchill et al. [Chu12] showed that MAGIC is undecidable by using counter-machines. As a result, the instance of MAGIC that are produced are rather large. Later Churchill et al. [CBH21] presented a reduction using Turing machines that was efficient; however, the instances of Magic had every players moves forced, so it was not a natural instance of the game. Biderman [Bid20] showed that the problem of mate-in-n for Magic is not arithmetic, which means its beyond the arithmetic hierarchy. The reduction produces natural game positions. MAGIC is the only 2-player game that we know of that is undecidable.

15.5 Diophantine Equations: HTEN

In 1900 David Hilbert proposed 23 problems for mathematicians to work on. We state Hilbert's tenth problem in todays terminology.

Is the following problem decidable?

HTEN
Instance: A polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$.
Question: Does there exist $a_1, \ldots, a_n \in \mathbb{Z}$ such that $p(a_1, \ldots, a_n) = 0$?
Note: We denote the problem where the degree $\le d$ and the number of variables is $\le n$ by HILBERT10(d, n).
Note: The question of decidability is equivalent to the case where $a_1, \ldots, a_n \in \mathbb{N}$. If you compare the results we state to those in the literature they might not be the same since the literature often states results for the $\mathbb{N}$ version.

Hilbert had hoped this problem would lead to number theory of interest. It did lead to some, but the combined efforts of Davis-Putnam-Robinson [DPR61] and Matijasevic [Mat70] (see also a survey article by Davis [Dav73] and a book by Matijasevic[Mat93]) showed that the problem was undecidable.

Gasarch [Gas21] has a survey about what happens for particular degrees d and number of variables n. Chow (as quoted in [Gas21] speculates that looking at degree and number of variables may be the wrong question:

> One reason there isn't already a website of the type you envision [one that has a grid of what happens for degree d, number-of-vars n] is that from a number-theoretic (or decidability) point of view, parameterization by degree and number of variables is not as natural as it might seem at first glance. The most fruitful lines of research have been geometric, and so geometric concepts such as smoothness, dimension, and genus are more natural than, say, degree. A nice survey by a number theorist is the book *Rational Points on Varieties* by Bjorn Poonen [Poo14]. Much of it is highly technical; however, reading the preface is very enlightening. Roughly speaking, the current state of the art is that there is really only one known way to prove that a system of Diophantine equations has no rational solution.

In the list below, d is the degree and n is the number of variables.

1. Grechuk [Gre21] stratifies diophantine equations and looks at for which levels we know they are solvable.

2. The status of the following problem with regard to decidability is not known: Given a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_n]$ does there exist $a_1, \ldots, a_n \in \mathbb{Q}$ such that $p(a_1, \ldots, a_n) = 0$ Matijasevic [Mat] gives reasons why this may be the question Hilbert meant to ask.

3. Here are the current undecidability results with the smallest d and the smallest n. (a) (Jones [Jon82]) HILBERT10(8,174) is undecidable, (b) (Sun [Sun20]) There is a d such that HILBERT10$(d, 11)$ is undecidable.

4. Here are the current decidability results with the largest d and n. (Siegel [Sie72], see also Grunewald & Segal [GS81]) For any n, HILBERT10$(2, n)$ is decidable.

5. The case of HILBERT10(3,2) is open; however, there are reasons to think it is decidable. See Section 4.3 of Gasarch [Gas21].

6. Matijasevic & Robinson [MR75] conjecture that there is a d such that HILBERT10$(d,3)$ is undecidable. Baker [Bak68] stated that the number theoretic tools available at the time could not do much with degree 3. Since his paper was in 1968, the undecidability result was not yet known; however, had it been known, he might have conjectured that there is an n such that HILBERT10$(3,n)$ is undecidable.

7. The status of the following problem with regard to decidability is not known: Given k, does $x^3 + y^3 + z^3 = k$ have a solution in $\mathbb{Z}$?

15.6 Mortal Matrices

We present another undecidable problem that does not refer to Turing machines.

MORTAL MATRICES
Instance: A set of m $n \times n$ matrices $M_1, \ldots, M_m$ over $\mathbb{Z}$,
Question: Does there exists $i_1, \ldots, i_N$ such that $M_{i_1} \times \cdots \times M_{i_N}$ is the all zero matrix.
Note that we are allowed to use M_i many times.

Theorem 15.3.

(Cassaigne et al. [CHHN14]) MORTAL MATRICES is undecidable for: (1) 6 3×3 matrices, (2) 4 5×5 matrices, (3) 3 9×9 matrices, (4) 2 15×15 matrices.

(Bournez & Branicky [BB02]) MORTAL MATRICES is decidable for 2 2×2 matrices.

(Bell et al. [BHP12]) MORTAL MATRICES for 2×2 matrices is NP-hard.

Note that the question of decidability for 2 3×3 matrices is open.

15.7 Further Reading

There are many (if you consider around 25 to be many) other undecidable problems that do not involve Turing machines. Two good sources are Poonen's paper [Poo14] and the Wikipedia page on undecidable problems:
https://en.wikipedia.org/wiki/Listofundecidableproblems
We list a few of those that do not involve too much background knowledge.

1. THE WORD PROBLEM FOR GROUPS Given a group by being given a finite set of generators and relations, and given two words in the group w_1, w_2, determine whether $w_1 = w_2$. Novikov [Nov55] and Boone [Boo58] independently showed that there is a group G so that the word problem for that group is undecidable.

2. THE GROUP ISOMORPHISM PROBLEM Given two groups by being given a finite set of generators and relations, determine whether they are isomorphic. Adian [Adi55] and Rabin [Rab58] independently showed that determining given one group, determining whether it is the trivial group, is undecidable.

3. Tiling the Plane Wang [Wan60] posed the following problem: given square 1×1 tiles where the edges are colored, is there a tiling of the plane such that whenever two squares touch, the edge they share has the same color. The tiles cannot be rotated. Berger [Ber66] showed that the problem was undecidable. After a sequence of papers simplifying the proof (see Poonen [Poo14] for the sequence) Culik [Cul96] showed that the problem is undecidable with a fixed set of 13 tiles to draw from.

4. Context-Free Grammar Equivalence Given two context free languages L_1 and L_2, do they generate the same language? This was shown undecidable by Greibach [Gre67]. See also Hopcroft [Hop69].

5. Skolem's Linear Recurrence Problem Given a linear recurrence of the form $u_n = a_{k-1}u_{n-1} + \cdots + a_0 u_{n-k}$ $(a_0, \ldots, a_{k-1} \in \mathbb{Z})$ and initial conditions, does there exist i such that $a_i = 0$? Skolem posted the question of if this is decidable in 1934. Halava et al. [HHHK05] showed that the problem was undecidable.

6. Airport Travel de Marcken [dM21] formulated a problem about planning a trip that he then proved is undecidable.

Chapter 16

Constraint Logic

16.1 Introduction

Constraint Logic is a technique that has been used to prove problems NP-hard, PSPACE-hard, EXPTIME-hard, and Undecidable. This chapter will give just a taste of this vast area. The interested reader should see the book by Hearn & Demaine [HD09] and/or the Ph.D. thesis of Hearn [Hea06].

In general, Constraint Logic aims to define one model of computation for which natural problems are complete for the natural complexity class in a variety of scenarios:

1. The *number of players* influencing the computation can vary, from ***0 players*** (a standard computation or a simulation like LIFE) to ***1 player*** (a nondeterministic computation or a puzzle with choices) to ***2 players*** (a perfect-information game where players alternate choices) to ***team games*** (with three or more players divided into two teams, and imperfect information between players).

2. The *length* of the computation can vary from ***bounded*** where the number of operations/moves is at most polynomial in the size of the computation, and ***unbounded*** where there is no a priori bound so the computation may go on for exponentially long.

Figure 14.1 summarizes the natural complexity classes for each combination of these parameters. Constraint Logic, interpreted appropriately for each setting (as described below), gives a complete problem for each.

16.2 Constraint Logic Graphs

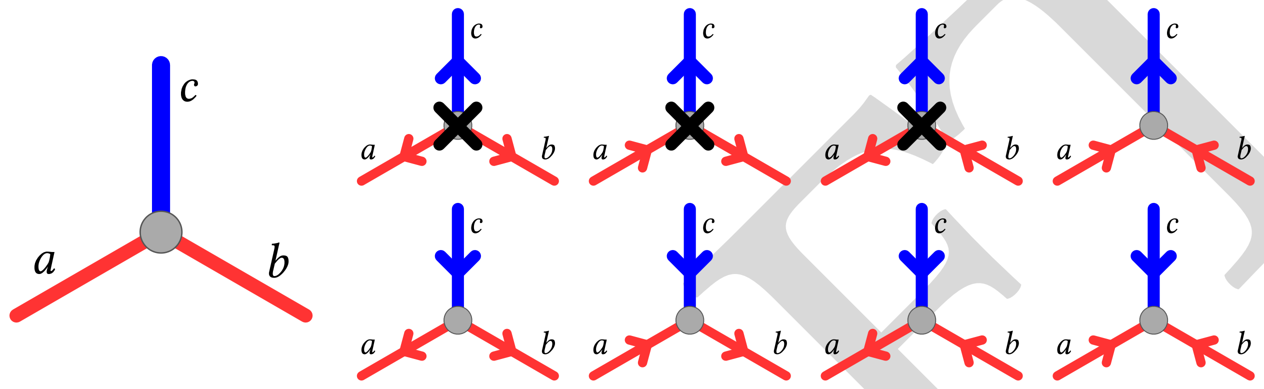
In each case, Constraint Logic defines computation in terms of a directed weighted graph representing the state of a machine:

Definition 16.1.

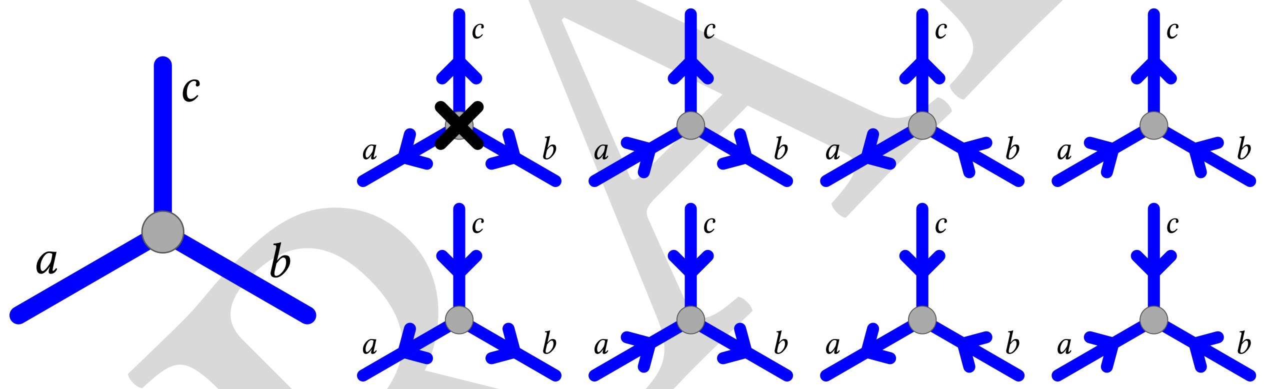
1. A ***constraint graph*** is an undirected weighted graph G where every weight is either 1 ("red", drawn thinner) or 2 ("blue", drawn thicker). (The constraint graph is the "machine".)

2. A ***configuration*** is a constraint graph together with an orientation (direction) for each edge. (A configuration represents a state of the machine.)

3. A ***valid configuration*** satisfies the ***constraints*** that every vertex has incoming weight at least 2.



(a) AND vertex: the top (blue) edge can be directed out if and only if the left *and* right (red) edges are directed in.



(b) OR vertex: the top (blue) edge can be directed out if and only if the left *or* right (blue) edge is directed in.

Figure 16.1: The two main types of Constraint Logic vertices, and all eight possible configurations, with illegal configurations labeled with an ×. (Based on [HD09, Figure 2.1].)

The key is the constraints that define "valid configuration". How do these constraints represent computation? Figure 16.1 gives two examples of vertices which represent (in a certain sense) AND and OR constraints. The top edge is constrained to direct out only when we have AND (both) or OR (at least one) of the bottom edges directed in. Intuitively, this makes it possible to express Boolean logic, and reduce from various versions of SAT.

16.3 Constraint Graph Satisfaction (CGS)

We start with the simplest form of Constraint Logic, corresponding to 1-player bounded computation. This problem is just a problem graph theory. It is "bounded" in the sense that each edge orientation can be chosen only once.

Constraint Graph Satisfaction (CGS)
Instance: A constraint graph G.
Question: Is there a valid configuration for G?
Note: CGS can be considered a 1-player game: the player is given the constraint graph and tries to find a way to orient the edges to obtain a valid configuration.

Hearn & Demaine [HD09] proved the following:

Theorem 16.2.

1. *3SAT $\le_p$ CGS, hence CGS is NP-complete.*
2. *CGS restricted to planar graphs is NP-complete.*
3. *CGS restricted to planar graphs where every vertex is an AND or an OR according to Figure 16.1 is NP-complete.*
4. *CGS restricted to grid graphs is NP-complete.*

CGS has not been applied to prove natural problems are NP-complete. Nonetheless, Theorem 16.2 is a good starting point for the study of Constraint Logic.

16.4 Nondeterministic Constraint Logic (NCL)

Next we consider the problem of, given two valid configurations, whether you can move from one to the other. This is essentially a 1-player unbounded game. It is "unbounded" because each edge can flip (exponentially) many times.

Nondeterministic Constraint Logic (NCL)
Instance: A constraint graph G and two valid configurations C_1 and C_2 of G.
Question: Is there a sequence of valid configurations that (1) begins with C_1, (2) ends with C_2, (3) every adjacent pair differ in that one edge's orientation was flipped?
Note: The term "Nondeterministic" is used because an NSPACE algorithm can solve the problem by guessing the next valid configuration. Because PSPACE = NSPACE, the problem is in PSPACE.

Hearn & Demaine [HD09] proved the following:

Theorem 16.3.

1. *NCL is PSPACE-complete.*
2. *NCL restricted to planar graphs is PSPACE-complete.*
3. *NCL restricted to planar graphs where every vertex is an AND or an OR according to Figure 16.1 is PSPACE-complete.*
4. *NCL restricted to grid graphs is NP-complete.*

The following two puzzles were proved hard by a reduction from NCL.[1]

SLIDING BLOCKS
The sliding block game is a puzzle game (both physical and digital) played on a square grid containing rectangular blocks. The player can slide each block horizontally and vertically so long as it does not collide with other blocks.
Instance: An initial position for the sliding-block game, and a special block and position for that block.
Question: Can the player win by moving the special block to the specified position?

RUSH HOUR
Rush Hour is a puzzle game (both physical and digital) played on a square grid containing horizontal $1 \times k$ and vertical $k \times 1$ cars (for varying k), and the player can slide each car along their long axis so long as it does not collide with other cars.
Instance: An initial position for the game Rush Hour, and a special car and position for that car.
Question: Can the player win by moving the special car to a specified position?

Theorem 16.4.

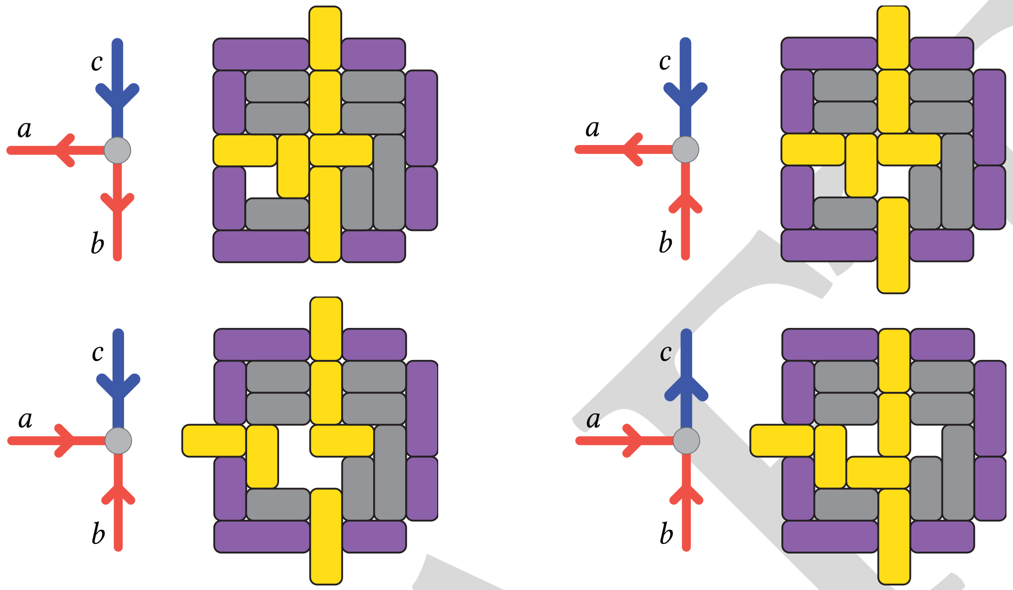
1. *(Flake & Baum [FB02], Hearn & Demaine [HD09]) RUSH HOUR is PSPACE-complete. The case where all cars are length* 2 *or* 3 *is still PSPACE-complete.*

2. *(Hearn & Demaine [HD09]) SLIDING BLOCKS is PSPACE-complete. The case where all of the rectangles are* 1×2 *is still PSPACE-complete.*

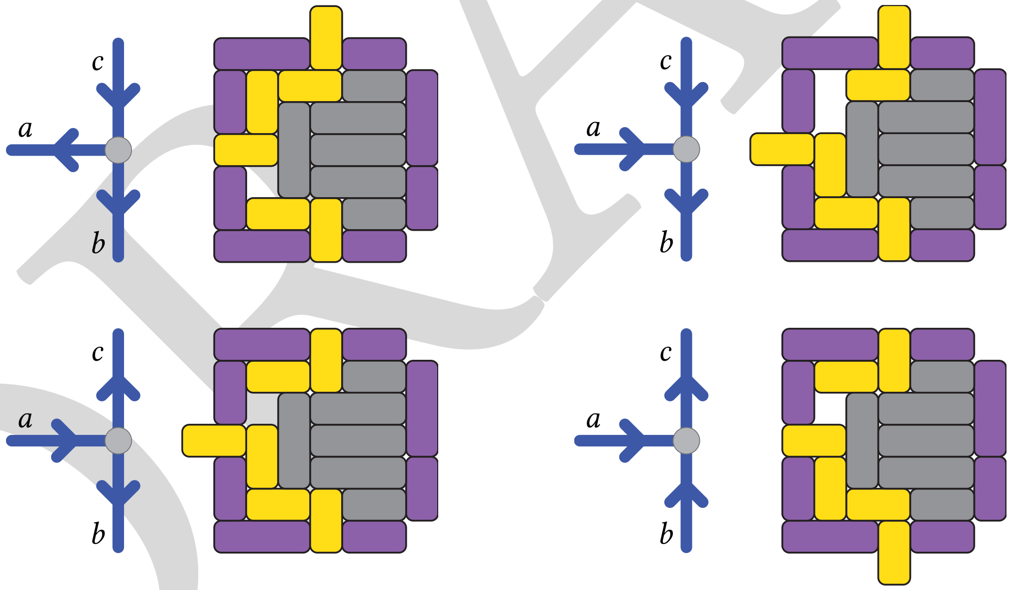
Figure 16.2 gives a sketch of the proof of Theorem 16.4.2 for 1×2 and 1×3 blocks. By Theorem 16.3.3, we just need to build an AND vertex and an OR vertex of a constraint graph, and show how to connect them into an arbitrary planar graph. Figure 16.2 gives two SLIDING BLOCKS gadgets that implement AND and OR vertices, and shows some sample move sequences for how to reverse edge directions by sliding blocks around. Here a block being retracted inside the gadget represents an NCL edge directed *away* from the vertex, and a block extending outside the gadget represents an NCL edge directed *into* the vertex (the opposite of what might seem natural). The naturally limited capacity of each gadget to store blocks ends up implementing the "incoming weight of at least 2" constraint. These gadgets can then be put together into a grid, following the grid graph of Theorem 16.3.3. For grid graphs we need a "straight" and "turn" gadget which simply propagate an edge in a desired direction; these gadgets can be constructed from an OR gadget by closing off one side. When the gadgets are against the outer boundary, we also need to extend the corner blocks to fill the empty corners in Figure 16.2; otherwise, they will be filled by the adjacent gadget. See [HD09] for details, and for the more complicated 1×2 construction.

Next we describe some "reconfiguration" problems that were proved hard by a reduction from NCL. In general, the ***reconfiguration*** version of an NP problem is, given two certificates, to determine whether you can change one into the other by a sequence of "local" moves. The natural notion of locality depends on the problem.

[1]Flake & Baum's RUSH HOUR hardness proof [FB02] predates and provided inspiration for NCL.



(a) AND gadget: The top middle block can be retracted into the gadget if and only if the left middle block *and* the bottom middle block can be extended out of the gadget (i.e., retracted into the neighboring gadget).



(b) OR gadget: The top middle block can be retracted into the gadget if and only if the left middle block *or* the bottom middle block can be extended out of the gadget (i.e., retracted into the neighboring gadget).

Figure 16.2: The reduction from NCL to Sliding Blocks consists of just two gadgets (plus an argument about how they fit together, omitted here). Dark blocks are never useful to slide. (Based on [HD09, Figure 1.2].)

RECONFIGURATION SAT
Instance: Boolean formula $\varphi(x_1, \ldots, x_n)$ and two satisfying assignments $\vec{x}, \vec{y}$ for φ.
Question: Is there a sequence of satisfying assignments for φ that (1) begins with $\vec{x}$, (2) ends with $\vec{y}$, and (3) every two consecutive satisfying assignments differ in only one variable?

RECONFIGURATION NAE-3SAT
Instance: NAE-3SAT formula $\varphi(x_1, \ldots, x_n)$ and two satisfying assignments $\vec{x}, \vec{y}$ for φ.
Question: Is there a sequence of satisfying assignments for φ that (1) begins with $\vec{x}$, (2) ends with $\vec{y}$, and (3) every two consecutive satisfying assignments differ in only one variable?

Theorem 16.5.

1. *(Gopalan et al. [GKMP09]) RECONFIGURATION SAT is PSPACE-complete.*
2. *(Gopalan et al. [GKMP09]) RECONFIGURATION NAE-3SAT is PSPACE-complete.*
3. *(Cardinal et al. [CDE+20]) RECONFIGURATION NAE-3SAT restricted to planar formulas is PSPACE-complete.*

Note that PLANAR NAE-3SAT is in P, so it is more surprising that RECONFIGURATION PLANAR NAE-3SAT is PSPACE-complete.

Holzer & Jakobi [HJ12] show the following maze puzzles hard using a reduction from NCL. (The reader needs to look at their paper for formal definitions of the mazes.)

Theorem 16.6.

1. *Determining whether a a Rolling Block Maze can be solved is PSPACE-complete.*
2. *Determining whether an Alice Maze can be solved is PSPACE-complete.*

For more games and puzzles proved PSPACE-complete using NCL, see Hearn [Hea06], Hearn & Demaine [HD09], Hearn [Hea09], and Holzer & Jakobi [HJ12].

16.5 DETERMINISTIC CONSTRAINT LOGIC (DCL)

Next we look at a 0-player version of Constraint Logic, which corresponds to a more typical computation. To make it zero player, we add rules to Constraint Logic to ensure that all moves are deterministic, rather than chosen by a player. Aside from orientation of edges, we also track whether an edge is "active" or "inactive". An edge is ***active*** if it was just flipped in the last round. Otherwise, it is ***inactive***. We call a *vertex* ***active*** if its active incoming edges have a total weight of at least 2.

DETERMINISTIC CONSTRAINT LOGIC (DCL)
Instance: A constraint graph G and an edge e.
Question: Consider a sequence of rounds, where in each round the following things happen:

- Inactive edges pointing to active vertices get reversed.
- Active edges pointing to inactive vertices get reversed.
- The edges that have been reversed are the new active edges.

Does edge e ever get reversed?

It is easy to see that DCL is in PSPACE. Demaine et al. [DHHL22] proved[2]

Theorem 16.7.

1. *DCL is PSPACE-complete.*
2. *DCL restricted to planar graphs is PSPACE-complete.*

Demaine et al. [DHHL22] use Theorem 16.7 (or rather, a framework used to prove Theorem 16.7) to prove PSPACE-completeness for predicting the behavior of several reversible deterministic systems. We give one example.

BILLIARDS
Instance: A set of billiard balls on a table. The table also has fixed blocks which, when a ball hits it, reflect off at the same angle. We are also given, for each ball, the direction the ball will initially roll and the ball's initial velocity. Finally, we are given a ball and a place on the table.
Question: Will the specified ball ever get to the specified place?

Theorem 16.8.

1. *(Fredkin & Toffoli [FT82]) BILLIARDS is PSPACE-complete.*
2. *(Demaine et al. [DHHL22]) DCL $\le$ BILLIARDS, hence BILLIARDS is PSPACE-complete. This proof (1) is simpler than that of Fredkin & Toffoli, and (2) has only two balls moving at any one time and they are close together.*

16.6 2-PLAYER CONSTRAINT LOGIC (2CL)

We have looked at 0-player and 1-player Constraint Logic. Next we turn to 2-player Constraint Logic. We define two games corresponding to bounded and unbounded length:

Definition 16.9.

[2]This result was claimed in Hearn & Demaine [HD09]; however, the proof had a very subtle flaw. The flaw is discussed and a correct proof is given by Demaine et al. [DHHL22].

1. A ***2-player constraint graph*** is a constraint graph where each edge is labeled White or Black, in addition to being labeled red (weight 1) or blue (weight 2). The two colorings are independent of each other.

2. The ***Bounded/Unbounded 2-Player Constraint Logic Game*** is as follows:

   (a) Initially the two players, called "white" and "black", are given a 2-player constraint graph and a valid configuration. Each player is also given a target edge.

   (b) The players alternate making moves, with white going first. A move consists of flipping the orientation of an edge of the same color as the player, such that the resulting configuration is still valid.

   (c) In the *bounded* game, each edge can be reversed at most once.

   (d) The first player to reverse their target edge wins.

BOUNDED/UNBOUNDED 2-PLAYER CONSTRAINT LOGIC (BOUNDED/UNBOUNDED 2CL
Instance: 2-player constraint graph, valid orientation, and two target edges (so a starting board for the game).
Question: Determine who wins the Bounded/Unbounded 2-Player Constraint Logic Game.

Hearn & Demaine [HD09] proved the following:

Theorem 16.10.

1. *BOUNDED 2CL is PSPACE-complete.*

2. *BOUNDED 2CL restricted to planar graphs is PSPACE-complete.*

3. *UNBOUNDED 2CL is EXPTIME-complete.*

4. *UNBOUNDED 2CL restricted to planar graphs is EXPTIME-complete.*

We describe three games that have reductions from BOUNDED 2CL:

Definition 16.11.

1. ***Amazons*** is played by two players—black and white—with an equal number of black and white queens on a chess board. Each turn, a player must make a "queen move" with one of his queens, and then shoots an arrow onto any square reachable by a "queen move" from the new position of the queen. Here, a queen move is any non-zero move in a straight line diagonally, horizontally, or vertically. Queens may not move through or shoot through squares occupied by arrows or other queens. The player who is able to move last wins. We denote the problem of determining who wins by AMAZONS.

2. ***Konane*** is played with 2 colors of stones—black and white—on a board. Each turn, a player can perform with a single piece, 1 or more jumps over opponent pieces, as long as they all lie in a straight line. The jumped pieces are removed. The last player to move wins. We denote the problem of determining who wins by KONANE.

3. ***Cross Purposes*** is played with black and white stones on the intersection of a Go board. A black stone represents towers of two blocks, and a move consists of "pushing" a black stone over, resulting in two white stones either above, below, left, or right of the stone. The two players are named Vertical and Horizontal, with Vertical moving first. Vertical may only tip over a stone up or down, and Horizontal may only tip over the stone left or right, so long as the two spaces are unoccupied before hand. The last player to be able to move wins. We denote the problem of determining who wins by Cross Purposes.

It is easy to see that Amazons, Konane, and Cross Purposes are in PSPACE. The following theorem is proved by a reduction from 2CL.

Theorem 16.12. *Amazons, Konane, and Cross Purposes are PSPACE-complete.*

Bilò et al. [BGL+18] show the following using a reduction from 2CL. (the reader needs to look at their paper for formal definitions of the game).

Theorem 16.13. *Peg Duotaire (a 2-player peg jumping game) is PSPACE-complete.*

Unbounded 2CL has not yet been used to show natural problems EXPTIME-complete; however, it is part of a framework for lower bounds developed by Demaine et al. [DHL20] and Ani et al. [ADHL22].

Exercise 16.14. (This is a large project.) Prove Go, Chess, or Checkers EXPTIME-complete using Unbounded 2CL.

16.7 Team Private Constraint Logic (TPCL)

Definition 16.15. The ***Bounded/Unbounded Team Private Constraint Logic Game*** is a version of the Constraint Logic game played with three players—one black player and two white players—on a ***team private constraint graph***. Like a 2-player constraint graph, every edge is marked as flippable by only by the white team or only by the black team. A new feature is that the state of the edges in the game is semi-private information: each edge is marked as visible to (only) the white players, one of the white players, or the black player.

The same moves allowed in 2CL games apply, including in the bounded case a limit of one reversal per edge. Players face the additional restriction that all moves that they make must be legal given visible information. Passing is also allowed, which makes it difficult to gain any information about changes to the state of the board based on the possible moves of other players. This is a necessary condition, because a black player may move an invisible edge, which may not change the available moves for a white player, thus giving them some knowledge about the black player's moves. By allowing passing, the white player does not know whether the state of an invisible edge was changed, or if the black player decided to pass on a turn.

Bounded/Unbounded Team Private Constraint Logic (BTPCL)
Instance: A team private constraint graph.
Question: Determine who wins the Bounded/Unbounded Team Private Constraint Logic Game.

Hearn & Demaine [HD09] proved the following:

Theorem 16.16. *BTPCL ∈ NEXPTIME.*

Proof. If there exists a winning strategy for the white player, the strategy is deterministic and is a function of the visible state. There are at most *n* different visible edges, so the number of possible states is exponential in the number of visible edges. Thus the white players may guess a strategy at the beginning of the game, which takes exponentially many bits, and the white players play deterministically with that strategy to verify that it is a solution. The game may run for many rounds, but the information used to describe the moves is exponential, hence determining who wins is in NEXPTIME. □

Theorem 16.17. *BTPCL is NEXPTIME-complete.*

Theorem 16.18. *BTPCL is CE-complete and in particular Undecidable.*

The last result provides a game with a finite board (the constraint graph) where the question of who wins is Undecidable. This result is surprising because Turing machines or word-RAM algorithms can run for an arbitrary amount of time and have an arbitrary amount of space to store information and their state, whereas the game has finite board space and cannot represent the state of an arbitrarily large computation. Effectively, the state of the simulated machine is in the heads of the players as they consider what move would be optimal.

Part III

Below NP

Chapter 17

The 3SUM-Conjecture: A Method for Obtaining Quadratic Lower Bounds

17.1 Introduction

A large part of this book is all about showing that some problems are probably not solvable in polynomial time. But even within polynomial time there are distinctions.. We will show certain problems are probably not in subquadratic time.

17.2 The 3SUM Problem

Gajentaan & Overmars [GO12] used the following problem to show that other problems are probably not in subquadratic time.

3SUM
Instance: n integers.
Question: Do three of the integers sum to 0?
Note: We consider any arithmetic operation to be unit cost.

The following theorem gives better and better algorithms for 3SUM.

Theorem 17.1.

1. *3SUM can be solved in $O(n^3)$ time.*
2. *3SUM can be solved in $O(n^2 \log n)$ time.*
3. *3SUM can be solved in randomized $O(n^2)$ time.*
4. *3SUM can be solved in deterministic $O(n^2)$ time.*

Proof. Let A be the original input of n integers.
1) The trivial algorithms suffice to get $O(n^3)$ time: check all $O(n^3)$ 3-sets of A to see if any of them sum to 0.

2) First compute all the pairwise sums of A and sort them into an array B. This takes $O(n^2 \log n)$ steps. Then, for each element of A, use binary search to see if its negation is in B. This takes $O(n \log n)$ steps. If you find a negation in B then the answer is YES, otherwise NO.

3) First compute all the pairwise sums of A and put their negations into a hash table B together with the pair (for example, a number x to $x \pmod p$ where p is a prime close to n^2). This takes $O(n^2)$ steps. Then, for each element of $x \in A$ hash it into the table. See if there is a pair already there, and if there is see if the sum is 0. With very high probability there will only be $O(1)$ pairs to check.

4) First sort A. This takes $O(n \log n)$ steps. Place a pointer at both the front and the end of A. Then, for each $x \in A$, do the following: If the sum of the integers at the two pointers and x is smaller than 0, we move the first array's pointer forward; if the sum is larger than 0, we move the second array's point backwards; otherwise, we have the three integers sum to 0, and we are done. If the two pointers crossover, we move onto the next integer in A. This algorithm clearly takes $O(n^2)$ time. □

Exercise 17.2. Code up all four algorithms in Theorem 17.1. Run them on data and see which ones do well when.

Is there an algorithm for 3SUM that runs in time better than $O(n^2)$? This depends on your definition of "better". The following are known:

1. If the integers are in $[-u, u]$ then 3SUM can be solved in $O(n + u \log n)$ time. We leave this an an exercise.

2. Baran et al. [BDP08] showed the following: Assume the word-RAM model which can manipulate $\log n$-bit words in constant time. Then there is a randomized algorithm for 3SUM that takes time
$$O\left(\frac{(n \log\log n)^2}{(\log n)^2}\right).$$

3. Gronlund & Pettie [GP18] have shown that there is a randomized algorithm for 3SUM that takes time
$$O\left(\frac{n^2 \log\log n}{\log n}\right)$$
and a deterministic algorithm that takes time
$$O\left(\frac{n^2 (\log\log n)^{2/3}}{(\log n)^{2/3}}\right).$$

4. Chan [Cha20] has shown there is a deterministic algorithm for 3SUM that runs in time
$$O\left(\frac{n^2 (\log\log n)^{O(1)}}{\log^2(n)}\right).$$

5. Gronlund & Pettie [GP18] have also shown that *there exists* a decision tree algorithm that had depth (so time) $O(n^{1.5}\sqrt{\log n})$ for 3SUM. Their proof does not show how to actually construct the decision tree in subquadratic time.

Exercise 17.3. Show that if the integers are in $[-u, u]$ then 3SUM can be solved in $O(n + u \log n)$ time.

While the algorithms above are impressive and very clever none are that much better than $O(n^2)$. We need a terminology for that.

Definition 17.4. An algorithm is ***subquadratic*** if there exists $\varepsilon > 0$ such that it runs in time $O(n^{2-\varepsilon})$.

Despite enormous effort nobody has obtained a subquadratic algorithm for 3SUM. In the next section we make that a conjecture and, from the conjecture, obtain quadratic lower bounds on other problems.

Exercise 17.5. Let $a, b, c \in \mathbb{Z}$ such that they are not all equal. Show that (a, a^3), (b, b^3) and (c, c^3) are colinear if and only if $a + b + c = 0$.

17.3 The 3SUM-Conjecture

Gajentaan & Overmars [GO12] (essentially) defined 3SUM-hardness and showed many problems are 3SUM-hard. Many of the problems are in computational geometry.

Conjecture 17.6. *The* ***3SUM-Conjecture****: There is no subquadratic algorithm for 3SUM.*

We will later see ways in which this conjecture is similar to the conjecture that SAT is not in P and ways in which it is different.

We define a notion of reduction between problems.

Definition 17.7. Let A and B be sets or functions (they will almost always be sets).

1. $A \le_{sq} B$ means that if there is a subquadratic algorithm for B then it can be used to obtain a subquadratic algorithm for A. Usually this will be just like $\le_p$ where you solve an instance of A by quickly creating an instance of B. However, there are times when you need to make more queries to B. The *sq* stands for ***sub-quadratic***.

2. $A \equiv_{sq} B$ if $\mathcal{A} \le_{sq} B$ and $B \le_{sq} A$.

Definition 17.8. A problem A is 3SUM-hard if 3SUM $\le_{sq} A$.

By the 3SUM-Conjecture we think that, if A is 3SUM-hard, then there is no subquadratic algorithm for A.

The definition of NP-hard is used to show that problems are not in P, contingent on the conjecture that P $\ne$ NP. The definition of 3SUM-hard is used to show that problems do not have subquadratic algorithms, contingent on the conjecture that 3SUM does not. We list out similarities and differences between the two theories.

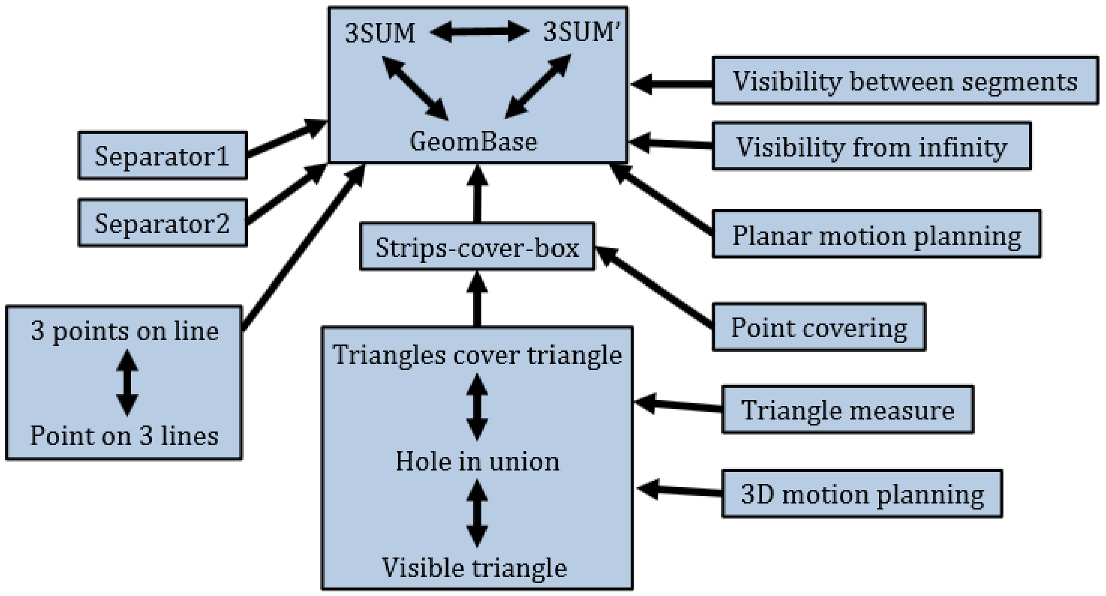


Figure 17.1: The Relationships of Problems to 3SUM

1. Both use reductions and build up a large set of problems that are thought to be hard. The number of NP-hard problems is far larger than the number of 3SUM-hard problems.

2. Contrast the following:

   (a) A problem B is NP-hard if for all $A \in \mathrm{NP}$, $A \le_p B$. SAT is a natural NP-hard problem. As a consequence, one can show C is NP-hard by showing SAT $\le_p C$.

   (b) A problem B is 3SUM-hard if 3SUM $\le_{sq} B$. Note that we *do not* have a result for 3SUM that is analogous to the Cook-Levin Theorem. We suspect 3SUM requires quadratic time and use it as such.

3. There is no notion of 3SUM-complete since there is no natural class like NP or PSPACE that would make sense.

If we did not know the Cook-Levin theorem, but really thought SAT was hard, we could still have a large set of problems that are NP-complete and think they were hard. That is the position we are in with 3SUM-hard.

Exercise 17.9. Let 2SUM be the problem of, given an array A of integers, does there exists $x, y \in A$ such that $x + y = 0$.

1. Show that 2SUM is in $O(n \log n)$ time.

2. Show that 2SUM is in $O(n)$ time.

3. F3SUM is the following problem: Given a set of integers A, determine whether there exists $x, y, z \in A$ such that $x + y + z = 0$ AND if there is then output an (x, y, z) that works. Show that F3SUM $\le_{sq}$ 3SUM.

17.4 Many 3SUM-hard Problems

17.4.1 Three Variants of 3SUM

Recall that if the integers are in $[-u, u]$ then 3SUM can be solved in time $O(n + u \log n)$. For $u = O(n^{2-\epsilon})$ this yields a subquadratic algorithm for 3SUM. What if u is bigger? The hashing technique that Baran et al. [BDP08] used to get a slightly better than $O(n^2)$ algorithm for 3SUM (we mentioned it above) can be used to prove the following.

Theorem 17.10. *The 3SUM problem restricted to $[-n^3, n^3]$ is 3SUM-hard.*

Proof sketch: The proof hashes the set of n elements to $[-n^3, n^3]$. ∎

Theorem 17.10 is interesting since it rules out the possibility that 3SUM is only hard when it involves large numbers.

Pătraşcu [Păt10] considered the following problem and proved it was 3SUM-hard.

CONVOLUTION 3SUM
Instance: A set of n integers $a_1, \ldots, a_n$.
Question: Is there $i \neq j$ such that $a_{i+j} = a_i + a_j$. Note that proving CONVOLUTION 3SUM is in $O(n^2)$ is much easier than showing 3SUM is in $O(n^2)$.

Theorem 17.11. *CONVOLUTION 3SUM is 3SUM-hard.*

Proof sketch: This proof uses a family of hash functions. ∎

Gajentaan & Overmars [GO12] defined the following problem and showed it was 3SUM-hard. It is used in many proofs that problems are 3SUM-hard.

3SUM′
Instance: Three sets A, B, C of n integers.
Question: Is there $a \in A$, $b \in B$, and $c \in C$ such that $a + b = c$.

Theorem 17.12. *3SUM $\equiv_{sq}$ 3SUM′.*

Proof. 3SUM $\leq_{sq}$ 3SUM′.

This is easy to see by a reduction from 3SUM with the original instance being S; just put $A = S$, $B = S$, $C = -S$.

3SUM′ $\leq_{sq}$ 3SUM.

Given A, B, C we can make S to be all the elements of $A + large$, $B + 2\,large$, and $C - 3\,large$, where *large* just means a large number that is added to each element of the corresponding sets. □

Note that the reduction 3SUM′ $\leq_{sq}$ 3SUM used large numbers.

17.4.2 3SUM-hard Problems in Computational Geometry

All of the results in this section are by Gajentaan and Overmars [GO12].

GeomBase and GeomBase′

GeomBase
Instance: n points in $\mathbb{Z}^2$ with y-coordinate in $\{0, 1, 2\}$.
Question: Does there exist a non-horizontal line hitting 3 points of this set. This is displayed in Figure 17.2

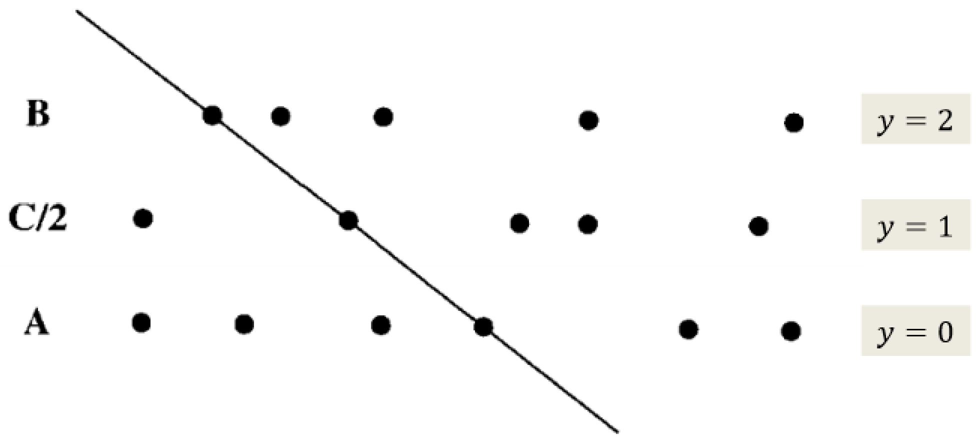


Figure 17.2: The Problem of GeomBase

Theorem 17.13. *3SUM′ $\equiv_{sq}$ GeomBase. Hence GeomBase is 3SUM-hard.*

Proof. 3SUM′ $\leq_{sq}$ GeomBase:

Given A, B, C, an input to 3SUM′, we consider the following input to GeomBase

$$\{(a, 0) \mid a \in A\} \cup \{(b, 2) \mid b \in B\} \cup \{(\frac{c}{2}, 1) \mid c \in C\}$$

It is easy to show that three points lie on a non-horizontal line if and only if there exists $a \in A$, $b \in B$, and $c \in C$, such that $a + b = c$.

GeomBase $\leq_{sq}$ 3SUM′:

Reverse the above reduction. We may have that the points may not be lattice points, but we can always scale the x-coordinates to become integers, and then we can reduce to 3SUM′.

□

We will sometimes need the following variant of GeomBase.

GeomBase′
Instance: n points in $\mathbb{Z}^2$ with y-coordinate in $\{0, 1, 2\}$, and ε. View the points as holes in the $y = 0, 1, 2$ lines and enlarge them to be ε-long. I addition we view the $y = 0, 1, 2$ lines as finite segments.
Question: Does there exist a non-horizontal line going through 3 of the holes. hitting 3 points of this set.

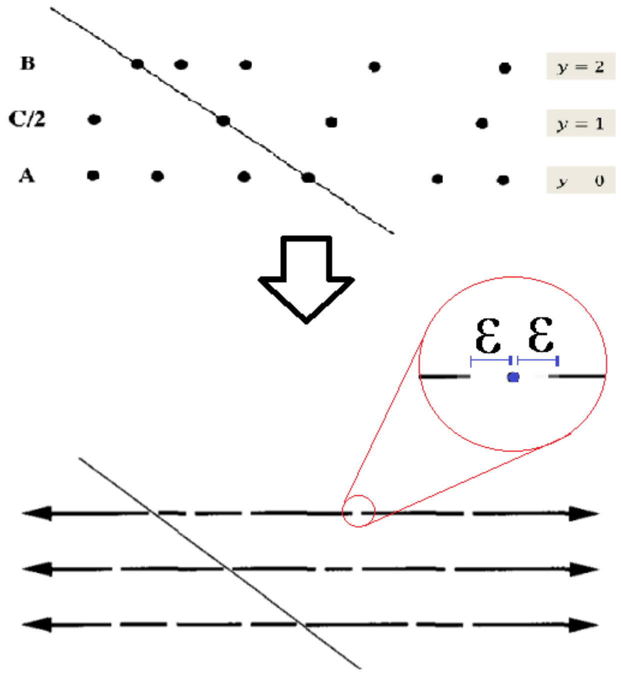


Figure 17.3: The Transformation to GeomBase′

Theorem 17.14. *GeomBase $\leq_{sq}$ GeomBase′*

Proof sketch: This proof is displayed in Figure 17.2. ▮

Collinearity

We now begin a shift towards Incidence Problems.

Collinear
Instance: n points in $\mathbb{Z} \times \mathbb{Z}$,
Question: Are three of the points collinear?

Theorem 17.15. *3SUM $\leq_{sq}$ Collinear, hence Collinear is 3SUM-hard.*

Proof. Given an instance of 3SUM A, we map it to the instance of Collinear $\{(x, x^3) \mid x \in A\}$, as shown in Figure 17.4.

We show that three points on the curve will lie on a line if and only if there are three integers summing to 0 in the original set.

Indeed, notice that

$$\frac{b^3 - a^3}{b - a} = \frac{c^3 - a^3}{c - a} \iff b^2 + ba + a^2 = c^2 + ca + a^2 \iff (b - c)(b + c + a) = 0 \iff b + c + a = 0,$$

where in the last equality we assume that the three numbers are distinct. □

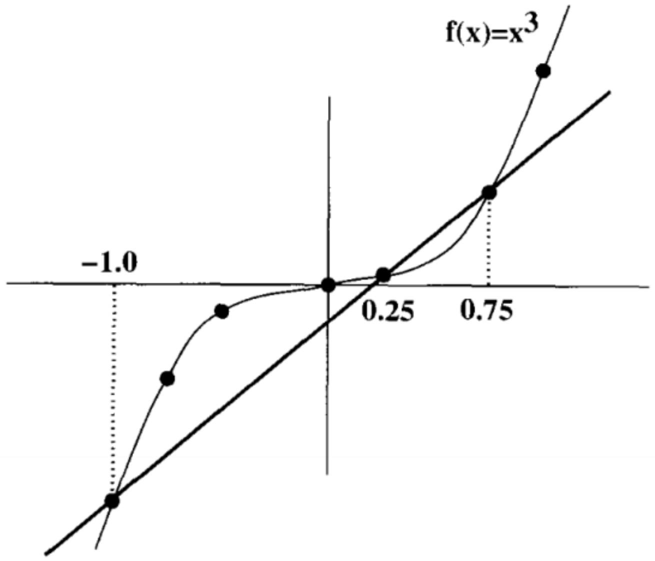


Figure 17.4: The Reduction from 3SAT to Collinearity

Concurrency

Concurrency is the geometric dual[1] of collinearity.

CONCURRENT
Instance: n lines.
Question: Is there a point on three of the lines (such lines are called ***concurrent***).

Theorem 17.16. *COLLINEAR $\equiv_{sq}$ CONCURRENT and hence CONCURRENT is 3SUM-hard.*

Proof. COLLINEAR $\leq_{sq}$ CONCURRENT

Given a set of points X we map it to the set of lines $\{ax+by+1 = 0 \mid (a,b) \in X\}$. (This is called projective plane duality.) Then, this preserves point/line incidence; if three points were collinear, the three corresponding lines are incident, and vice versa. Therefore, we have a subquadratic reduction.

CONCURRENT $\leq_{sq}$ COLLINEAR

Use the reverse of the COLLINEAR $\leq_{sq}$ CONCURRENT reduction. □

As a side note, all the d dimensional versions of the problems mentioned so far, are $d+1$-sum hard.

[1]For more details about Geometric Duality, please refer to the lecture notes here: https://www.cs.duke.edu/harish/papers/geoduality.pdf

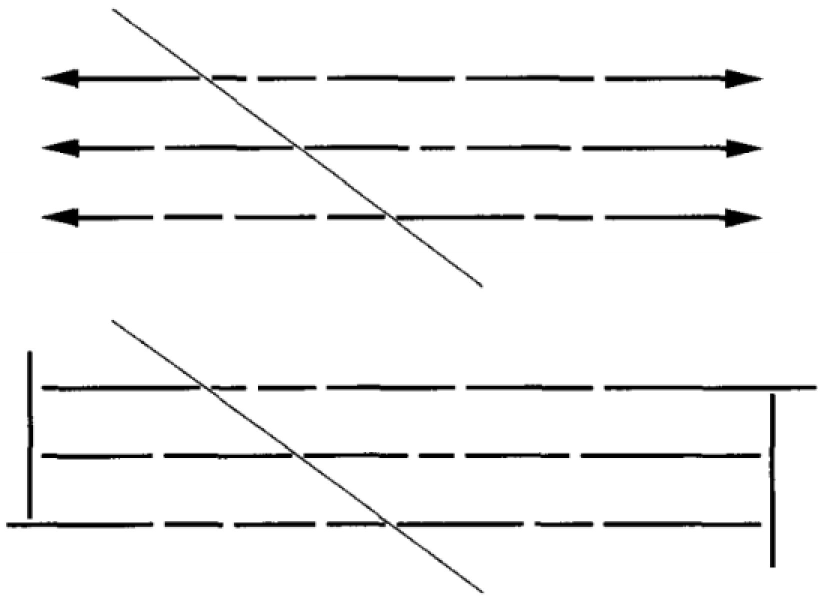


Figure 17.5: The Reduction from GeomBase′ to Separator

Separator

> Separator
> *Instance:* n line segments in the plane.
> *Question:* Is there a line that separates the n line segments into two nonempty groups. This line is not allowed to intersect any of the segments. This is often called the ***Separator Problem***.

Theorem 17.17. *GeomBase′ $\leq_{sq}$ Separator, hence Separator is 3SUM-hard.*

Proof. Let A be an instance of GeomBase. Take the complement of A (relative to the $y = 1, 2, 3$ lines) to get an instance B of Separator. It is easy to see that $A \in$ GeomBase if and only if $B \in$ Separator. We can reduce from GeomBase to Separator. See Figure 17.5 for clarity. Consider the corresponding GeomBase' Instance. □

Strips Cover Box

We are now on course with Covering Problems.

Definition 17.18. A ***strip*** is just a fixed region in between two parallel lines.

> Strips Cover Box Problem(Strips)
> *Instance:* A set of n strips and an axis-aligned rectangle.
> *Question:* Can a union of the strips cover the rectangle?

Theorem 17.19. *GeomBase $\leq_{sq}$ Strips, hence Strips is 3SUM-hard.*

Proof. Consider the corresponding instance of GeomBase', and rotate it 90 degrees. Then, consider the following duality: take the point (m, b) and map it to the line $y = mx + b$. The reverse construction is possible for all non-vertical lines. This is described in Figure 17.6.

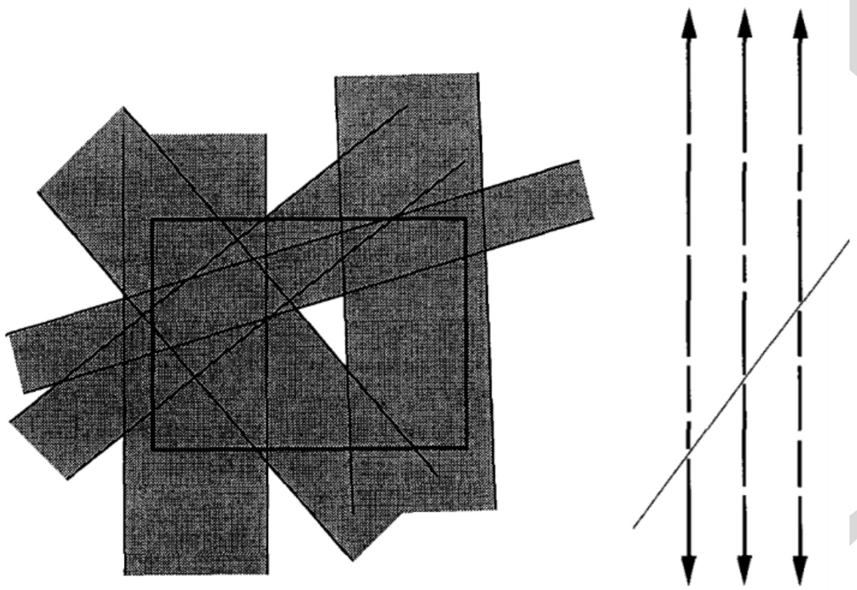


Figure 17.6: The Reduction from GeomBase to Strips Using GeomBase′

Then, observe that points on a a single vertical segment, will get mapped into a strip of lines. Further, we can get the box by considering the 6 half-planes that occur because of the rays, and then consider the bounding box of the hole formed by the union of these 6 regions. Then, we restrict half planes to this rectangle by adding 6 more strips. We then have that if there is a point left behind by these strips, then GeomBase was in the negative. □

Triangles Cover Triangle

We can also do the same type of question with Triangles.

Triangle Cover Triangle Problem (TriCovTri)
Instance: A set of n triangles and a target triangle.
Question: Can the set of triangles cover the target one?

Theorem 17.20. *Strips $\leq_{sq}$ TriCovTri, hence TriCovTri is 3SUM-hard.*

Proof. We start with a box. Make a triangle that covers the box, and then triangulate the exterior of the rectangle, but interior of triangle. Then, for each strip, triangulate each intersection. Then, we have the question do a bunch of triangles cover a triangle. We need to make sure we don't blow up complexity when we triangulate regions of strips, but this is fine as triangulation results in $O(1)$-gons. □

Hole in Union

> HOLE IN UNION PROBLEM (HOLEINU)
> *Instance:* A set of n triangles place in the plane. They can overlap.
> *Question:* Does the set have a hole? That is, is there a closed region within the union that is not covered by any of the triangles?

Theorem 17.21. *TRICOVTRI $\leq_{sq}$ HOLEINU, hence HOLEINU is 3SUM-hard.*

Proof. Given an instance of TRICOVTRI form an instance of HOLEINU by taking very thin triangles that cover the edges of the original target triangle. We then have that there is a hole if and only if the triangles do not cover the target triangle. □

Triangle Measure

We have yet another question regarding triangles.

> TRIANGLE MEASURE PROBLEM (TRIMEAS)
> *Instance:* A set of n triangles in the plane.
> *Question:* Compute the measure of their union. Thats just the area.

Theorem 17.22. *TRICOVTRI $\leq_{sq}$ TRIMEAS, so TRIMEAS is 3SUM-hard.*

Proof. Given an instance of TRICOVTRI we do the following. First, we can add extra triangles to cover the edges of the original triangle instance. Then, we have that the area of the union of the triangles is equal to the area of the original triangle if and only if the triangles cover all the triangles. □

Point-Covering

> POINT-COVERING PROBLEM (POINTCOV)
> *Instance:* n half-planes and a number k.
> *Question:* Is there a k-way intersection, as in is there a point covered by at least k of the half-plane?

Observe if $k \leq \frac{n}{2}$, we have that the answer is going to always be yes.

But for larger values of k, this becomes harder.

Theorem 17.23. *STRIPS $\leq_{sq}$ POINTCOV, hence POINTCOV is 3SUM-hard.*

Proof. For each strip, take the 2 half-planes that do not cover the strip. Then, we have any point that does not lie in any of the strips lies in exactly n half-planes. Any other point will lie in fewer half-planes. Then, add 4 more half-planes that are directed inwards into the rectangle with each line overlapping with a rectangle edge. Then, if there is a point that is covered by exactly $n+4$ of the $2n+4$ half-planes, we know the Strips will not cover the Box, and thus we have a reduction. □

Visibility Between Segments

We now shift towards Visibility Problems. Our first problem deals with Visibility Between Segments.

> VISIBILITY BETWEEN SEGMENTS (VISBETSEG)
> *Instance:* A set of n horizontal line segments in the plane, and two particular segments s_1 and s_2.
> *Question:* Are there points on s_1 and s_2 that can see each other; in other words, is there a segment that, aside from its endpoints on s_1 and s_2 does not intersect any of the n horizontal line segments. This is displayed in Figure 17.7.

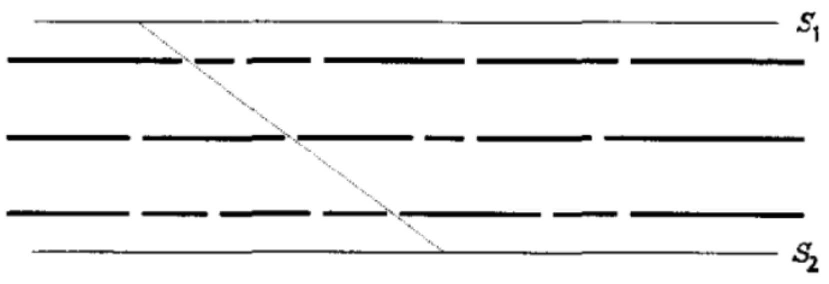


Figure 17.7: Segment Visibility Reduction Using GeomBase

Theorem 17.24. *GEOMBASE′ $\leq_{sq}$ VISBETSEG, hence VISBETSEG is 3SUM-hard.*

Proof. Take an instance of GEOMBASE′. Add two segments, one above the segments and the other below. These two segments can only see each other if and only if there are three collinear ε-neighborhoods corresponding to three collinear points in the original instance. □

Visible Triangle

We consider a three-dimensional version of the previous problem with triangles.

> VISIBLE TRIANGLE (VISTRI)
> *Instance:* A set of n horizontal triangles in $\mathbb{Z}^3$ (3-dimensions), a special triangle T, and a given point in $\mathbb{Z}^3$.
> *Question:* Can we see triangle T from that given point? Triangle T. This assumes that the n horizontal triangles are not transparent.

Theorem 17.25. *TRICOVTRI $\equiv_{sq}$ VISTRI, hence VISTRI is 3SUM-hard.*

Proof. We can assume T has z-coordinate 0, and then make all the other triangles have different heights above T. Then, we can let the point be the point of infinity, and we have that T is visible from infinity if and only if the triangles do not cover T. This is depicted in Figure 17.8

We also have a reverse reduction for this problem from this problem to Triangles Cover Triangle. □

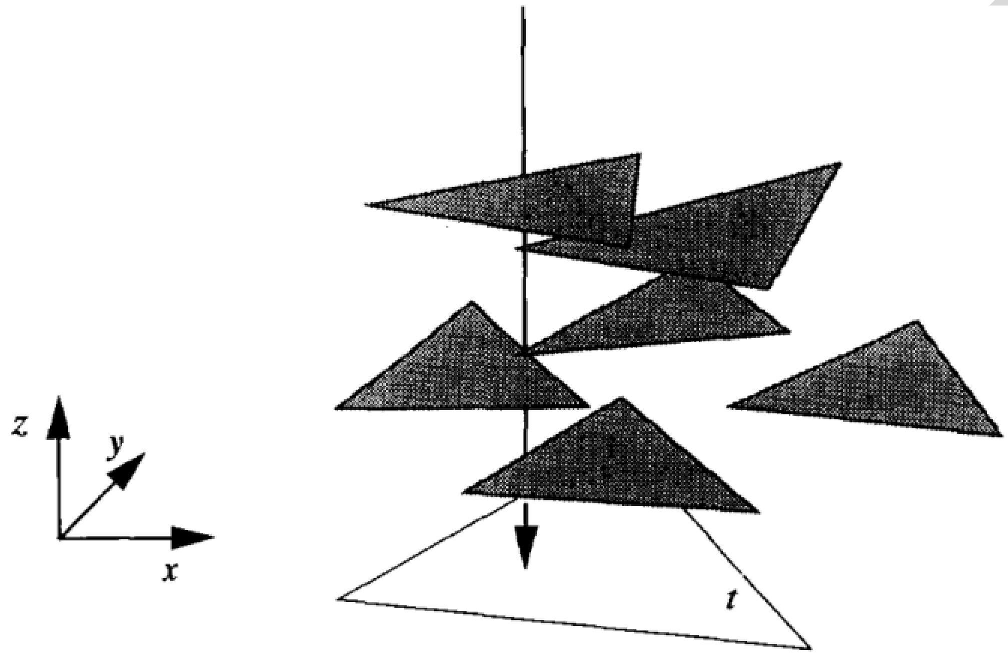


Figure 17.8: Visible Triangle Reduction

Planar Motion Planning

We also consider a group of Motion Planning Problems.

> PLANAR MOTION PLANNING (PLMOTPLAN)
> *Instance:* n line segments, some horizontal and some vertical. (we think of the line segments as obstacles) and two points called ***the source*** and ***the goal***.
> *Question:* Can we move a robot (represented in the form of a line segment) allowing translations and rotations, from the source to the goal without colliding into any obstacles?

Theorem 17.26. *GEOMBASE $\leq_{sq}$ PLMOTPLAN, hence PLMOTPLAN is 3SUM-hard.*

Proof. The reduction is evident from the Figure 17.9. □

3-Dimensional Motion Planning

We can then extend the previous problem into $3D$-space, to get 3-dimensional Motion Planning.

> 3-DIMENSIONAL MOTION PLANNING (3DMOTPLAN)
> *Instance:* A set of n horizontal non-intersecting triangle obstacles in $\mathbb{Z}^3$ and a robot represented as a vertical line segment.
> *Question:* Can the robot move through the obstacles without collision, using translations only?

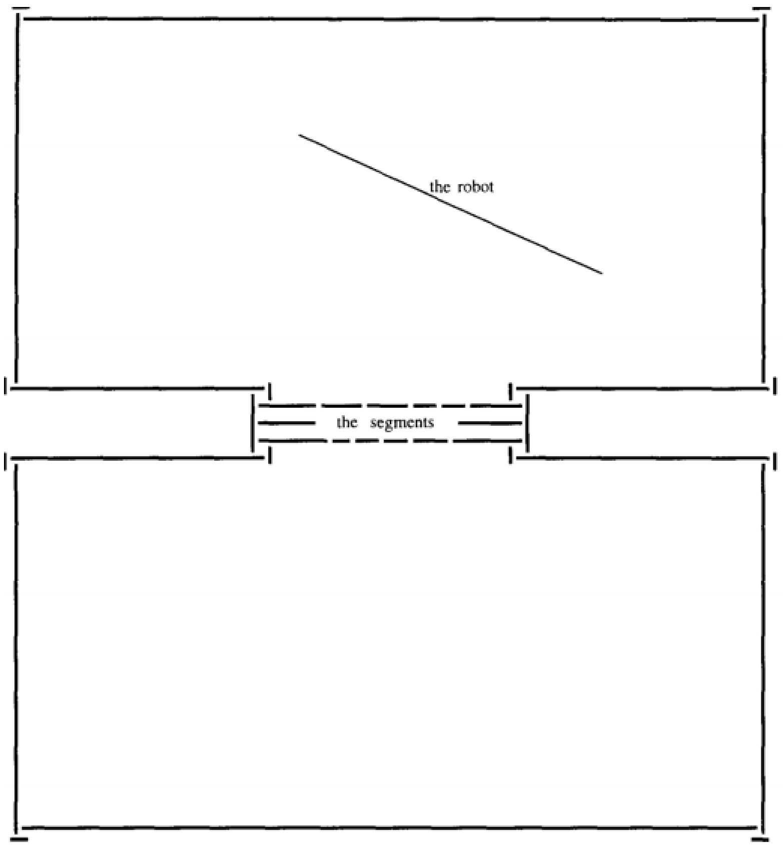


Figure 17.9: Planar Motion Planning

There is an algorithm to solve this problem in $O(n^2 \log n) time$.

Theorem 17.27. *TriCovTri $\leq_{sq}$ 3DMotPlan, hence 3DMotPlan is 3SUM-hard.*

Proof. First we create a cage to prevent the robot from leaving the original triangle T in the original problem instance. Then, we see that we can go from a given source from the top of the cage to the bottom of the cage, if and only if there is a point not covered by a triangle, and we are done. This is depicted in Figure 17.10. □

Fixed-Angle Chains

The results on this problem are due to Soss et al. [SEO03].

Definition 17.28. A ***fixed-angle chain*** is a chain of line segments which follow each other at fixed angles. We can imagine this in 3 dimensions as an object where the segments are attached to each other at joints; the joints have a fixed angle but can rotate freely. In 2 dimensions, since the angles are fixed, we can only change whether the fixed angle is a left-hand turn or a right hand turn. Flipping an angle this way flips the whole subsequent structure as if the chain were a rigid body (see Figure 17.11).

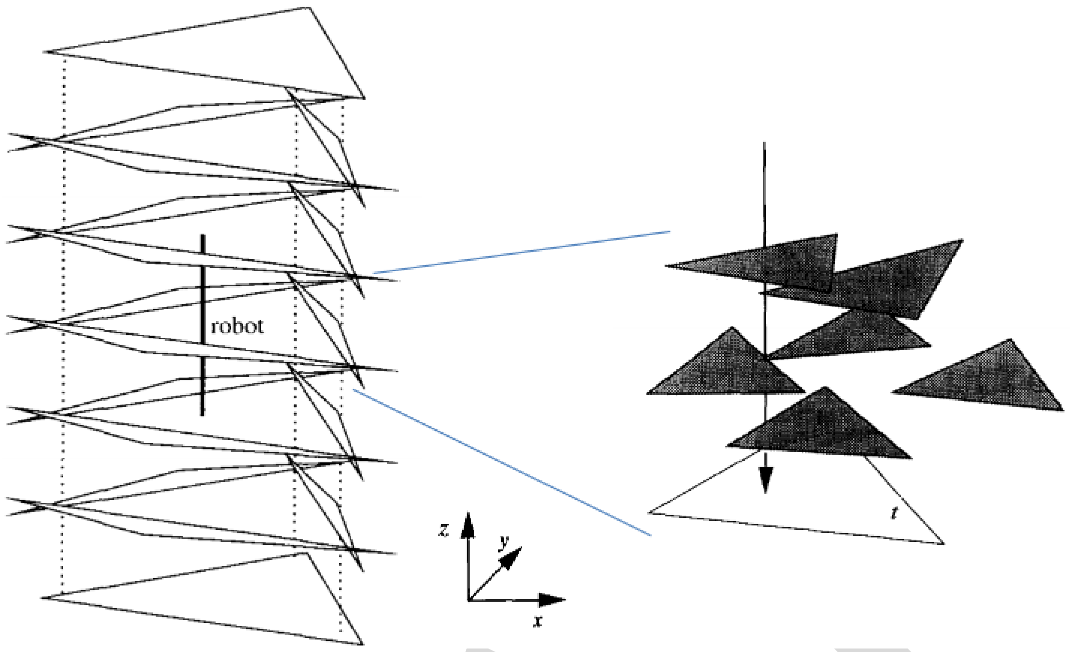


Figure 17.10: 3D Motion Planning

Fixed-Angle Chain (FixAngCh)
Instance: A fixed-angle chain in 2 dimensions. The number of line segments in it is our n.
Question: Is there an angle which can be flipped to cause a collision (two different points on the chain occupying the same point in the plane).

Theorem 17.29. *3SUM′ $\leq_{sq}$ FixAngCh, hence FixAngCh is 3SUM-hard.*

Proof. Given A, B, C, an instance of 3SUM′, we proceed as follows. First find $M = \max(|x| : x \in A \cup B \cup C)$, which is large enough to separate the 3 groups if doubled. We then add $-2M$ to A and $2M$ to $-C$ to get A' and C', and set $B' = -B/2$. We then construct the chain in the following diagram, where the two lines $y = 0$ and $y = 1$ are each broken up by 'teeth' located at the values of the elements of A' and C' respectively; the two lines are joined by a step function where the steps are located at the values of the elements of B'.

It is trivial to verify that, with a careful construction, flipping any edge aside from the vertical edges corresponding to the elements of B' does not cause a collision. Thus, we only need to worry about these vertical segments. Consider flipping the segment at $b' \in B'$; the only segments that might come into contact are the 'teeth'. Suppose that the teeth at $a' \in A'$ and $c' \in C'$ come into contact; flipping horizontally at b' reflects everything to the right across the line $x = b'$, and thus causes $c' \to -(c' - 2b')$, so this only happens if $a' = -(c' - 2b')$ for some $a' \in A', b' \in B', c' \in C'$. But $a' = a - 2M$ for some $a \in A$, $b' = -b/2$ for some $b \in B$, and $c' = -c + 2M$ for some $c \in C$. Thus, we can conclude that

$$a' = -(c' - 2b') \implies a - 2M = -(-c + 2M) - b \implies a + b = c$$

so there is a solution to the 3SUM′ problem if and only if there is a solution to the fixed-angle chains problem for this construction. □

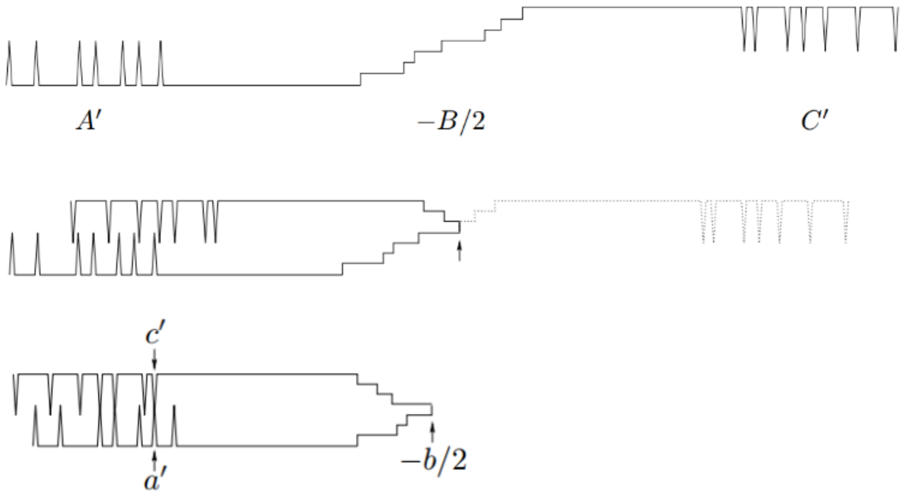


Figure 17.11: Reduction from 3SUM′ to Fixed-Angle Chain

Remark: Setting $C' = C + 2M$ (as opposed to $C' = -C + 2M$) gives a reduction from the 3SUM′ variant where the goal is to find $a + b + c = 0$.

The fastest known algorithm for fixed-angle chains is $O(n^3)$, due to Soss & Toussaint [ST00].

17.5 Other Lower Bounds from 3SUM-Hardness

Pătraşcu [Păt10] showed that some graph problems are 3SUM-hard as we note in the next two theorems that we state without proof.

Theorem 17.30. *Consider the following problem: Given a weighted graph and a number x we want to know whether some triangle has weight x.*

1. *There is an $O(|E|^{3/2})$ algorithm for this problem. (This is obvious.)*

2. *If there is an $O(|E|^{3/2} - \varepsilon)$ algorithm for this problem then there is an $O(n^{2-\delta})$ algorithm for 3SUM.*

Theorem 17.31. *Consider the following problem: Given an (unweighted) graph we want to know whether there are $|E|$ triangle.*

1. *There is an $|E|^{3/2}$ algorithm for this problem (this is obvious).*

2. *If there is an $O(|E|^{4/3} - \varepsilon)$ algorithm for this problem then there is an $O(n^{2-\delta})$ algorithm for 3SUM.*

Exercise 17.32. (This is open.) Narrow the gap between the upper and lower bound in Theorem 17.31

17.6 dSUM and Its Relation to Other Problems

The 3SUM problem can easily be generalized.

> *d*SUM
> *Instance:* A set of n integers.
> *Question:* Do some d of the integers sum to 0?

Exercise 17.33. Show that there is a randomized $O(n^{\lceil \frac{d}{2} \rceil})$ algorithm for dSUM.

Pătraşcu & Williams [PW10] showed that improving the algorithm in Exercise 17.33 is equivalent to other problems being improved. After we state theorem we will comment on what it means.

Theorem 17.34. *Let $d < n^{0.99}$. If dSUM with numbers of $O(d \lg n)$ bits can be solved in $n^{o(d)}$ time, then 3SAT can be solved in $2^{o(n)}$ time (thus violating the ETH).*

We really want to say

Theorem 17.34 is evidence that dSUM is not in time $O(n^{o(d)})$.

However, the authors of the paper are more ambivalent (page 1066) (comments in square brackets are ours).

> We have expended significant effort attempting to either find an improved algorithm [for dSUM and other problems they have results about], or give interesting evidence against its possibility. In this paper, we present several hypothesis which appear plausible [dSUM in time $n^{o(d)}$ is one of them], given the current state of knowledge. We prove that if any of the hypothesis are true then CNF SAT has an improved algorithm [We just presented the improvement on 3SAT]. One can either interpret our reductions as new attacks on the complexity of CNF SAT, or lower bounds (ruling out all hypothesis) conditional on the hardness of CNF SAT [in our case assuming 3SAT is not in time $2^{o(n)}$, we get dSUM is not in time $n^{o(d)}$].

Borassi et al. [BCH16] have shown several problems cannot be done in subquadratic time using a hypothesis that is stronger than ETH, namely the ***strong exponential time hypothesis*** (SETH).

17.7 The Orthogonal Vectors Conjecture

In this chapter we have used the 3SUM-Conjecture as a hardness assumption to obtain quadratic lower bounds. We now look at another hardness assumption to obtain quadratic lower bounds.

> Orthogonal Vectors (OrthVec)
> *Instance:* n vectors in $\{0, 1\}^d$ where $d = O(\log n)$.
> *Question:* Do two of the vectors have an inner product that is 0 mod 2?

A naive algorithm for OrthVec solves it in time $O(n^2 d)$ by trying all possible pairs. Abboud et al. [AWY15] obtained the following slight improvement.

Theorem 17.35. *There is a randomized algorithm of* OrthVec *that runs in time* $O(n^{2-\Omega(\frac{1}{\log(\frac{d}{n})})})$.

There are no known algorithms for the problem that run in time $n^{2-\varepsilon}$ for some $\varepsilon > 0$. This leads to the following conjecture, due to Williams [Wil05]. See also Abboud et al. [AVW14], Backurs et al. [BI15], and Abboud et al. [ABV15]

Conjecture 17.36. The Orthogonal Vectors Conjecture (OVC) *For all* $\varepsilon > 0$, *there is a* $c \geq 1$ *such that* OrthVec *cannot be solved in time* $n^{2-\varepsilon}$ *on instances with* $d = c\log(n)$.

We now have several conjectures with rather concrete bounds in them: ETH, SETH, 3SUM-Conjecture, and now OVC. Clearly SETH $\implies$ ETH. Are any other implications known? Yes. Williams [Wil05] showed the following.

Theorem 17.37. *SETH* $\implies$ *OVC.*

The lack of any subquadratic algorithm for OrthVec, and Theorem 17.37, are evidence for OVC. In addition, OVC holds in several restricted computational models. Kane & Williams [KW19] show:

1. OV has branching complexity $\tilde{\Theta}(n \cdot \min(n, 2^d))$ for all sufficiently large n, d. (Recall that $\tilde{\Theta}(f(n))$ means that we ignore log factors.)

2. OV has Boolean formula complexity $\tilde{\Theta}(n \cdot \min(n, 2^d))$ over all complete bases of $O(1)$ fan-in.

3. OV requires $\tilde{\Theta}(n \cdot \min(n, 2^d))$ wires, in formulas comprised of gates computing arbitrary symmetric functions of unbounded fan-in.

What does OVC imply and vice-versa?

Theorem 17.38. *Assume OVC. Then the following problems do not have subquadratic algorithms.*

1. *(Backurs & Indyk [BI15]) Edit Distance: Given 2 strings* x, y *how many times do you need to delete or insert of replace a letter from either so that at the end the resulting strings are the same. (There are many variants depending on what operations are allowed.)*

2. *(Bringmann [Bri14]) Fréchet Distance: This is a measure of similarity between two curves that takes into account the location and ordering of the points on the curve. The formal definition is rather long so we omit it.*

3. *(Backurs & Indyk [BI16]) Regular Expression Matching: Given a regular expression* p *and a string* t, *does* p *generate some substring of* t.

4. *(Roditty & V. Williams [RV13]) Approximating the diameter of a graph: See Section 18.3 for definition of diameter.*

5. *(Burchin et al. [BBK+16]) Curve Simplification: This has to do with finding a polygonal curve that is close to the original. The formal definition is rather long so we omit it.*

While the above results show that many problems are subquadratic assuming OrthVec is subquadratic, this does not necessarily imply that they are *equivalent* to OrthVec. Chen & Williams [CW19] study equivalences between OrthVec and different problems. They showed the following.

Theorem 17.39. *Each of the following problems is subquadratic-equivalent to OrthVec.*

1. *Min-IP: Given n blue vectors in $\{0,1\}^d$, and n red vectors in $\{0,1\}^d$, find the red-blue pair of vectors with minimum inner product.*

2. *Max-IP: Given n blue vectors in $\{0,1\}^d$, and n red vectors in $\{0,1\}^d$, find the red-blue pair of vectors with maximum inner product.*

3. *Equals-IP: Given n blue vectors in $\{0,1\}^d$, and n red vectors in $\{0,1\}^d$, and an integer k, find a red-blue pair of vectors with inner product k, or report that none exists.*

4. *Red-Blue-Closest Pair: Let $p \in [1,2]$ and $d = n^{o(1)}$. Approximating the ℓ_p-closest red-blue pair among n red points and n blue points in $\mathbb{R}^d$.*

17.8 Lower Bounds on Data Structures via the 3SUM-Conjecture

Imagine that you want a data structure for (1) storing a graph with n vertices and m edges, or (2) a collection of m set within a universe of n elements. There are three issues:

- How long will it take to to set up the data structure? This is called ***preprocessing***.

- How much space will the data structure need?

- How long will it take to update the data structure? There are many update operations you might allow. For graphs vertex operations (add or delete), or edge operations (add or delete). For sets adding an element, deleting an element, adding a set, deleting a set, merging sets, maintaining the min or max (if the elements are numbers) There are other operations as well.

- How long will it take to answer a query? There are many queries you might be interested in. For graphs reachability, number of (strongly) connected components, others. For sets membership is the the key one, though there are others.

- Assume that you want to make L queries where L is large. It may be that some queries take a long time; however, while doing it you modify the data structure a lot, so that later queries are much faster. We don't want to look at the worst case. We want to say that L queries took $\alpha(n)L$ time. The function $\alpha(n)$ is the ***amortized query time***). There are many queries you might be interested in.

- One can also look at ***amortized update time***.

In the context of data structures, *fast* is polylog (or even $O(1)$) and *slow* is n^δ. Often there is a tradeoff.

1. Pătraşcu [Păt10] was the first person to use the 3SUM-Conjecture to get lower bounds on dynamic data structures. We give one of his results:

   Dynamic Reachability The problem is to find a data structure for directed graphs that allows (1) updates: insertion and deletion of edges (but not vertices), and (2) queries: given vertices u, v determines if there is a directed path from u to v. There exists $\delta > 0$ such that, for any data structure for this problem, either updates or queries take $\Omega(n^{\delta})$. All of the later papers build on this paper.

2. Abboud & V. V. Williams [AV14] considered many problems where the 3SUM-Conjecture (or other assumptions) were used to get lower bounds on data structures. We give two of the problems they considered which have the same lower bound assuming the 3SUM-Conjecture.

   ***st*-Reachability** The problem is to find a data structure for a directed graphs and two nodes s, t that allows (1) updates: insertion and deletion of edges (but not vertices), and (2) queries: is there a directed path from s to t?

   Bipartite Perfect Matching The problem is to find a data structure for an undirected bipartite graphs that allows (1) updates: insertion and deletion of edges (but not vertices), and (2) queries: is there a perfect matching?

   For both problems the following holds: For all α there is no data structure that does updates in $O(m^{\alpha})$ and queries in $O(m^{2/3-\alpha})$.

3. Kopelowitz et al. [KPP16] considered many problems. We consider one of them.

   The Static Set Disjointness Problem. The Universe U has n elements. (1) store subsets of U statically so there are no updates, (2) queries: given two sets, are they disjoint?

   They show that if query time is $O(1)$ then preprocessing must take $\Omega(n^{2-o(1)})$,

17.9 $\exists\mathbb{R}$-complete

This section largely draws from a survey by Cardinal[Car15].

Consider the following two question:

$$S_1 = \exists x, y, z \in \mathbb{R} : x^2 + y^z + z^2 < 0.$$
$$S_2 = \exists x, y, z \in \mathbb{R} : x^2 + y^2 + z^2 > 0.$$

Clearly S_1 is false and S_2 is true. We consider the problem where you are given a sentence like S_1 or S_2 and you need to determine whether it is true or false.

Existential Theory of the Reals (ETR)
Instance: A sentence of the form

$$\exists x_1 : \exists x_2 : \cdots : \exists x_n : C_1 \wedge \cdots \wedge C_k.$$

where each C_i is a polynomial equality or inequality in $x_1, \ldots, x_n$ (it might not use all of them). The polynomials are over $\mathbb{Z}$.
Question: If the quantifiers range over $\mathbb{R}$ then is the sentence true?

It is not obvious that ETR is decidable. However, the following are known:

Theorem 17.40.

(Tarski [Tar48]) ETR is decidable. (Tarski had the result in 1930 but did not publish it until 1948.)

(Canny [Can88]) ETR $\in$ PSPACE.

The problem ETR is thought to be hard. Hence the following definitions, due to Schaefer [Sch12], make sense when trying to get a handle on how hard some problems are.

Definition 17.41.

1. A decision problem A is ***in*** $\exists\mathbb{R}$ if $A \le$ ETR.
2. A decision problem A is $\exists\mathbb{R}$***-complete*** if $A \in \exists\mathbb{R}$ and ETR $\le A$.

Schaefer & Stefankovic [SS17] (see also [Mat14]) showed that $\exists\mathbb{R}$ does not change if you restrict the instances of ETR to those with either (1) all of the inequalities or strict, or (2) $k = 1$ and C_1 is of the form $p(x_1, \ldots, x_n) = 0$.

We will present a few problems that are $\exists\mathbb{R}$-complete (without proof). We choose those that do not require much background.

Definition 17.42. Let S be a collection of sets. The ***Intersection Graph for S*** is (V, E) where $V = 2^S$ and $(A, B) \in E$ if and only if $A \cap B \neq \emptyset$.

Our concern will be, given a graph G is it an intersection graph for some S. We will put restrictions on S.

X-SET-RECOGNITION
Instance: A graph G.
Question: There are two versions of this problem.
X=Line Segments: Is G the intersection graph for some S which is a set of segments in the plane? Jan Kratochvíl and Jirí Matoušek [KM94] showed this problem is $\exists\mathbb{R}$-complete.

X=Unit Disks: Is G the intersection graph for some S which is a set of unit disks in the plane? Colin McDiarmid and Fiona Skerman [MS13]. showed this problem is $\exists\mathbb{R}$-complete.

Definition 17.43.

1. A ***linkage*** is a graph $G = (V, E)$ where every edge e is given a length $\ell(e)$.
2. A ***geometric realization of a linkage*** is mapping of $f\colon V \longrightarrow \mathbb{R}^2$ such that if $(x, y) \in E$ then $d(f(x), f(y)) = \ell(x, y)$. Note that we allow f to map two vertices to the same point, and we allow f to map a vertex z to end up on the line between $f(x)$ and $f(y)$.
3. We could also demand that f is an injection and that points never end up between two other points. In this case we do not use terms ***linkage*** and ***realization*** but instead use the terms ***weighted graph recognition***.

Realization and Recognition Problems
Instance: A linkage/weighted graph G.
Question: (Realization) Is there a geometric realization of the linkage G? Schaefer [Sch12] showed this was $\exists\mathbb{R}$-complete even if the edges all have unit length.
Question: (Recognition) Is there a geometric recognition of the weighted graph G? Schaefer [Sch12] showed this was $\exists\mathbb{R}$-complete even if the edges all have unit length.

17.10 Further Results

Barequet & Har-Peled [BH01] proved that the following problems (and more) are 3SUM-hard. They are from computational geometry. Some care must be taken to define these problems rigorously since the inputs are real numbers. We ignore such issues.

1. Given two simple polygons P and Q, determine whether P can be translated to fit inside Q.
2. Given two simple polygons P and Q, determine whether P can be translated and rotated to fit inside Q.
3. Given two simple polygons P and Q, determine whether P can be rotated around a given point to fit into Q
4. Given P, a finite sets of reals, and a set S of intervals of real numbers, Determine whether there a $u \in \mathbb{R}$ so that $P + u \subseteq S$?

Aronov & Har-Peled [AH08] study the problem of finding the "deepest" point in an arrangement of disks, where the ***depth*** of a point denotes the number of disks that contain it. They show this problem is 3SUM-hard.

Abboud et al. [AVW14] consider the local alignment problem: given two input strings and a scoring function on pairs of letters, one is asked to find the substrings of the two input strings that are most similar under the scoring function. They show that if there exists $\varepsilon > 0$ and an $O(n^{2-\varepsilon})$ algorithm for this problem then there exists a $\delta > 0$ such that (a) 3SUM can be done in $O(n^{2-\delta})$ time (so the 3SUM-Conjecture is false), (b) CNF SAT can be done in $O((2-\delta)^n)$ time (so SETH is false), (c) the following problem can be done on $O(4-\delta)^n)$ time (which people who have looked at it do not think it can be done): The Max Weight k-Clique problem, which is, given a weighted graph, find a k-clique of max weight or say there is no k-clique.

Chapter 18

The APSP-Conjecture: A Method for Obtaining Cubic Lower Bounds

18.1 APSP: All Pairs Shortest Paths

In Chapter 17 we used the assumption that 3SUM cannot be solved in subquadratic time to prove that many other problems can not be solved in subquadratic time. In this section we use the assumption that the All Pairs Shortest Paths Problem (APSP) cannot be solved in subcubic time to prove that there are many other problems that can not be solved in subcubic time.

Recall that big-O notation is used when you want to ignore constants. We need a notation for when we want to ignore log factors.

Notation 18.1. $f = \tilde{O}(g)$ means that there exists $n_o, c \in \mathbb{N}$ such that, for all $n \geq n_o$, $f(n) \leq (\log n)^c g(n)$.

Notation 18.2.

1. We denote a weighted directed graph by $G = (V, E, w)$ where w is the weight function. The weights will always be integers. They will be nonnegative except for the negative triangle problem.

2. We use n for the number of vertices and m for the number of edges.

3. Let $G = (V, E, w)$ be a weighted directed or undirected graph with weights in $\mathbb{N}$. Let $x, y \in V$. Then $\mathrm{dist}_G(x, y)$ is the length of the shortest path between x and y. We will sometimes use $\mathrm{dist}(x, y)$ when G is clear.

Note: This chapter will deal with weighted directed graphs. Most of what we prove is true for weighted undirected graphs. In Exercise 18.15 we will invite the reader to redo the entire chapter with variants of weighted directed graphs.

All Pairs Shortest Paths (APSP)
Instance: A weighted directed graph $G = (V, E, w)$. The weights are in $\mathbb{N}$.
Question: For all pairs of vertices x, y compute $\mathrm{dist}_G(x, y)$.
Note: We will consider any arithmetic operation to have unit cost.

This problem is a very important and central problem in graph theory. There is a well-known $O(n^3)$ algorithm for this problem due to Floyd and Warshall. The algorithm is as follows:

```
Data: A graph $G = (V, E, w)$ given as an adjacency matrix $w(i, j)$
Result: The matrix dist.
$\forall i, j \in V$: dist$(i, j) \leftarrow w(i, j)$ ;
for $k$=1 to $n$ do
    for $i$=1 to $n$ do
        for $j$ =1 to $n$ do
            if $dist(i, j) > dist(i, k) + dist(k, j)$ then
                dist$(i, j) \leftarrow$ dist$(i, k)$ + dist$(k, j)$ ;
            end
        end
    end
end
```

Algorithm 1: Floyd-Warshall Algorithm

Another way to compute the all pair shortest path is by invoking the Dijkstra's algorithm for each vertex. Dijkstra's algorithm finds the shortest path from a given vertex v to all other vertices in the graph. Hence, invoking it n times by choosing a different starting vertex v each time, will result in calculating the all pair shortest path. A single invocation of the Dijkstra's algorithm takes $O(n + m \log m)$. Hence, n iterations of this algorithm takes a time of $O(nm + n^2 \log m)$. In the worst case, m can be $O(n^2)$ and hence the worst case running time using this approach is also $O(n^3)$.

Is there an algorithm for APSP that runs in time better than $O(n^3)$? This depends on your definition of "better". The following are known:

1. Fredman [Fre76] gave a $O\left(\frac{n^3 \sqrt[3]{\log \log n}}{\sqrt[3]{\log n}}\right)$ deterministic algorithm.
2. Williams [Wil18] gave a randomized algorithm which ran in time $O\left(\frac{n^3}{2^{\Omega(\sqrt{\log n})}}\right)$. There were many results between Fredman (1976) and Williams (2018).
3. Zwick [Zwi98] showed that APSP can be approximated well: For all $\varepsilon > 0$ there exists a $(1+\varepsilon)$-approximation to the APSP problem running in time $\tilde{O}(\frac{n^\omega}{\varepsilon})$, where ω is the exponent in the running time of the fastest known matrix multiplication problem (i.e. $\omega < 2.3728639$).
4. Chan & Williams [CW21] found an algorithm for APSP in deterministic time $\frac{n^3}{2^{\Omega(\sqrt{\log n})}}$ time, matching the randomized algorithm listed above.

While the algorithms above are impressive and very clever, the first two are not that much better than $O(n^3)$, and the third is an approximation. We need a definition for "not that much better than $O(n^3)$".

Definition 18.3. An algorithm is ***subcubic*** if there exists $\varepsilon > 0$ such that it runs in time $O(n^{3-\varepsilon})$.

Note that neither Fredman's or William's algorithm is subcubic.

Despite enormous effort nobody has obtained a subcubic algorithm for APSP. In the next section we make that a conjecture.

18.2 The APSP-Conjecture

Conjecture 18.4. *The **APSP-Conjecture**: There is no subcubic algorithm for APSP.*

We define a notion of reduction between problems.

Definition 18.5. Let A and B be sets or functions (they will almost always be sets).

1. $A \leq_{sc} B$ means that if there is a subcubic algorithm for B then it can be used to obtain a subcubic algorithm for A. Analogies to $\leq_p$ are not quite right since subcubic is not closed under composition. Hence we proceed more formally than we would like to.

   (a) (The usual way to do this.) On input x produce in linear time a y such that $x \in A$ if and only if $y \in B$. We insist on linear time since $|y| \leq O(|x|)$ so subcubic in $|y|$ is subcubic in $|x|$. It would suffice to have the algorithm be subcubic but the output be linear; however, this seems to never be the case in practice.

   (b) (This sometimes is needed.) On input x produce in linear time $y_1, \ldots, y_k$ (k is a constant) such that $x \in A$ can be determined from the answers to $y_1 \in B, \ldots, y_k \in B$.

   (c) We leave it to the reader to modify the above definitions for the case where the problem is a function instead of a set.

   The *sc* stands for ***sub-cubic***.

2. $A \equiv_{sc} B$ if $A \leq_{sc} B$ and $B \leq_{sc} A$. We often use the terminology ***A and B are subcubic equivalent***.

Definition 18.6. Let A be a problem.

1. A is ***APSP-hard*** if APSP $\leq_{sc} A$.

2. A is ***APSP-complete*** if A is APSP-hard and $A \leq_{sc}$ APSP.

Because of Conjecture 18.4 we think that if A is APSP-hard then there is no subcubic algorithm for A. In brief:

1. When you read "A is APSP-complete" you should think: A is in cubic time but not in subcubic time.

2. When you read "A is APSP-hard" you should think: A is in not in subcubic time.

18.3 Problems of Interest: Centrality Measures

We define several measures on graphs that are called ***Centrality Measures*** (the reason for the name will be clear once we define the measures). These measures appear in a variety of applications such as social network, transportation and allocation problems, biological networks, etc. Hence, calculating these measures efficiently has a lot of real life implications. All of these measures have trivial $O(n^3)$ algorithms (they all begin by first doing APSP). We will later show reductions between them and some other problems. The goal is to get them to be subcubic equivalent to APSP; however, alas, that is an open problem.

Definition 18.7. Let G be a weighted directed graph and v be a vertex. Look at all of the distances from v to the other vertices. We denote the max of these by α_v.

> RADIUS AND CENTER
> *Instance:* An weighted directed graph $G = (V, E, w)$.
> *Question:* Find the radius of G which is $\min_v \alpha_v$, and the center of G which is the vertex v which minimizes α_v.

> DIAM
> *Instance:* A weighted directed graph $G = (V, E, w)$.
> *Question:* Find the diameter of G which is the maximum possible distance between any two vertices in the graph. Mathematically, $\max_{u,v} \mathrm{dist}(u, v)$.

> MEDIAN
> *Instance:* A weighted directed graph $G = (V, E, w)$.
> *Question:* Find the median of the graph which is the minimum value over $v \in V$ of the sum of distances from v to the other vertices. Mathematically, $\min_v \sum_u \mathrm{dist}(v, u)$.

18.4 Other Measures

We now define another measure called the betweenness centrality. This measure determines how useful a vertex is to shortest paths in the graph.

Definition 18.8. Let $G = (V, E, w)$ be a weighted directed graph and $s, t, x \in V$.

1. $\textsc{BentCent}_{s,t}(x)$ is the fraction of shortest paths between s and t that contain x. *BC* stands for ***Betweenness Centrality***

2. $\textsc{BentCent}(x)$ is $\sum_{s,t \in V} \textsc{BentCent}_{s,t}(x)$.

3. $$\textsc{PosBetCent}(x) = \begin{cases} 1 & \text{if } \textsc{BentCent}(x) > 0 \\ 0 & \text{otherwise} \end{cases}$$

   PosBetCent stands for ***Positive Betweenness Centrality***. If the graph G is not understood we sometimes write $\textsc{PosBetCent}(G, x)$. In simpler terms: $\textsc{PosBetCent}(x) = 1$ if and only if x is on some shortest path.

> POSITIVE BETWEENNESS CENTRALITY (POSBETCENT)
> *Instance:* A weighted directed graph $G = (V, E, w)$ and an $x \in V$.
> *Question:* What is $\textsc{PosBetCent}(G, x)$? In other words, is x on some shortest path?

The last problem we present is not a measure; however, it will be useful.

> NEGATIVE TRIANGLE (NEGTRI)
> *Instance:* A weighted directed graph $G = (V, E, w)$ with weights in $\{-M, \ldots, M\}$.
> *Question:* Is there a triangle with weight-sum negative.

18.5 Subcubic Equivalence

It is clear that all the problems in Sections 18.3 and 18.4 are $\le_{sc}$ APSP and hence in time $O(n^3)$. Do any of them have a sub-cubic algorithm? The following known theorem shows that, assuming the APSP-Conjecture, no.

Theorem 18.9. *(Abboud et al. [AGV15].) Radius, Median, BentCent, and NegTri are APSP-complete.*

Proof sketch:

For all of the problems mentioned here the proof that they are subcubic reducible to APSP is easy.

The other directions are obtained by the combined efforts of Abboud et al. [AGV15] and Williams & Vassilevska Williams [VW18].

▮

Note: See Boroujeni et al. [BDE+19b] for evidence that Diam subcubic-complete. The status of Diam is still open.

We will prove some other subcubic reductions. From Theorem 18.9 and what we prove in the next few sections we will have Figure 18.1.

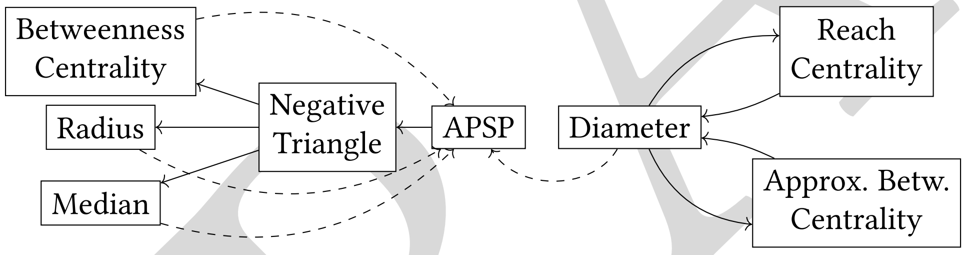


Figure 18.1: Subcubic Equivalence Between Various Problems

See Figure 18.1 for a diagram of reductions. We explicitly mark the ones which are non-trivial and the ones that are folklore.

18.6 Diam and PosBetCent are Subcubic Equivalent

Theorem 18.10. *Diam $\le_{sc}$ PosBetCent.*

Proof. Figure 18.2 is an example of the reduction. Here is the reduction.

1. Input a weighted directed graph $G = (V, E, w)$. Without loss of generality, let us assume that all distances, and hence, the diameter are even (Multiply all distance by 2 initially, and divide the finally obtained diameter by 2). Let M be the maximum weight.

2. (This is not part of the algorithm, this is a definition we will use later.) For $1 \le D \le M$ let $G_D = (V_D, E_D)$ be the following graph:

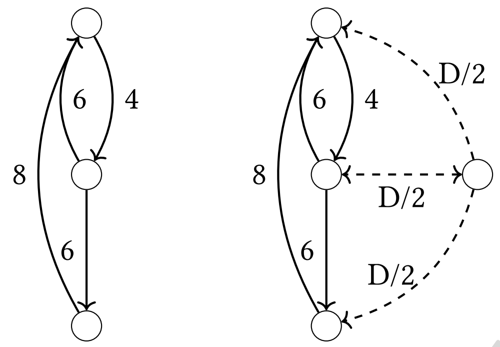


Figure 18.2: (a) Original Graph G (b) Transformed Graph G'

$V_D = V \cup \{x\}$ i.e. add a new vertex labeled x;

$E_D = E \cup \{(x, u, \frac{D}{2}) \mid u \in V\}$.

The largest possible value of D such that $\textsc{PosBetCent}(G_D, x) = 1$ is the required diameter of the original graph G. Note that $\textsc{PosBetCent}(G_0, x) \geq \cdots \geq \textsc{PosBetCent}(G_M, x)$.

3. Perform a binary search on D to find the largest D such that $\textsc{PosBetCent}(G_D, x) = 1$. Output that D.

Now doing a binary search on the values of D, to find the largest value of D such that $\textsc{PosBetCent}(x) = 1$. In the example of Figure 18.2 we get $D = 10$.

□

Theorem 18.11. *$\textsc{PosBetCent} \leq_{sc} \textsc{Diam}$.*

Proof. We present the reductions.

1. Input a weighted directed graph $G = (V, E, w)$ and $x \in V$. We assume G is a complete graph and all edges have weights, though some may be 0. Let M be the max weight and let $\tilde{D} = 3M|V|$, which is much bigger than any shortest path in G.

2. Using Dijkstra's algorithm compute $\text{dist}_G(v, x)$ and $\text{dist}_G(x, v)$ for all $v \in V$. This takes subcubic time. We will use it later.

3. We create a graph $G' = (V', D')$ as follows.

   $V' = V \cup \{v_a\ v \in V - \{x\}\} \cup \{v_b \mid v \in V - \{x\}\}$ (so there are three copies of every $v \in V$).

   $E' = E \cup E_1 \cup E_2 \cup E_3 \cup E_4$ where E_1, E_2, E_3, E_4 are the following sets of weighted edges:

   $E_1 = \{(v_a, v) \mid v \in V - \{x\}\}$ with $w'(v_a, v) = \tilde{D} - \text{dist}_G(v, x)$.

   $E_2 = \{(v, v_b) \mid v \in V - \{x\}\}$ with $w'(v, v_b) = \tilde{D} - \text{dist}_G(x, v)$.

   $E_3 = \{(v, v_a) \mid v \in V\}$ with $w'(v, v_a) = 0$.

   $E_4 = \{(v_b, v) \mid v \in V\}$ with $w'(v_b, v) = 0$.

   See Figure 18.3 for an example.

4. Let $D = \textsc{Diam}(G')$.

5. (This is not part of the algorithm. This is commentary.) The longest path in G' has to go from some s_a to some t_b. We can assume the path is of the form

$$s_a \to s \Rightarrow t \to t_b$$

where $s_a \to s$ is and edge, $s \Rightarrow t$ is a path in G, and $t \to t_b$ is an edge. Hence, in G', the distance of the path from s_a to t_b is

$$(\tilde{D} - \mathrm{dist}_G(s,x)) + \mathrm{dist}_G(s,t) + (\tilde{D} - \mathrm{dist}_G(x,t)) = 2\tilde{D} + \mathrm{dist}_G(s,t) - \mathrm{dist}_G(s,x) - \mathrm{dist}_G(x,t).$$

Since $\mathrm{dist}_G(s,t) - \mathrm{dist}_G(s,x) - \mathrm{dist}_G(x,t) \le 0$,

$$\forall s,t \in V : 2\tilde{D} + \mathrm{dist}_G(s,t) - \mathrm{dist}_G(s,x) - \mathrm{dist}_G(x,t) \le \mathrm{dist}_{G'}(s_a,t_b) \le 2\tilde{D}.$$

Note the following:

(a) If PosBetCent$(x) = 1$ then there exists $s,t \in V$ such that $\mathrm{dist}_G(s,t) = \mathrm{dist}_G(s,x) + \mathrm{dist}_G(x,t)$, hence $\mathrm{dist}_{G'}(s_a,t_b) = 2\tilde{D}$.

(b) If PosBetCent$(x) = 0$ then for all $s,t \in V$, $\mathrm{dist}_{G'}(s,t) < \mathrm{dist}_G(s,x) + \mathrm{dist}_G(x,t)$, hence $\mathrm{dist}_{G'}(s_a,t_b) < 2\tilde{D}$.

6. If $D = 2\tilde{D}$ then output YES, else output NO.

□

18.7 NegTri

Theorem 18.12. *NegTri* $\le_{sc}$ *Radius.*

Proof. Here is the reduction. The graph we construct will be undirected. Since formally Radius is a set of directed graphs, one can take each edge and $\{x,y\}$ and make two directed edges (x,y) and (y,x).

1. Input a weighted directed graph $G = (V,E,w)$ with weights in $\{-M,\dots,M\}$. Let $Q = 3M$.

2. We create a graph $G' = (V',E')$ as follows (See Figure 18.4).

   x is a new vertex. V_a, V_b, V_c, V_d are each copies of V.

   If $v \in V$ then v_a (v_b, v_c, v_d) is the analog of v in V_a (V_b, V_c, V_d).

   $V' = \{x\} \cup V_a \cup V_b \cup V_c \cup V_d$

   There is an edge of weight $2Q + M$ from x to each vertex of V_a.

   For all $(u,v) \in E$ we put edges of weight $Q + w(u,v)$ between (1) u_a, v_b, (2) u_b, v_c, (3) u_c, v_d.

   For all $(u,v) \in E$ we put edges of weight $2Q$ between u_a and v_d.

   There are no edges within V_a or V_b or V_c or V_d.

$$\tilde{D} = 3 \times 7 \times 4 = 84$$

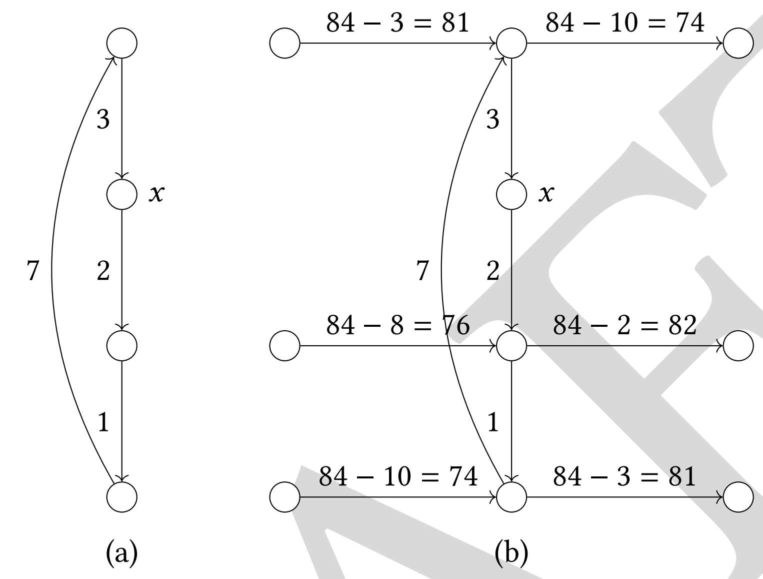


Figure 18.3: (a) Graph G (b) Graph G'

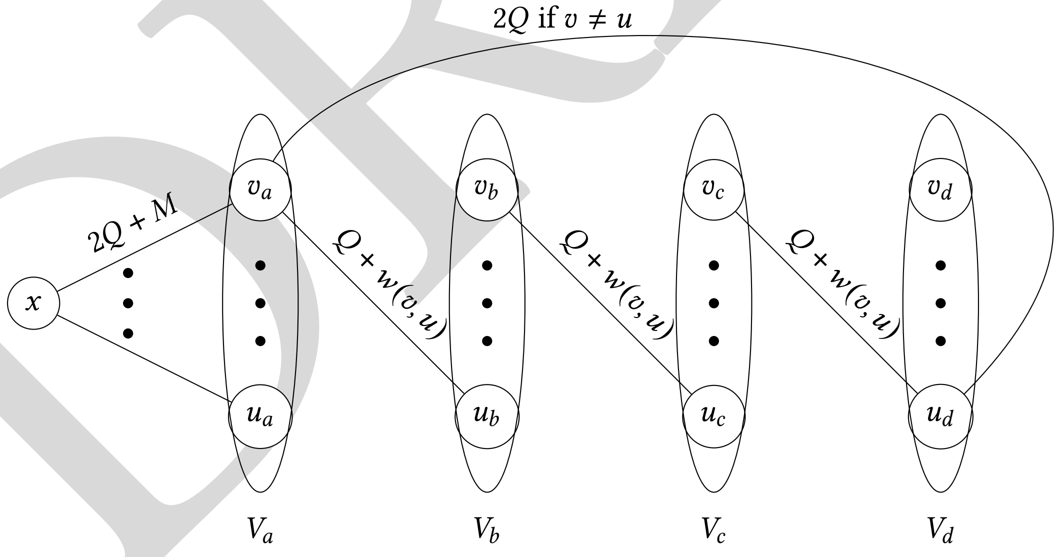


Figure 18.4: Reduction of NegTri to Radius

3. If the radius of G' is $< 9M$ then output YES (there is a negative triangle). Else output no. We prove this works below.

Claim 1. $\textsc{Radius}(G') < 9M$ if and only if G has a negative triangle.

Proof. Let $R = \textsc{Radius}(G')$.

We will make and prove four observations which will essentially lead to the claim.

- **Observation 1**: Any vertex of the form v_b, v_c, or v_d is more than $9M$ away from x. Hence, if $R < 9M$, then the center of the graph is contained in V_a.

  We look at v_b, v_c, and v_d.

  If $v_b \in V_b$ then there exists u_a such that

$$\begin{aligned}\text{dist}_{G'}(v_b, x) &= \text{dist}_{G'}(v_b, u_a) + \text{dist}_{G'}(u_a, x)\\ &= (Q + w(v,u) + (2Q + M)\\ &= 10M + w(v,u)\\ &\geq 9M.\end{aligned}$$

  A similar arguments work for v_c and v_d.

- **Observation 2**: Let $v_a \in V_a$ and let $z \in V' - \{v_b, v_c, v_d\}$. Then $\text{dist}_{G'}(v_a, z) \leq 8M$.

  There are cases:

  $z = x$. Then $\text{dist}_{G'}(v_a, x) = 2Q + M = 7M < 8M$.

  $z = u_b$. Then $\text{dist}_{G'}(v_a, u_b) = Q + w(v,u) \leq 3M + M = 4M < 8M$.

  $z = u_c$. Then there exists $w \in V$ such that

$$\begin{aligned}distGp(v_a, u_c) =\ & \text{dist}_{G'}(v_a, u_b) + \text{dist}_{G'}(u_b, u_c)\\ =\ & (Q + w(v,w) + (Q + w(w,u) = 6M + w(v,w) + w(w,u)\\ \leq 8M\end{aligned}$$

  $z = u_d$. Then the shortest distance to x uses the edge from v_a to u_d, which has distance $2Q = 6M$.

  $z = u_a$. Then $\text{dist}_{G'}(v_a, u_a)$ is determined by going from v_a to u_b and then from u_b to v_a, so

$$\text{dist}_{G'}(v_a, u_a) \leq Q + w(v,u) + Q + w(u,v) \leq 3M + M + 3M + M = 8M.$$

- **Observation 3**: If vertex v in graph G was present in a negative triangle, then $\text{dist}_{G'}(v_a, v_b) < 3Q = 9M$.

  Let the triangle by v, w, x. So $w(v,w) + w(w,x) + w(x,v) < 0$. Then $\text{dist}_{G'}(v_a, v_b)$ can be bounded by the route that goes from v_a to w_b, then w_b to x_c, then x_c to v_b, so

$$\begin{aligned}\text{dist}_{G'}(v_a, v_b) &\leq (Q + w(v,u)) + (Q + w(u,x)) + (Q + w(x,v))\\ &= 3Q + w(v,u) + w(u,x) + w(x,v) < 9M\end{aligned}$$

- **Observation 4**: Finally, if a vertex v is not in a negative triangle in the original graph, then $\mathrm{dist}_{G'}(v_a, v_b) \geq \min\{3Q, 4Q - 2M\} = 9M$.

  We leave this to the reader.

The claim follows easily from the above four observations. □

Given this claim, we need to construct the graph H and check if the radius in this new graph is strictly less than 9M. This completes the reduction. □

Exercise 18.13. Give a sub-cubic reduction from Negative-Triangle to Median.

18.8 Connection to SETH

The Strong Exponential Time Hypothesis (SETH) states that there is no $\delta < 1$ such that SAT can be solved in time $O(2^{\delta n})$. We mentioned this (in a different form) in Hypothesis 7.2. Note that SETH implies P ≠ NP.

Roditty & Vassilevska Williams [RV13] obtained a running time bound on computing the approximate value of the diameter and a lower bound assuming SETH. We state both.

Theorem 18.14.

1. *There is an expected run time $\tilde{O}(m\sqrt{n})$ algorithm for 1.5-approximation for* Diam. *Note that if $m = \Theta(n^{1.5})$ (which could be called quasi-sparse but never is) this is an $\tilde{O}(n^2)$ algorithm.*

2. *Assume SETH. There is no ε such that there is an $O(m^{2-\varepsilon})$ time, 1.5-approximation algorithm, for the diameter of a graph.*

Exercise 18.15. This chapter dealt with directed weighted graphs. There are three other options: undirected weighted graphs, directed unweighted graphs, and undirected unweighted. For each of those options try to redo this entire chapter. Which theorems are true with similar (or even the same) proofs?

18.9 Further Results

18.9.1 More APSP-Complete and APSP-Hard Problems

1. The Matrix Product Verification Problem is as follows: Given matrices A, B, C verify that $AB = C$ where the product is over the $(\min, +)$-semiring. Williams & Vassilevska Williams [VW18] showed this problem is APSP-complete.

2. The Replacement Paths Problem is as follows: Given weighted directed graph G, vertices s, t, and a shortest (s, t)-path P compute the length of the shortest (s, t)-path that does not use any edge from P. Williams & Vassilevska Williams [VW18] showed this problem is APSP-complete.

3. Minimum weight cycle in graph of non-negative edge weight: Given a weighted graph with nonnegative edge weights, find the minimum weight cycle in the graph. Williams & Vassilevska Williams [VW18] showed that this problem is APSP-complete.

4. Second shortest simple path is as follows: Given a weighted directed graph G, and two nodes s and t, find the second shortest simple path between s and t in G. Williams & Vassilevska Williams [VW18] showed that this problem is APSP-complete.

5. CoDiameter: Given a graph G, the goal of CoDiameter is to report a vertex which does not participate in an edge of length equal to the diameter of G. Boroujeni et al. [BDE+19b] showed that this problem is APSP-complete by a reduction from APSP. Boroujeni also defines CoRadius, CoRadius, CoNegativeTriangle, and CoMedian are APSP-complete, and show them APSP-complete.

6. APSP Verification: Given a graph G and a matrix D, determine whether D is the correct distance matrix for G. That is, check that

$$\forall (i,j) \in E : D_{i,j} = dist_G(i,j).$$

   It is known that this problem is APSP-complete.

7. The Tree Edit Problem is (informally) as follows: Given two trees, what is the least number of changes needed to get one from the other. Bringmann et al. [BGMW20] show the this problem is APSP-hard.

8. The Metricity problem is as follows: Given an $n \times n$ nonnegative matrix A, determine whether it defines a metric space on $[n]$, i.e. if A is symmetric, has 0s on diagonal and entries satisfy the triangle inequality. Williams & Vassilevska Williams [VW18] showed this problem is APSP-hard.

18.9.2 Using the Unweighted APSP Problem For a Hardness Assumption

The next two problems have as their hypothesis that the unweighted APSP problem is hard. Both results are by Lincoln et al. [LPV20].

1. The All Edges Monochromatic Triangle Problem is as follows. Given an n-node graph G with edges labeled a color from 1 to n^2, decide for each edge if it belongs to a monochromatic triangle, a triangle whose 3 edges have the same color. If this problem has a $T(n)$ time algorithm then the unweighted APSP has an $O(T(n)\log^n)$ time algorithm.

2. The Min-Max Product Problem is as follows. Given two matrices A, B compute the min max matrix C where $C_{i,j} = \min_k \max(A_{ik}, B_{kj})$. If this problem has a $T(n)$ algorithm then the Unweighted APSP problem has a $O(T(n)\log n)$ time algorithm.

18.9.3 Unusual Hardness Assumptions, Proofs, or Results

1. The All-Pairs Min Cut problem asks: Given a graph G compute, for every pair of vertices s,t, find a min s-t cut. Abboud et al. [AGI+19] showed that this problem has a super-cubic lower bound of $n^{\omega-1-o(1)}k^2$ from a reduction from 4-clique (instead of APSP).

2. The Dynamic Shortest Paths problem asks: preprocess a planar graph G such that insertions/deletions of edges are supported as well as distance queries between two nodes u, v assuming the graph is planar at all time steps. Abboud & Dahlgaard [AD16] showed the following: Assume the APSP cannot be solved in subcubic time. for all $\varepsilon > 0$, dynamic shortest path problem cannot be solved in time $O(n^{\frac{1}{2}-\varepsilon})$.

3. Boolean matrix multiplication (BMM): If boolean matrix multiplication has a sub cubic combinatorial algorithm, then so does the triangle detection problem in graphs. This was shown by Williams & Vassilevska Williams [VW18]. All known algorithms for triangle detection take cubic time, hence using the hardness of triangle detection as an assumption is reasonable.

4. CoAPSP Verification: Given a graph G and a matrix D, either find a pair (i, j) such that $D_{i,j}$ is equal to the distance between vertices i and j in G, or determine that there is no such pair. Boroujeni et al. [BDE+19b] gave a subcubic reduction from Diam to it. Recall that Diam is thought to require cubic time; however, we do not know whether Diam is APSP-hard.

5. $\{-1, 0, 1\}$ – APSP is as follows: Given a weighted directed graph with edge weights in $\{-1, 0, 1\}$, compute the APSP. Despite the complication of having negative edge weights, this problem has a subcubic ($O(n^{2.52})$) algorithm given by Zwick [Zwi02]. This problem seemed to require cubic time but did not. Consider that a cautionary note.

18.9.4 Using the OR of APSP, SETH, and 3SUM-Conjecture

In Chapter 7 we used ETH and SETH as assumptions. In Chapter 17 we used the 3SUM-Conjecture conjecture as an assumption. In this section we used the APSP-Conjecture as an assumption. We think all of these assumptions, that is, the AND of the assumptions, is true. Abboud et al. [AVY18] assumed the OR of the assumptions is true. That is, they assumed that at least one of the APSP-Conjecture, SETH, and the 3SUM-Conjecture, is true. They looked at four problems involving data structures for graphs or directed graphs. All four of them had one type of query. In all four, the lower bound was the same: you must have either amortized query time $n^{1-o(1)}$, or amortized updated time $n^{1-o(1)}$, or preprocessing time $n^{3-o(1)}$. We present the four problems.

1. Directed Graphs. Edge updates. The number of strongly connected components. Denoted $\#SCC$.

2. Directed Graphs with one node s specified. Edge updates. The number of nodes reachable from s. Denoted $\#SSR$.

3. Undirected Graphs with one nodes s specified. Node updates. The number of nodes adjacent to s. Denoted $\#SS$-Subgraphs-connectivity.

4. Directed Graphs with weights in $\{1, \ldots, n\}$ and two nodes s, t specified. What is the max flow from s to t. Denoted Max Flow.

18.10 Open Problems

Some of the open problems which are closely related to these topics are as follows:

- Are Diam and APSP subcubic equivalent?
- Is there a theorem for approximating Radius and Median similar to the one given by Roditty and Vassilevska Williams [RV13] for Diam.
- Lastly, the big open problem is does there exist a truly sub-cubic time algorithm to the APSP problem?

Chapter 19

Lower Bounds for Online Algorithms

19.1 Introduction

For all of the problems studied so far the following were true:

1. The algorithm is given the entire input all at once.
2. The answer is a string (e.g., YES or NO or an assignment that satisfies the max number of clauses).

An ***online problem*** is one where the following are true:

1. The input is given in pieces (e.g., requests for memory access).
2. The answer is a sequence of answers given each time you see more information (e.g., assignments of whether to put a page in cache or memory).

We give a small example.

Example 19.1. There are bins of size 10. Numbers in $\{1, \ldots, 10\}$ will be coming in and you want to pack them into bins (sum in a bin is ≤ 10) to use as few bins as possible. But as soon as you see a number you must put it into an existing bin or create a new one. We give two algorithms.
First Fit Put a number in the least indexed bin that it fits, and if there is none then create a new bin.

On input $3, 3, 3, 3, 3, 3, 4, 4, 4$ the result is as follows.

1. Bin 1: 3,3,3.
2. Bin 2: 3,3,3.
3. Bin 3: 4,4.
4. Bin 4: 4.

This is not optimal since that would be 3,3,4 then 3,3,4, then 3,3,4. But the algorithm cannot see ahead.
First Fit with a 1-Rule Put a number in the least indexed bin that it fits, UNLESS when you do this you have a bin with exactly 9.

On input $3, 3, 3, 3, 3, 3, 4, 4, 4$ the result is optimal:

1. Bin 1: 3,3,4.
2. Bin 2: 3,3,4.
3. Bin 3: 3,3,4.

There are other inputs where this algorithm is not optimal. For example, on input 3,3,3,2,8 the result is as follows.

1. Bin 1: 3,3,2.
2. Bin 2: 3.
3. Bin 3: 8.

This is not optimal since that would be 3, 3, 3 and 2, 8.

In analyzing online algorithms, we usually, do not really care about the running time. The source of hardness here is lack of information, not complexity assumptions like $P \neq NP$. Since the algorithm does not know the rest of the input it may not be able to make the optimum decisions.

In online problems, we want to have a *good solution*. How to measure a *solution*? We compare the outcome of an online algorithm with the best possible offline solution. This is the notion of ***competitive ratio***, which is defined below:

Definition 19.2. Suppose, someone knows the whole input and finds an optimum solution (OPT). We want to compare ourselves with this optimum. Let OPT denote the cost of an optimal offline solution. Let Algorithm denote the cost of our algorithm. α-competitive ratio is defined as follow:

1. (in minimization): for all σ, $\textsc{Algorithm}(\sigma) \leq \alpha \text{OPT}(\sigma) + \gamma$. Note that $\alpha \geq 1$ and we the smaller it is, the better the algorithm.

2. (in maximization): For all σ, $\textsc{Algorithm}(\sigma) \geq \alpha \text{OPT}(\sigma)$. Note that $\alpha \ll 1$ and the smaller it is, the better the algorithm.

Where γ is some constant independent of σ.

For this chapter, "algorithm" means ***online algorithm***.

Exercise 19.3. Prove that the First Fit algorithm for bin packing has competitive ratio ≤ 2.

Johnson et al. [JDU+74] showed that the competitive ratio for first fit bin packing is $\frac{17}{10}$. They also look at other approaches.

19.2 Caching Problem

There are n pages of RAM and k pages of cache. When a page from RAM is requested it is put into the cache; however, some page from the cache has to be kicked out of cache and put into RAM. We want the pages in the cache to be ones that are going to be requested either soon or a lot.

We will assume that the first k requests have already been made and are in the cache.

> CACHE
> *Instance:* A sequence of requests for pages from RAM. The cache is size k and initially there are k pages in the cache.
> *Question:* Every time a request is made we first check if the page is already in the cache. If so then the cost is 0 and we do not need to do anything (though we may keep track of the fact that the request was made). If the page is not in cache then we bring it into cache and put some page that was in cache into memory. The cost is 1. The goal is to minimize the cost, so minimize the number of times that a page is requested that is not in cache.

There are several algorithms that one may consider the caching problem such as:

1. **FIFO:** First In, First Out. Remove from cache the element that has been there the longest.
2. **LIFO:** Last In First Out. Remove from cache the element that was put there most recently.
3. **LRU:** Least Recently Used. Remove from cache the element that has been requested the longest ago.
4. **LFFO:** Least Frequency First Out. Remove from cache the element that has had the least requests.

Definition 19.4. A ***fault*** is when a request is made that is not in the cache.

Theorem 19.5. *LRU is k-competitive.*

Proof. Let $S = \sigma_1, \ldots, \sigma_N$ be a sequence of requests. We need to show that for every k faults that LRU makes, the optimal algorithm would make at least 1 fault.

We assume that the first k different page requests are put into the cache by both the optimal algorithm and LRU. We only consider the page requests after the first k.

We partition S (except for the first k distinct requests) as $\Sigma_1, \ldots, \Sigma_L$ where for all $i \geq 1$, in Σ_i LRU makes exactly k faults. We show that, for all $i \geq 1$, each Σ_i can be mapped to a fault of the optimal algorithm.

Let $1 \leq i \leq L$. Let σ be the last request made in Σ_{i-1}. There are three cases.

1. **Case 1:** The k page requests in Σ_i are distinct from each other and from σ. Hence σ together with the requests in Σ_i have $k+1$ distinct page requests. Hence the optimal algorithm will have a fault.

2. **Case 2:** There is some page τ that LRU faults on twice in Σ_i. Say $\sigma_i = \tau$ and $\sigma_j = \tau$. Then when the σ_i request is made τ is brought into cache; however, by the time the σ_j request is made, τ has been evicted. When it was evicted it was the Least Recently Requested page. Hence between σ_i and τ being evicted, there must have been $k+1$ distinct requests. Hence the optimal algorithm will have a fault.

3. **Case 3:** All of the faults in Σ_i are distinct from each other but one of them is σ. Hence σ must have been evicted. From here the reasoning is as in Case 2.

□

Now that we have an online k-competitive algorithm, the question arises, is there a better one? Sadly no.

Theorem 19.6. *There is no deterministic algorithm with competitive ratio better than k for the caching problem.*

Proof. Let Algorithm be a deterministic online algorithm for the cache problem. We use an adversary argument. That is, we feed the algorithm a sequence of page requests that will force it to have competitive ratio $\sim k$. We denote what we feed it $\sigma_1, \ldots, \sigma_N$. We will think of N as large, much larger than k. We will assume k divides N.

1. Initially request $k+1$ distinct pages. This will cause a fault. Let τ_1 be the evicted page.

2. Ask for τ_1. Let τ_2 be the evicted page.

3. Ask for τ_2. Let τ_3 be the evicted page (it could be that $\tau_1 = \tau_3$).

4. Ask for τ_3. Let τ_4 be the evicted page.

5. Do this N times.

If there are N requests then there will be N faults (after the first k which we do not count).

The optimal would have been to evict the page that will be requested furthest in the future. Realize that this means that after a fault there will not be another fault for at least k requests. Hence OPT causes $\le \frac{N}{k}$ requests.

Since Algorithm causes $\sim N-k$ faults and OPT causes $\frac{N}{k}$ faults, the competitive ratio is bounded by:

$$\frac{N-k}{N/k} \sim k.$$

□

The above technique to prove lower bounds on deterministic algorithms is typical: for every deterministic algorithm construct a sequence of requests that cause many faults for that algorithm (which cannot see the future) but fairly few faults for the optimal algorithm (which can see the future).

But what about randomized algorithms? Fiat et al. [FKL+91] showed the following

Definition 19.7. For all k, H_k is $\sum_{i=1}^{k} \frac{1}{i}$ which is $\ln k + \Theta(1)$.

Theorem 19.8. *Let k be the size of the cache.*

1. *There is a randomized paging algorithm with competitive ratio $2H_k = 2\ln k + O(1)$.*

2. *If* Algorithm *is any randomized paging algorithm then the competitive ratio is $\geq H_k = \ln k - O(1)$.*

Proof.
1) We present the ***Marking Algorithm***

1. Initially there are k pages in cache. They are not marked.

2. Whenever a request is made, whether or not it is in cache, it is marked.

3. If a request is made for a page not in cache then a page is chosen uniformly at random from the unmarked pages to be evicted.

4. When all k pages in cache are marked, all marks except the most recent one are removed.

2) We show that any randomized paging algorithm has competitive ratio $\geq H_k$.

□

19.3 Yao's Lemma

Proving a lower bound on the expected runtime of an randomized algorithms seems hard! An adversary argument won't work since that just gives one input that the algorithm is bad at.

Yao's lemma [Yao77] allows us to obtain lower bounds on the expected runtime of a randomized algorithm by looking at lower bounds on deterministic algorithms.

Assume you have some problem (e.g., Cache). There is an associated cost that we are trying to minimize. Picture the following two scenarios:

1. Let $\mathcal{A}$ be a set of deterministic algorithms. Let q be a distribution over a set of inputs.

   Fix an $a \in \mathcal{A}$. Use q to pick the input x. We denote the expected cost $E[c(a,x)]$. We pick the a that minimizes this. Hence we have the quantity $\min_{a\in\mathcal{A}} E[c(a,x)]$.

2. Let $\mathcal{X}$ be a set of inputs. Let p be a distribution over a set of algorithms.

   Fix an $x \in \mathcal{X}$. Use p to pick the algorithm a. We denote the expected cost $E[c(a,x)]$. We pick the x that maximizes this. Hence we have the quantity $\max_{x\in X} E[c(a,x)]$.

The following is Yao's Lemma:

Lemma 19.9. $\max_{x\in X} E[c(a,x)] \geq \min_{a\in\mathcal{A}} E[c(a,x)]$.

To get the intuition we recap what each side of the equation is.

- $\max_{x\in X} E[c(a,x)]$. We are picking the input x to make the expected cost (picking the algorithm via distribution p) as high as possible.

- $\min_{a\in\mathcal{A}} E[c(a,x)]$. We are picking the algorithm a to make the expected cost (picking the input via distribution q) as low as possible.

19.4 Online Matching

ONLINE MATCHING
Instance: A bipartite graph $((U,V),E)$ is going to be the final input. Initially we have the set V in advance, which are called ***offline vertices***. The vertices in U arrive one by one, which are called ***online vertices***. At the time we get u_i, we get all of its neighbors as well.
Question: When we receive an online vertex u_i, we need to match u_i to one of the offline vertices that has not already been matched, if there is one. This decision is irrevocable. The goal is to maximize the number of matches.

The ***greedy algorithm*** for this problem will, given a new u_i, match it to the least indexed, still available, $v \in V$ such that $(u_i, v) \in E$. (This algorithm does not look particularly greedy. We discuss a truly greedy algorithm for the weighted case in the exercises.)

Exercise 19.10. Let $G = ((U,V),E)$ be a bipartite graph. We will assume here and throughout this paper that $|U| = |V|$.

1. Show that the greedy algorithm for online matching always returns a maximal matching. Note that this is maximal, meaning that no edge can be added.

2. Show that for any bipartite graph a maximal matching is $\geq \frac{1}{2}|V|$.

3. For all n give an example of a graph where $|U| = |V| = n$ and an arrival order for U where (1) the graph has a matching of size n, but (2) the greedy algorithm produces a matching of size $\frac{n}{2}$.
   Hint: Use the bipartite graph where (a) for $1 \leq i \leq \frac{n}{2}$ there are edges (u_i, v_i) and $(u_i, v_{i+(n/2)})$, and (b) for $\frac{n}{2}+1 \leq i \leq n$ there is an edge $(u_i, v_{i-(n/2)})$. See Figure 19.1.

4. Show that the greedy algorithm has competitive ratio $\frac{1}{2}$. (this follows from the Parts 2 and 3).

5. Show that *any* deterministic algorithm will have competitive ratio $\leq \frac{1}{2}$.
   Hint: Use an adversary argument.

We now consider a promising randomized algorithm, just to show that it does not do well after all.

Theorem 19.11. *Consider the randomized algorithm which picks a match for u_i at random from the vertices that are available.*

1. *If you run this algorithm on the graph from Exercise 19.10.3 the expected competitive ratio is $\frac{3}{4}$.*

2. *Let $G = (V,U,E)$ be the following bipartite graph. $V = \{v_1, \ldots, v_n\}$. $U = \{u_1, \ldots, u_n\}$. For all i there is an edge (u_i, v_i). For all $1 \leq i \leq \frac{n}{2}$ for all $\frac{n}{2}+1 \leq j \leq n$ there is an edge (u_i, v_j). If you run the algorithm on this graph then the expected competitive ratio is 2.*

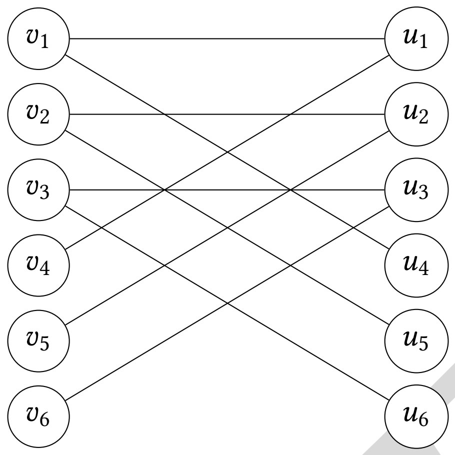


Figure 19.1: Greedy Online Matching has Competitive Ratio at Most $\frac{1}{2}$

Proof.
1) For $1 \le i \le \frac{n}{2}$ when u_i arrives the probability that it will match to $v_{i+(n/2)}$ is $\frac{1}{2}$. Hence the expected number of $v_{n/2}, \ldots, v_n$ that are available for $u_{n/2}, \ldots, u_n$ is $\frac{n}{4}$. So the expected number of matches is $\frac{n}{2} + \frac{n}{4} = \frac{3n}{4}$. The optimal is n. hence the competitive ratio is $\frac{3}{4}$.

2) The vertices $u_1, u_2, \ldots, u_n$ arrive in that order. For $1 \le i \le \frac{n}{2}$ if u_i gets matched to v_i thats good (in terms of maximizing matches) since it does not take take a vertex that is the only neighbor of some u_i with $i \ge \frac{n}{2} + 1$. Hence we call such a vertex ***good***.

$$\Pr[u_1 \text{ is good}) = \frac{1}{n/2+1}$$
$$Pr[u_2 \text{ is good}) = Pr[u_1 \text{ is good})\frac{1}{n/2+1} + Pr[u_1 \text{ is bad})\frac{1}{n/2} \le \frac{1}{n/2}$$

We leave it as an exercise to show that

$$Pr[u_i \text{ is good}) = 1/(n/2 - i + 2)$$

Thus we have:

$$E(\textsc{Algorithm}) \le \sum \frac{1}{n/2 - i + 2} + n/2 = O(\log n) + n/2.$$

Note that there is a matching of size n. Hence the competitive ratio is

$$\frac{O(\log n) + (n/2)}{n} \sim n.$$

□

To recap: any deterministic greedy algorithm, and the simple randomized algorithm, have competitive ratio 2. Is there a randomized algorithm with compete ratio < 2. Yes! Karp et al. [KVV90] (see also Birnbaum & Mathieu [BM08], and Plotkin [Plo13]) showed the following.

Theorem 19.12. *Let e be Euler's number, roughly 2.7185.*

1. *There is a randomized algorithm for online matching with competitive ratio* $\frac{e-1}{e} \sim 0.632$.
2. *All randomized algorithm for online matching have competitive ratio* $\leq \frac{e-1}{e} \sim 0.632$.

Proof sketch: We give the algorithm and prove that no randomized algorithm does better. We will not show that the competitive ratio is $\frac{e-1}{e}$.

Here is the algorithm, which we call RPerm.

1. Before the algorithm begins pick a permutation of V at random. This gives a ranking of V.
2. When $u \in U$ is seen, match it to the least element of V (according to the ranking) that is unmatched.

In preparation for applying Lemma 19.9 we present a distribution of inputs.

- For all i and $i \leq j$, we have the edge (u_i, v_j).
- For every permutation π, $I(\pi)$ relabels the vertices in V by π, i.e. we have edges (u_i, v_{π_i}).
- The input is $I(\pi)$, where π is chosen uniformly at random.

Claim 1: For any algorithm A we have, $E[A(P)] \leq E[\text{RPerm}(I)]$.
Proof of Claim 1

Consider an arbitrary iteration i. At (the beginning of) step i, let the set of eligible vertices be $Q(i) = \{v_{\pi_i} | j \geq i\}$.

a) Suppose that, A (or RPerm) have k unmatched eligible vertices. Any two subsets of size k from $Q(i)$ are the set of unmatched eligible vertices, with the same probability.

Let $P(i,k)$ be the probability that at iteration i, the number of unmatched eligible vertices is k. In fact, we have $Pr_{A(P)}(i,k) = Pr_{\text{RPerm}(I)}(i,k)$.

This shows that the expected number of steps that makes the number of unmatched eligible vertices 0 are the same in A and RPerm.
End of Proof of Claim 1
Claim 2: $E[\text{RPerm}(I)] \leq n(1 - 1/e)$
Proof of Claim 2

Consider the algorithm RPerm. For each iteration i, define the following two random variables:

- $x(i) = n - i + 1 = |Q(i)|$
- $y(i)$ =number of unmatched eligible vertices.

Consider that we have:

- $\Delta x = -1$.
- $\Delta y = -2$ if v_{π_i} is unmatched and u_i will not me matched to it.
- $\Delta y = -1$ otherwise.

By Claim 1 we have $Pr[v_{\pi_i} \text{ is unmatched}] = \frac{y(i)}{|Q(i)|}$. Therefore

$$Pr[y(i+1) - y(i) = -2] = \frac{y(i)}{x(i)} \frac{y(i)-1}{y(i)}$$

Thus, we have

$$E[\Delta y] = -1 - \frac{y(i)-1}{x(i)}$$

which gives us

$$\frac{E[\Delta y]}{E[\Delta x]} = 1 + \frac{y(i)-1}{x(i)}$$

When n goes to infinity, this can be approximated by the solution of the following differential equation:

$$\frac{dy}{dx} = 1 + \frac{y-1}{x}$$

which gives us

$$y(n+1) = \frac{n}{e} - o(n).$$

This completes the proof of the Claim.
End of Proof of Claim 2

From Lemma 19.9 and Claims 1,2 we have

▮

19.5 Online Set Cover

We first recall the usual offline Set Cover Problem.

Set Cover
Instance: F the entire domain which we can think of as a set $\{1, \dots, n\}$, and E a set of subsets of F.
Question: What is the size of the smallest collection of sets from E that contain (cover) all of the elements of F.

We will be interested in the online version of this problem.

OLS
Instance: F the entire domain which we can think of as a set $\{1, \dots, n\}$. It will be given ahead of time. E is a collection of subsets of F. It is also given ahead of time. Elements of F, together with a list of which sets in E cover them arrive one by one.
Question: When we receive an elements $x \in F$ and a set of subsets $E_1, \dots, E_k$ that each cover x, pick one. Indeed, this decision is irrevocable. The goal is to minimize the number of sets in E that cover all of the elements received. (Note that we may well not receive all of the elements.)

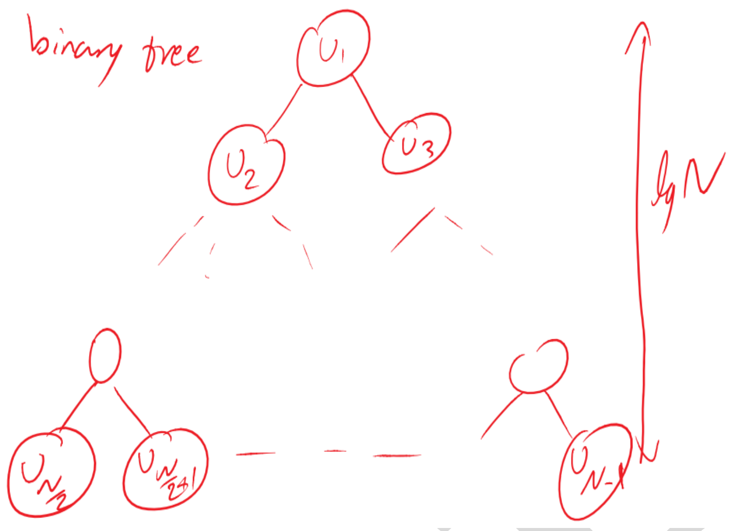


Figure 19.2: Binary Tree

This online problem is very different from both Cache and Online Matching. For those two problems the offline version was in P. For Set Cover the offline version is NP-hard.

To prove a lower bound on OLS we (surprisingly) need to assume P $\neq$ NP. Recall that Theorem 9.30.4 states that if P $\neq$ NP then there is no $(1 - o(1)) \lg n$-approximation for (offline) set cover. (This means that there is no algorithm that will return a number of sets that is $\leq (1 - o(1)) \ln n \times \text{OPT}$ where OPT is the optimal (minimum) number of sets needed.) From this result it is easy to obtain the following:

Theorem 19.13. *If P $\neq$ NP then the competitive ratio for OLS is* $\geq (1 - o(1)) \ln n$.

We will prove a larger lower bound on the competitive ratio.

Theorem 19.14. *If P $\neq$ NP then the competitive ratio for OLS is* $\geq \Omega(\log^2 n)$.

Proof. Let (E, F) be an offline hard instance with an optimal solution of size k, but any algorithm that runs in polynomial time, can not compute a solution better than k' (we know $k' \in \Omega(\log(n))k$). We will construct an online instance (U', F', E') with an optimal solution of size k s.t. an online algorithm with solution of size better than $k'O(\log(n))$ requires solving (E, F) with a cost better than k'.

This implies an $\Omega(\log^2(n))$-hardness for online algorithms that run in polynomial time.

We will shortly see that for a size N, we have $|E| = m$, $F = n$, $|U'| = (N' - 1)m$, $|F'| = \frac{N}{2}n$, $|E'| = m.\log(N)$.

We depict these elements in the binary tree showed in Figure 19.2. For every copy, at $i \in [1 \ldots N-1]$, we have a path from root to that copy.

AUGUSTE: ADD TO EXCEL. ITS PNG. WE DID IT OURSELVES- YOU WILL REDO.

Let the ordered vector of indices P_i, denote the indices of the path form root to i.

- $P_1 =< 1 >$
- $P_i =< P_{[i/2]}, i >$

Constructing the sets:

Substantively, every leaf $i \in [N/2, \ldots, N-1]$, has a copy of offline subsets F, that covers the same set of elements form $U_i, U_{i/2}, \ldots, U_1$.

For every $f \in F$ and $i \in [1 \ldots N-1]$, let $U_i(f)$ denote the elements of U_i that correspond to the copies of f.

For every $i \in [N/2, \ldots, N-1]$, let F_i denote a copy of F such that, for all $f \in F$, $f_i = \{j \in \{1, \ldots, \log N\} \mid U_{p_i(j)}(f)\}$.

For every leaf i, we construct an online sequence E'_i as follow:

$$E'_i = \langle U_{p_i(1)}, U_{p_i(2)}, \ldots, U_{p_i(\log(N))} \rangle.$$

By Yao's lemma, it is sufficient to show that if we pick a leaf i uniformly at random, then every online deterministic algorithm that runs in polynomial time uses $\Omega(\log(n)k')$ sets to cover the online requests E'_i.

Consider an arbitrary step $j \in [1, \ldots, \log(N)]$ in which we receive $U_{p_i(j)}$. Consider the subtree T rooted at $U_{p_i(j)}$. Only the sets that are in the leaves of T can be useful.

We may assume that the online algorithm has to choose at least k' sets among these sets to cover $U_{p_i(j)}$. Let k_1 denote the number of selected sets in the left subtree of T, while k_2 denote the number of those at the right subtree. Consider that $k_1 + k_2 \geq k'$ which gives us $max(k_1, k_2) \geq \frac{k'}{2}$.

With out loss of generality assume that $k_2 \geq k_1$. With probability $\frac{1}{2}$ the next U-node might go to the left subtree. Hence, all the sets that contribute to k_2 will be redundant. Therefore the online solution wastes at least $\frac{k'}{2}$ sets with probability $\frac{1}{2}$. Thus, we have:

$$E[\text{size of an online solution}] \geq \frac{\log(N)}{2} \frac{k'}{2}$$

as desired.

Note that OPT is still k. we can simply choose the optimal solution at the final leaf.

□

19.6 Further Results

1. In Section 19.4 we looked at online matching for bipartite graphs where the *vertices* arrive. We found that (a) deterministic algorithms always have competitive ratio $\leq \frac{1}{2}$, (b) there is a randomized algorithms with competitive ratio $\frac{e-1}{e}$, and (c) $\frac{e-1}{e}$ is the best one can do.

   What about general graphs? What if edges arrive? Gamlash et al. [GKM+19] showed that (a) for vertex arrivals in general graphs there is a randomized algorithm with competitive ratio $\frac{1}{2} + \Omega(1)$, and (b) for edge arrivals randomization does not help.

2. Role-matchmaking is a problem where players of different skills levels arrive and must be assigned to a team as soon as they arrive. The goal is to have the teams be balanced so that no team dominates. This can get very complicated since different skills is not 1-dimensional. For example, in soccer a team may need a good Goalkeeper more than a great midfielder. This problem has immediate applications to many popular online video games where such as *League of Legends* and *Dota 2*. Alman & McKay [AM17] view this as a dynamic data

structures problem. The show (a) assuming the 3SUM-Conjecture conjecture, any data structure for this problem requires $n^{1-o(1)}$ time per insertion or $n^{2-o(1)}$ time per query, and (2) there is an approximation algorithm that takes $O(\log n)$ per operation.

Chapter 20

Lower Bounds on Streaming Algorithms

20.1 Overview

In Chapter 19 we looked at online algorithms. Now we look at streaming algorithms, which have many similarities with online algorithms. They both require decisions before seeing all data but streaming algorithms can defer actions with limited memory.

The main tool used to get lower bounds on streaming algorithms is communication complexity. We will introduce the field and state results we need.

20.2 Introduction to Streaming Algorithms

Informally, streaming algorithms take a large set of data and process it without seeing the entire stream, and with limited memory. The study of this field was initiated by a seminar paper of Alon et al. [AMS99], which won the Gödel prize.

Streaming algorithms are used to process very long data streams. Online social networks such as Facebook and Twitter produce such streams.

Definition 20.1. ***Streaming Algorithms*** are algorithms for processing data streams in which the input is presented as a sequence of items (often numbers or edges of a graph) and can be examined in only a few passes (typically just one) by using relatively little memory (much less than the input size), and also limited processing time per item. We often produce approximate answers based on a summary or "sketch" of the data stream in memory. A common technique for creating a "sketch" is sampling at random.

Example 20.2. The input will be all of the numbers in $\{1, \ldots, n\}$ appearing once except there is one number that will not appear. The output will be that one number. Space is $O(\log n)$ and there is only one pass through the data. Because of the space limitation you cannot keep all or even most of the stream in memory.

Here is the algorithm: as the numbers come in keep a running sum s of them. This sum will take only $O(\log n)$ space. Once you have the sum s compute $\frac{n(n+1)}{2} - s$. That's your number!

Exercise 20.3. Generalize Example 20.2 to a stream that is missing 2 numbers. Then k numbers. Keep track of the space needed as a function of k. Can you reduce space by allowing more passes through the stream?

The data stream of the input can be a sequence of items such as integers or edges of a graph. In many cases when the streaming data are items, a vector $\vec{a} = (a_1, \ldots, a_n)$ is initialized to the zero vector $\vec{o}$ and the input is a stream of updates in the form of $< i, c >$, in which a_i is incremented by an integer c, i.e. $a_i = a_i + c$. Note that c can be negative here. Below are four important streaming problems for items based on the vector $\vec{a}$.

1. Evaluate the k^{th} frequency moment: $F_k(\vec{a}) = \sum_{i=1}^{n} a_i^k$.
2. Find heavy hitters, which is to find all the elements i with frequency $a_i > T$.
3. Count the number of distinct elements.
4. Calculate entropy: $E(\vec{a}) = \sum_{i=1}^{n} \frac{a_i}{m} \log \frac{a_i}{m}$, where $m = \sum_{i=1}^{n} a_i$.

Note here in all streaming algorithms, we want to minimize space, and then update time, even through multiple passes. The accuracy of the algorithm is often defined as an ***(ε, δ)-approximation***. It means the algorithm achieves an error of less than ε with probability $1 - \delta$.

20.3 Streaming for Graph Algorithms

There are two main versions of graph streaming:

1. Insertion only. Edges are added to the graph over time.
2. Dynamic. Edges are added to the graph but can also be deleted. If (i, j) appears twice then the first time means to add it and the second time means to delete it.

Here we introduce three popular models of graph streaming algorithms:

1. ***Semi-streaming*** model sets the input to be the stream of edges. If an edge appears with a negative sign, it represents deletion. Briefly speaking, semi-streaming requires $\widetilde{O}(n)$ for n edges.
2. The ***Parameterized stream*** model is a combination of fixed parameter tractable algorithms and streaming algorithms. There is a parameterized problem (e.g., Vertex Cover with parameter k). You want a streaming algorithm for it where the space is a function of the parameter (e.g., $\widetilde{O}(k)$) or involves the parameter in some way (e.g.,) $k$polylog n).
3. ***Sliding window*** model is a more generalized one, in which the function of interest is computing over a fixed size window of the stream according to the time and the update. Note that when new items are added to the window, items from the end of window are deleted.

In fact, there are lots of interesting lower bounds for massive graph problems, especially in the semi-streaming model where edges E are streaming in as inputs (often in adversarial order, but sometimes in a random order) with the bounded storage space, which is $\widetilde{O}(n) = O(n \cdot \text{polylog } n)$, where $n = |V|$.

20.4 Semi-Streaming Algorithm for Maximum Matching

Maximum Matching
Instance: $G = (V, E)$, a graph.
Question: Find the largest set of disjoint edges in G. (Any set of disjoint edges is called ***a matching***.)

We will look at semi-streaming algorithms for Maximum Matching with regard to (a) number of passes, (b) space used, and (c) approximation factors.

We look at an easy streaming algorithm to establish a baseline.

Theorem 20.4. *There is a semi-streaming algorithm for* MaximalMat *where (a) as soon as an edge comes in you need to decide whether or not to put it in the matching, and you can't change your mind; (b) there is only one pass through the data; (c) the space used is $\widetilde{O}(n)$; (d) the algorithm has approximation factor $\frac{1}{2}$ i.e. $Alg(\text{worst case}) \geq 1/2 \cdot Opt$.*

Proof.

1. The vertices are stored. Each vertex requires $O(\log n)$ bits to store, so the total storage is $\widetilde{O}(n)$. Initially all vertices are unmarked and the set M is empty.

2. Whenever a new edge $e = (x, y)$ comes in, if both x and y are unmarked, add e to M and mark both end vertices.

Since each selected edge ruins at most two edges in the optimal solution, the approximation factor of this algorithm is 2, i.e. $Alg(\text{worst case}) = 1/2 \cdot Opt$. □

Better results are known. Farhadi [FHM+20] showed the following:

1. There is a semi-streaming algorithm for maximum matching in bipartite graphs that uses (a) 1 pass, (2) $\widetilde{O}(n)$ space, and (3) has approximation factor 0.6.

2. There is a semi-streaming algorithm for maximum matching in general graphs that uses (a) 1 pass, (2) $\widetilde{O}(n)$ space, and (3) has approximation factor 0.545.

Before going into some hardness proofs, let's get insight into communication complexity, which is used as the most common technique for computing lower bounds in streaming algorithms.

20.5 Communication Complexity

In this section we present just the communication complexity we need for our lower bounds. An excellent reference that includes proofs of all the theorems in this section, is the book by Kushilevitz & Nisan [KN97].

Assume Alice has $x \in \{0, 1\}^n$ and Bob has $y \in \{0, 1\}^n$. They want to know whether $x = y$. Alice *could* just send Bob the entire string x, and then Bob could send back 0 if $x \neq y$ and 1 if $x = y$. The entire protocol takes $n + 1$ bits. Can they accomplish this with less bits? What if we allow randomization and a small probability of error? This is the beginning of *communication complexity.*

Exercise 20.5.

1. Show that any deterministic protocol for determining whether $x = y$ requires $\geq n$ bits.

2. Show that there is a randomized protocol for determining whether $x = y$ that uses private coins, $O(\log n)$ bits, and has probability of error $\leq \frac{1}{n}$.
   Hint: View the sequence $a_{n-1} \cdots a_0$ as the polynomial $a_{n-1}x^{n-1} + \cdots + a_0$ over mod p for a suitably chosen prime p.

Definition 20.6. Let $f : X \times Y \longrightarrow Z$ be a function. The 2-party communication model consists of two players, Alice and Bob. Alice is given an input $x \in X$ and Bob is given an input $y \in Y$. Their goal is to compute $f(x, y)$. Neither knows the other players input. We will be concerned with how many bits they need to communicate in order to compute f.

1. A ***protocol for f*** is just what you think it is: an algorithm for Alice and Bob to compute f. Formally it is a decision tree where what a player sends is based on what they have already seen and their own input.

2. The ***best protocol*** for f is the one with the smallest worst case for number of bits needed.

3. The ***communication complexity*** of f is the worst case of the best protocol.

4. There are many types of protocols. (1) Deterministic, (2) Randomized with a small probability of error, (3) ***One-way protocol*** means only one player sends information and the other one receives information, where both roles, sender and receiver, needed to be specified and only the receiver needs to be able to compute f.

Index, IndexSame, and Disj (short for disjointness) are three well-known problems in the area of communication complexity.

Index
Instance: Alice has a string $x \in \{0, 1\}^n$ and Bob has a natural number $i \in [n]$.
Question: Bob wants to know x_i. We use the one-way model so Alice sends Bob a string and just from that Bob needs to find x_i.

IndexSame
Instance: Alice has a string $x \in \{0, 1\}^n$ and Bob has a natural number $i \in [n - 1]$.
Question: Bob wants to know whether $x_i = x_{i+1}$. We use the one-way model so Alice sends Bob a string and just from that Bob needs to determine whether $x_i = x_{i+1}$.

Disj
Instance: Alice has a string $x \in \{0, 1\}^n$ and Bob has a string $y \in \{0, 1\}^n$.
Question: They both want to know whether the sets represented (by bit vectors) of x and y are disjoint. The communication is 2-way and they can have as many rounds as they want.

The first part of the next theorem is folklore. The second part was first proven by Kalyanasundaram & Schintger [KS92]. Razborov [Raz92] obtained an easier proof. We will not prove the theorem; however, we will make the easy parts of it exercises.

Definition 20.7. A ***Randomized Protocol*** is one that uses public coins and has probability of correctness $\geq \frac{2}{3}$.

Theorem 20.8.

1. *Index requires $\Omega(n)$ bits (in the 1-way communication model). This lower bound also holds for both deterministic and randomized protocols.*

2. *IndexSame requires $\Omega(n)$ bits (in the 1-way communication model). This lower bound also holds for both deterministic and randomized protocols.*

3. *Disj requires $\Omega(n)$ bits (even though Alice and Bob may use many rounds of communication). This lower bound holds for both deterministic and randomized protocols. It also holds when promised that $\sum x_i = \sum y_i = \lfloor n/4 \rfloor$.*

Exercise 20.9.

1. Prove that Index requires $\geq n$ bits in the 1-way deterministic model.

2. Prove that if IndexSame can be done with $o(n)$ bits then Index can be done with $o(n)$ bits. **Hint:** Given (x, i), an instance of Index, create $(x', 2i - 1)$ an instance of IndexSame so that $x_i = 1$ if and only if $x'_i = x'_{i+1}$. Do this by letting replacing $x_i = 0$ with 01 and $x_i = 1$ with 11.

3. Prove that IndexSame cannot be done with $o(n)$ bits. (This is just the contrapositive of Part 2. We list it here so we can refer to it.)

20.6 Lower Bounds on Graph Streaming Problems

In this section we reduce communication problems to several streaming graph problems. Since the communication problems have unconditional lower bounds (see Theorem 20.8) we obtain unconditional lower bounds on the space needed for the streaming graph problems.

We only use the lower bounds on the deterministic communication complexity. However, in all cases, there is a lower bound on the randomized communication complexity. Hence we really obtain lower bounds on the space needed for randomized streaming algorithms for graph problems.

20.6.1 Lower Bound Using Index: Max-Conn-Comp

Max-Conn-Comp(k), $k \geq 3$
Instance: A forest $G = (V, E)$.
Question: Is there a connected component of size $\geq k$. Note that k is not part of the input. We have actually defined an infinite set of problems.

Theorem 20.10. *Let $k \geq 3$. Any single-pass streaming graph algorithm that solves Max-Conn-Comp(k) problem on a forest needs $\Omega(n)$ space.*

Proof. Proved by reduction from INDEX to MAX-CONN-COMP(k).

Assume there is an $o(n)$ space streaming algorithm $\mathcal{A}$ for MAX-CONN-COMP(k). Note that it works for any arrival order of edges. We use $\mathcal{A}$ in the following protocol for INDEX. The protocol will take $o(n)$ space. which contradicts Theorem 20.8.1. Hence we will have that no such $\mathcal{A}$ exists.

1. Alice has $x_1 \cdots x_n \in \{0,1\}^n$ and Bob has $i \in [n]$. Our goal is that Alice gives Bob a string of length $o(n)$ and then Bob knows what x_i is.

2. Alice and Bob construct different parts of a graph. The vertices are $V_l \cup V_r \cup V_d$ where

   $V_l = \{l_1, l_2, \ldots, l_n\}$

   $V_r = \{r_1, r_2, \ldots, r_n\}$

   $V_d = \{d_1, d_2, \ldots, d_{k-2}\}$

   (a) Alice constructs the graph on vertices $V_l \cup V_r$ by letting

   $$E_A = \{(l_j, r_j) \mid x_j = 1\}.$$

   (b) Bob constructs the graph on vertices $\{r_i\} \cup V_d$ by letting

   $$E_B = \{(r_i, d_1)\} \cup \{(d_i, d_{i+1}) \mid 1 \le i \le k-3\}.$$

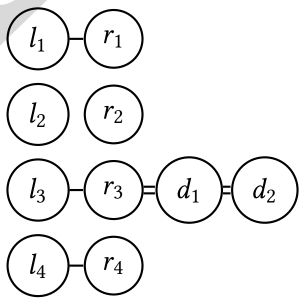
   (See Figure 20.1 for an example when $x = 1011$ and $k = 4$.)

3. Alice runs E_A through the streaming algorithm $\mathcal{A}$. Since it is an $o(n)$ space algorithm, when she is done there is $o(n)$ bits in memory. She tells Bob these bits. Note that this is just $o(n)$ bits.

4. Bob initializes the memory to those bits and then runs the streaming algorithm on E_B.

5. (Comment, not part of the algorithm.) If $x_i = 1$ then the path l_i-r_i-d_1-$\cdots$-d_{k-3} is a connected component of size k. If $x_i = 0$ then the longest connected component is of size $k-1$.

6. If when Bob finishes the streaming algorithm the answer is YES, then he knows that $x_i = 1$; if the answer is NO, then he knows that $x_i = 0$. The total 1-way communication is $o(n)$.



(−: Alice edge. =: Bob edge)

Figure 20.1: Instance of Max-Conn-Comp(k) Based on INDEX(1011, 3)

□

20.6.2 Lower Bound Using IndexSame: Is-Tree

Is-Tree
Instance: A graph G
Question: Is G a Tree?

Theorem 20.11. *Any single-pass streaming algorithm solving Is-Tree needs $\Omega(n)$ space.*

Proof. Essentially we use a similar proof strategy to the proof of Theorem 20.10: prove by reduction of IndexSame to Is-Tree. Assume there is an $o(n)$ space streaming algorithm $\mathcal{A}$ for Is-Tree. We use $\mathcal{A}$ in the following protocol for IndexSame. The protocol will take $o(n)$ space which contradicts Theorem 20.8.2. Hence we will have that no such $\mathcal{A}$ exists.

1. Alice has $x_1 \cdots x_n \in \{0,1\}^n$ and Bob has $i \in [n-1]$. Our goal is that Alice gives Bob a string of length $o(n)$ and then Bob knows if $x_i = x_{i+1}$ or not.

2. Alice and Bob construct different parts of a graph. The vertices are

$$V = \{zero, one, 1, 2, \ldots, n\}$$

   (a) Alice constructs the graph on vertices V

$$E_A = \{(zero, i) \mid x_i = 0\} \cup \{(one, i) \mid x_i = 1\}.$$

   (b) Bob constructs the graph with just one edge $E_b = \{(i, i+1\})$. (See Figure 20.2 for an example when $x = 01011$ and $k = 2$.)

3. Alice runs E_A through the streaming algorithm $\mathcal{A}$. Since it is an $o(n)$ space algorithm, when she is done there is $o(n)$ bits in memory. She tells Bob these bits. Note that this is just $o(n)$ bits.

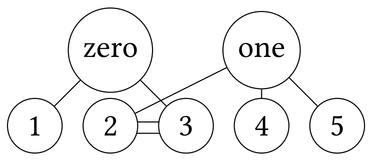
4. Bob initializes the memory to those bits and then runs the streaming algorithm on E_B.

5. (Comment, not part of the algorithm.) If $x_i = x_{i+1} = 0$ then $(zero, x_i)$, (x_i, x_{i+1}), and $(x_{i+1}, zero)$ are all edges so the graph is not a tree. Similar for $x_i = x_{i+1} = 1$. If $x_i \neq x_{i+1}$ then there are no cycles (exercise) so G is a tree.

6. If when Bob finishes the streaming algorithm the answer is YES (G is a tree) then he knows that $x_i \neq x_{i+1}$, if NO then he knows that $x_i = x_{i+1}$. The total 1-way communication is $o(n)$.

□

Exercise 20.12. Let $k \geq 3$. ***TREE-DIAM(k)*** is the following problem: Given $G = (V, E)$, a tree, determine whether the diameter of G is at least $k \geq 3$. Prove that any single-pass streaming algorithm for TREE-DIAM(k) needs $\Omega(n)$ memory.



(—: Alice edge. =: Bob edge)

Figure 20.2: Instance of IsTree(G) Based on INDEX-SAME(01011, 2)

20.6.3 Lower Bound Using Index: PerfMat

PerfMat
Instance: A graph G.
Question: Does G have a perfect matching (a disjoint set of edges such that every vertex is in one of the edges)?

Theorem 20.13. *Any single-pass streaming algorithm for* PerfMat *needs* $\Omega(m) = \Omega(n^2)$ *memory.*

Proof. We prove this by reduction from Index to PerfMat.

Here we assume string x in Index is a $n \times n$ array and Bob wants to compute x_{ij}. Then an instance of Index can be written as (x, i, j). Then we need space $\Omega(n^2)$ to solve this Index(x, i, j) problem.

Assume there is an $o(n^2)$ space streaming algorithm $\mathcal{A}$ for PerfMat. We use $\mathcal{A}$ in the following protocol for Index. The protocol will take $o(n)$ space which contradicts Theorem 20.8.1. Hence we will have that no such $\mathcal{A}$ exists.

1. Alice has $x_{ij} \in \{0, 1\}$ as $1 \leq i, j \leq n$. Bob has $(i, j) \in [n] \times [n]$. Our goal is that Alice gives Bob a string of length $o(n)$ and then Bob knows if x_{ij}.

2. Alice and Bob construct different parts of a graph. The vertices are

$$V = \{l_1, l_2, \ldots, l_{n-1}\} \cup \{r_1, r_2, \ldots, r_{n-1}\} \cup \{1, 2, \ldots, n\} \cup \{1', 2', \ldots, n'\}.$$

They will be arranged as seen in Figure 20.3

(a) Alice constructs the graph on vertices V

$$E_A = \{\{i, j'\} \mid x_{i,j} = 1\}.$$

(b) Bob constructs the graph on vertices V

$$\begin{aligned} E_B = \quad & \{\{l_k, k\} | 1 \leq k < i\} \cup \{\{l_k, k+1\} | i \leq k \leq n-1\} \cup \\ & \{\{r_k, k\} | 1 \leq k < j\} \cup \{\{r_k, k+1\} | j \leq k \leq n-1\} \end{aligned}$$

(See Figure 20.3 for an example where

$$x = \begin{matrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

and Bob's has $(3, 5)$.)

3. Alice runs E_A through the streaming algorithm $\mathcal{A}$. Since it is an $o(n^2)$ space algorithm, when she is done there is $o(n^2)$ bits in memory. She tells Bob these bits. Note that this is just $o(n^2)$ bits.

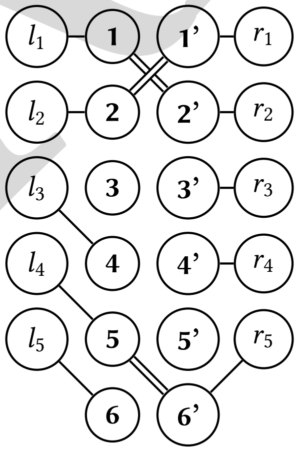
4. Bob initializes the memory to those bits and then runs the streaming algorithm on E_B.

5. (Comment, not part of the algorithm.) We leave the argument that

$$x_{ij} = 1 \textit{ if and only if } G \textit{ has a perfect matching}$$

to the reader.

6. If when Bob finishes the streaming algorithm the answer is YES (G has a matching) then he knows that $x_{ij} = 1$ if NO then he knows that $x_{ij} = 0$. The total 1-way communication is $o(n^2)$.



(−: Alice edge. =: Bob edge)

Figure 20.3: Instance of PerfMat(G) Based on INDEX(x, 3, 5)

□

20.6.4 Lower Bounds Using Index: Shortest-Path

Shortest Path
Instance: An unweighted graph G and two vertices v, w.
Question: What is the length of the shortest path from v to w?

We show that not only does any 1-pass streaming algorithm for this problem require $\Omega(n^2)$ space, even approximating the problem requires $\Omega(n^2)$ space.

Theorem 20.14. *Any single pass streaming algorithm that approximates Shortest Path with factor better than $\frac{5}{3}$ needs $\Omega(n^2)$ space. (So the algorithm produces a number that is $< \frac{5}{3} \times$ the length of the shortest path.)*

Proof. The proof here is similar to the same with the proof of PerfMat problem. We prove this by reduction from Index to Shortest-Path.

Here we assume string x in Index is a $n \times n$ array and Bob wants to compute x_{ij}. Then an instance of Index can be written as (x, i, j). Then we need space $\Omega(n^2)$ to solve this Index(x, i, j) problem.

Assume there is an $o(n^2)$ space streaming algorithm $\mathcal{A}$ for a better than $\frac{5}{3}$ approximation to shortest-path (note that it is strictly better than $\frac{5}{3}$ optimal).

We use $\mathcal{A}$ in the following protocol for Index. The protocol will take $o(n)$ space. which contradicts Theorem 20.8. Hence we will have that no such $\mathcal{A}$ exists.

1. Alice has $x_{ij} \in \{0, 1\}$ as $1 \le i, j \le n$. Bob has $(i, j) \in [n] \times [n]$. Our goal is that Alice gives Bob a string of length $o(n)$ and then Bob knows if x_{ij}.

2. Alice and Bob construct different parts of a graph G. v and w will be vertices of G and (G, v, w) will be in the input to the shortest-path problem. The vertices are

$$V = \{i, j, v, w\} \cup \{1, \ldots, n\} \cup \{1', \ldots, n'\}.$$

   They will be arranged as seen in Figure 20.4.

   (a) Alice constructs the graph on vertices V

$$E_A = \{\{i, j'\} \mid x_{i,j} = 1\}.$$

   (b) Bob constructs the graph on vertices V

$$E_B = \{(v, i), (j, w)\}.$$

   (See Figure 20.4 for an example where

$$x = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

and Bob's has $(2,5)$.)

3. Alice runs E_A through the streaming algorithm $\mathcal{A}$. Since it is an $o(n^2)$ space algorithm, when she is done there is $o(n^2)$ bits in memory. She tells Bob these bits. Note that this is just $o(n^2)$ bits.

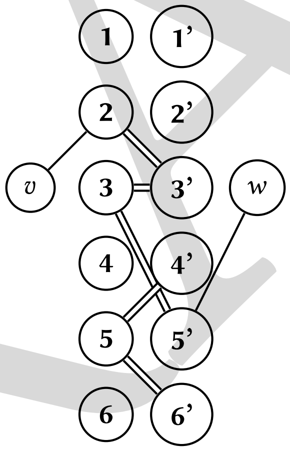
4. Bob initializes the memory to those bits and then runs the streaming algorithm on E_B.

5. (Comment, not part of the algorithm.)

   If $x_{ij} = 1$ then there is a path from v to w of length 3: (v,i), (i,j), (j,w). Hence a $< \frac{5}{3}$-approximation algorithm will return a number $< 3 \times \frac{5}{3} = 5$.

   If $x_{ij} = 0$ then the shortest path from v to w is at least 5: the path begins with (v,i). The next edge must be of the form (i,j'). Since $j' \neq j$, j' is not adjacent to w. So the next edge must be of the form (j',k). From node k the shortest distance possible to w is 2. Hence the shortest path is ≥ 5. Therefore a $< \frac{5}{3}$-approximation algorithm will return a number ≥ 5.

6. If when Bob finishes the streaming algorithm the answer is α. If $\alpha < 5$ then Bob knows $x_{ij} = 1$. If $\alpha \geq 5$ then Bob knows $x_{ij} = 0$. The total 1-way communication is $o(n^2)$.



(–: Alice edge. =: Bob edge)

Figure 20.4: Instance of Shortest-Path(v,w) Based on Index$(x,3,4)$

□

For the next exercise we will consider the following streaming problem.

Tree Diameter
Instance: A number $k \geq 3$ and a tree T. Note that the number is given first and can be easily stored, while the tree will be streamed.
Question: Is the diameter of T at least k?

Exercise 20.15. Let $k \geq 3$. Prove that any single-pass streaming algorithm for TD requires at least $\Omega(n)$ space.

20.6.5 Frequency Moments

FREQUENCY MOMENTS
Instance: A data stream of numbers from $[m]$: $y_1, y_2, \ldots, y_n$
Question: There are several.
For $k \in [m]$ the frequency of $k \in [m]$ is $x_k = |\{j \mid y_j = k\}|$.
The ***frequency vector*** is the vector $x = (x_1, x_2, \ldots, x_m)$.
Let $p \in \mathbb{N} \cup \{\infty\}$. The ***$p^{th}$ frequency moment*** of the input stream is defined as follows:

$$F_p = \begin{cases} \sum_{i=1}^{m} x_i^p & \text{if } p \in \mathbb{N} \\ \max_i x_i & \text{if } p = \infty \end{cases}$$

Note that F_0 is the number of distinct elements of the input. F_1 is the number of elements (with repetition).

Indyk & Woodruff [IW05] proved (among other things) the following.

Theorem 20.16. *Let $p > 2$.*

1. *There exists a randomized streaming algorithm which $1+\varepsilon$-approximates F_p in space $O(m^{1-(2/p)})$.*

2. *Any randomized streaming algorithm that $1+\varepsilon$-approximates F_p requires $\Omega(m^{1-\frac{2}{p}})$ space.*

20.7 Further Results

20.7.1 Graph Problems

For all of the problems listed the model is streaming with edge arrivals and n is the number of vertices.

1. For the MAXIMALMAT Esfandiari et al. [EHL+18] gives an algorithm that, with high probability, approximates the size of a maximum matching within a constant factor using $\tilde{O}(n^{2/3})$ space.

2. For the WEIGHTED MAXIMAL MATCHING PROBLEM Crouch & Stubbs [CS14] gives a $(4+\varepsilon)$ approximation algorithm which applies in semistreaming, sliding window, and MapReduce models. Chen et al. [CHK+21] studied this problem in the 1-pass model.

3. The PARAMETERIZED VERTEX COVER WITH PARAMETER k was studied by Chitnis et al. [CCHM15]. They proved a tight lower bound on the space of $\Omega(k^2)$ for randomized streaming algorithm.

4. MINIMUM SPANNING TREE ESTIMATION: Given a weighted undirected graph and ε, find a spanning tree that has weight $\leq (1+\varepsilon)$OPT where OPT is the weight of the minimal spanning tree. Assadi & N [AN21] proved that any algorithm that use $n^{o(1)}$ space requires $\Omega(1/\varepsilon)$ passes. The result still holds if the weights are constant.

5. ε-Connectivity: If at least $\varepsilon \cdot n$ edges need to be inserted into G to make it connected, G is said to be ***ε*-far from being connected**. Assadi & N [AN21] proved that any algorithm that use $n^{o(1)}$ space requires $\Omega(1/\varepsilon)$ passes.

6. Cycle-freeness: If at least $\varepsilon \cdot n$ edges need to be deleted from G to remove all its cycles, then G is said to be ***ε*-far from being cycle-free**. The problem is to determine whether a graph is cycle-free or ε-far from being cycle-free. Assadi & N [AN21] proved that any algorithm that use $n^{o(1)}$ space requires $\Omega(1/\varepsilon)$ passes.

7. Cycle-freeness: If at least $\varepsilon \cdot n$ edges need to be deleted from G to remove all its cycles, then G is said to be ***ε*-far from being cycle-free**. The problem is to determine whether a graph is cycle-free or ε-far from being cycle-free. Assadi & N [AN21] proved that any algorithm that use $n^{o(1)}$ space requires $\Omega(1/\varepsilon)$ passes.

8. The Gap Cycle Counting Problem: Let k be small. A graph G is streamed which is either a disjoint union of $\frac{n}{k}$ k-cycles or a disjoint union of $\frac{n}{2k}$ $2k$-cycles. Determine which is the case. Assadi [Ass21] showed that any p-pass streaming algorithm requires $n^{1-1/k^{\Omega 1/p}}$ space.

9. Assadi et al. [AR20] show that two-pass graph streaming algorithm for the s-t reachability problem for directed graphs requires space $n^{2-o(1)}$.

10. Goel et al. [GKK12] consider the maximum matching problem. They show that any one-pass algorithm cannot achieve better than 2/3 approximation. There have been improvements to the bound since this work and most recently, [Kap21] showed a $\frac{1}{1+ln2}$ bound.

11. Assadi [Ass21] consider approximating the maximum matching problem for two-pass algorithms and show that any such algorithm has approximation ratio at least $1-\Omega(\frac{\log RS(n)}{\log n})$ where $RS(n)$ denotes maximum number of disjoint induced matchings of size $\theta(n)$.

20.7.2 Non-Graph Problems

1. The Longest Increasing Subsequence: Given an ordered sequence of numbers $\vec{x} = (x_1, ..., x_n)$, find an increasing subsequence that is of maximal length. This is a streaming problem if, as the x_i's arrive, you decide if they will be in the increasing subsequence or not. Saks & Seshadhri [SS13] showed that, for all $\delta > 0$, a deterministic, single-pass streaming algorithm for additively approximating this problem to within an additive δn requires $O(\log^2 n/\delta)$ space. They also considered the Longest Common Subsequence problem (Given $\vec{x}$ and $\vec{y}$ find a maximal sequence that is a subsequence of both strings.) and gave an analogous result for that one as well.

2. Maximum Coverage: Given n, k and a set of m sets $S_i \subseteq \{1, \ldots, n\}$, find the k subsets that maximize the size of their union. There is a straightforward greedy $(1-e^{-1})$-approximation algorithm that runs in polynomial time. McGregor & Tu [MV19] give two single-pass streaming algorithms and one multi-pass streaming algorithm for approximations to this problem. For the multi-pass case they also have a lower bound.

(a) They have a single-pass streaming algorithm that for a $(1 - e^{-1} - \varepsilon)$-approximation that takes $\tilde{O}(\varepsilon^{-2}m)$ space.

(b) They have a single-pass streaming algorithm that for a $(1 - \varepsilon)$-approximation that takes $\tilde{O}(\varepsilon^{-2}m \min(k, \varepsilon^{-1}))$ space.

(c) They have an algorithm that for a $(1-e^{-1}-\varepsilon)$-approximation that takes $O(\varepsilon^{-1)}$ passes and $\tilde{O}(\varepsilon^{-2}k)$ space. They show that any $O(1)$ pass streaming algorithm for an $(1 - (1 - (1/k)^k) \sim 1 - \frac{1}{e}$ requires $\Omega(m)$ space.

3. Basic Counting: Given a stream of bits, maintain a count of the number of 1's in the last N elements seen from the stream. Datar et al. [DGIM02] showed this problem requires $\Omega(\varepsilon^{-1} \log^2 N)$ space for any randomized algorithms.

4. Sum: Given a stream of integers in $\{1, \ldots, R\}$, maintain the sum of the last N integers. Data et al. [DGIM02] showed that any streaming algorithm for this problem requires space $\Omega(\varepsilon^{-1}(\log^N + \log R \log N))$. This and the previous problem relate to computing the L_P norm with an underlying vector that has a single dimension.

5. Sorting by reversal on signed permutations: Given a data stream of a permutation S on $\{1, \ldots, n\}$, a reversal $r(i, j)$ will transfer $x = (x_1, \ldots, x_n)$ to $(x_1, \ldots, x_{i-1}, -x_j, \ldots, -x_i, x_{j+1}, \ldots x_n)$. Find the minimum number of reversals needed to sort S. Verbin & Yu [VY11] showed that this problem requires space $\Omega((n/8)^{1-1/t})$ for approximation factor $1 + 1/(4t - 2)$.

6. The approximate null vector problem: given $x_1, \ldots x_{d-1}$ vectors in $\mathbb{R}^d$ output a vector that is approximately orthogonal to all of them. Dagan et al. [DKS19] show that any one-pass streaming algorithm for this problem requires $\Omega(d^2)$ space.

7. Clarkson & Woodruff [CW09] consider a variety of Numerical Linear Algebra problems in the streaming model. They provide upper and lower bounds on the space complexity of one-pass algorithms. In what follows, A is an $n \times d$ matrix, B is an $n \times d'$ matrix and $c = d + d'$ and the input is assumed to be integers of $O(\log(nc))$ bits or $O(\log(nd))$ bits.

(a) For outputting a matrix C such that $\|A^T B - C\| \leq \varepsilon \|A\| \cdot \|B\|$, they show that $\Theta(c\varepsilon^{-2} \log(nc))$ space is needed.

(b) For $d' = 1$, i.e, when B is a vector b, finding an x such that $\|Ax - b\| \leq (1 + \varepsilon) \min_{x' \in \mathbb{R}^d} \|Ax' - b\|$ requires $\Theta(d^2 \varepsilon^{-1} \log(nd))$ space.

Chapter 21

Parallel Algorithms: The MPC and AMPC Models

21.1 Introduction

Throughout this book we have dealt with *sequential* computations. In this chapter we look at *parallel* computations.

There are many models of parallelism. One of the first ones was the PRAM (Parallel Random Access Machine) which allowed the processors to have access to a shared memory. This feature (or bug) leads to the need for subcategories of PRAM depending on if concurrent reads or concurrent writes are allowed. There is a vast literature on both algorithms and lower bounds for PRAMs. For algorithms we recommend Karp's survey [Kar88] and the papers it references. For lower bounds we recommend the papers of Cook & Dwork [CDR86] and Li & Yesha [LY89] and the papers they reference.

We study a more recent model called ***Massively Parallel Computation***. We will define it, give examples of problems it solves well, and then discuss lower bounds on it. We will note some relations to the PRAM when they come up. When we refer to PRAMs we will mean those that allow concurrent reads and writes. We omit details about how they resolve contentions.

21.2 The Massively Parallel Computing Model (MPC)

Beame et al. [BKS17] invented the following model.

Definition 21.1. The ***Massively Parallel Computation model (MPC)*** consists of the following:

- The input data size N. (For a graph $G = (V, E)$ this is $\max\{|V|, |E|\}$ which is usually $|E|$.)
- The number of machines M available for computation. The machines will communicate with each other in rounds (we elaborate on this later).
- The memory size, s words, each machine can hold.

A reasonable assumption here is that a single word stores a constant number of bits. Thus in practice, each machine is capable of storing $\Theta(s)$ number of bits. Moreover, in the literature, it is most often assumed that

$$M \cdot s = \Theta(N). \tag{21.1}$$

This is the most interesting scenario since the number of available resources for the algorithm ($M \cdot s$) is asymptotically equal to the size of the input data.

Input and output work as follows.

1. The input data is split across the M machines *arbitrarily*. If the input is a graph then initially each edge is given to some machine. Note that a machine may well have many edges.

2. There will be some machines designated as ***output machines***. At the end of the computation the answer will be stored there in a distributive manner. We give an example. If the problem is connected components then we want to assign to every vertex v a number n_v such that two vertices are in the same component if and only if they are assigned the same number. In addition to the input machines there will be n output machines, one for each vertex v, and at the end machine v has n_v.

Computation is performed in synchronous rounds. In each round, every machine performs some computation on the data that resides locally, then sends/receives messages to any other machine. Since the local memory of each machine is bounded by s words, we assume that a single machine can send and receive at most s words per round; however, each of these words can be addressed to or received from different machines. The two main parameters that are investigated in this model are:

- The number of communication rounds it takes for an algorithm to solve a problem, often called *running time*. The local computations are ignored in the analysis of the running time MPC algorithms because communication is the bottleneck. In practice, these local computations frequently run in linear or near-linear time, however, there is no bound on their length formally.

- The size of local memory s. Problems are easier to solve with larger s. In the extreme when $s \gg N$, one can just put the entire input on a single machine and solve it locally. Typically, s is polynomially smaller than N, e.g. $s = N^{\varepsilon}$ for some constant $\varepsilon < 1$.

If one of the machines had most of the input then, since we allow machines unlimited power, the problem could likely be done rather quickly. This is not what we want to model. We want to model problems where the input is so large that no machine can have most of it. Hence we have the following definition.

Definition 21.2. An MPC algorithm is ***strongly sublinear*** if each machine gets space $s \ll N$. So the space is much less than the input size. For graph problems (on dense graphs) it will mean that there exists $\delta < 1$ such that $s \leq n^{1+\delta}$. We will often use the term ***strongly sublinear*** rather than specify the space precisely.

Note the following

Fact 21.3.

1. The MPC algorithm is non-uniform. For every N we have a different MPC algorithm for input length N. These algorithms will be very similar.

2. An MPC-computation can be viewed as a directed graph in layers. The first layer has the input machines. The second layer has the machines that the first layer communicates with. Put a directed edge between a machine in the first layer and a machine that it sends to. Keep doing this for layers $2, 3, \ldots, R+1$. In Section 21.5 we use this viewpoint.

3. Any PRAM algorithm working in time t can be simulated by an MPC algorithm in $O(t)$ rounds and with strongly sublinear memory per machine.

21.3 Some Algorithms in the MPC Model

21.3.1 Connected Components

Connected Components (ConnComp)
Instance: An undirected graph $G = (V, E)$.
Question: Which vertices are in the same connected component? A solution is a labeling of vertices $\ell(v)$ such that $\ell(u) = \ell(v)$ if and only if vertices u and v are in the same connected component.

The runtime of an algorithm for ConnComp will depend on the number of vertices n, the number of edges m, and a new parameter, the diameter of the graph D, which we define.

Definition 21.4. Let G be a graph. The ***diameter D of G*** is the length of the longest shortest path between two vertices. Formally:

$$D = \max_{u,v \in V} \text{The length of the shortest path from } u \text{ to } v.$$

Behnezhad et al. [BDE+19a] proved showed the following theorem.

Theorem 21.5. *For all ε the following holds. There is a randomized strongly sublinear MPC algorithm for ConnComp such that, on a graph $G = (V, E)$ ($|V| = n$, $|E| = m$, Diameter D):*

1. *The total space used is $O(m)$*

2. *The number of rounds is $O(\log D + \log\log_{m/n}(n))$.*

3. *The algorithm succeeds with high probability.*

4. *The algorithm does not need to know D.*

We will not provide a proof of Theorem 21.5. Instead, we will give a short overview of a slightly weaker result due to Andoni et al. [ASS+18].

Theorem 21.6. *There is a randomized strongly sublinear MPC algorithm for ConnComp such that, on a graph $G = (V, E)$ ($|V| = n$, $|E| = m$, Diameter D):*

1. *The total space used is $O(m)$*

2. *The algorithm takes $O(\log D \cdot \log\log_{m/n}(n))$ rounds.*

3. *The algorithm succeeds with high probability.*

4. *The algorithm does not need to know D.*

Behnezadi et al. [BDE⁺19a] write the following about the ideas which lead to Theorem 21.6:
Graph exponentiation.

> Consider a simple algorithm that connects every vertex to vertices within its 2-hop (i.e., vertices of distance 2) by adding edges. It is not hard to see that the distance between any two vertices shrinks by a factor of 2. By repeating this procedure, each connected component becomes a clique within $O(\log D)$ steps. The problem with this approach, however, is that the required memory of a single machine can be up to $\Omega(n^2)$, which for sparse graphs is much larger than $O(m)$.

Solution to ConnComp Built off the Graph Exponentiation Technique.

> Suppose that every vertex in the graph has degree at least $d \gg \log n$. Select each vertex as a leader independently with probability $\Theta(\frac{\log n}{d})$. Then contract every non-leader vertex to a leader in its 1-hop (which w.h.p. exists). This shrinks the number of vertices from n to $O(n/d)$. As a result, the amount of space available per remaining vertex increases to $\Omega(\frac{m}{n/d}) = \Omega(\frac{nd}{n/d}) = d^2$. At this point, a variant of the aforementioned graph exponentiation technique can be used to increase vertex degrees to d^2 (but not more), which implies that another application of leader contraction decreases the number of vertices by a factor of $\Omega(d^2)$. Since the available space per remaining vertex increases doubly exponentially, $O(\log\log n)$ phases of leader contraction suffice to increase it to n per remaining vertex. Moreover, each phase requires $O(\log D)$ iterations of graph exponentiation, thus the overall round complexity is $O(\log D \cdot \log\log n)$.

21.3.2 Maximal Independent Set

Maximal Independent Set (Maximal Independent Set)
Instance: A graph $G = (V, E)$.
Question: Return an independent set I such that no independent set is a proper superset of I.

Note the following:

- Maximal Independent Set is very different from the Maximum Ind Set. Maximal Independent Setis in P by a simple greedy algorithm. Maximum Ind Set is NP-complete.

- The greedy algorithm for Maximal Independent Set is inherently sequential. Hence it is not obvious that there is a fast parallel algorithm for Maximal Independent Set. However, there is.

Luby [Lub86] gave a framework for Maximal Independent Set algorithms for the PRAM.
Luby's algorithm for Maximal Independent Set:

1. Fix a random permutation $\pi : [n] \rightarrow [n]$ of vertices.
2. A vertex v adds itself to I if, for all neighbors u of v, $\pi(v) < \pi(u)$.
3. Remove selected vertices and their neighbors.
4. Repeat until reaching an empty graph.

The following theorem comes from a simple analysis of the above method. The details can be found in Luby's paper.

Theorem 21.7. *There exists an algorithm in the PRAM model which, given a graph G on n vertices and m edges, solves Maximal Independent Set in $O(\log n)$ depth and $O(m)$ work.*

Ghaffari & Uitto [GU19] showed that the local nature of the Maximal Independent Set problem can be efficiently exploited in the MPC model. Namely they proved the following.

Theorem 21.8. *Let $\varepsilon \in (0, 1)$. There is a randomized MPC algorithm for Maximal Independent Set with the following properties:*

1. *Each machine uses $O(n^{\varepsilon})$ memory.*
2. *The number of rounds is $O(\sqrt{\log n} \cdot \log\log n)$ (the constant on the O depends on ε).*
3. *The probability that an Maximal Independent Set is found is $\geq 1 - \frac{1}{10n}$*

Theorem 21.8 is the best-known result for general graphs. Ghaffari, Grunau, Jin [GGJ20] showed that the Maximal Independent Set problem, restricted to some specific class of graphs (i.e. trees or graphs with bounded arboricity), has a randomized strongly sublinear MPC algorithm working in $O(\log\log n)$ rounds. Ghaffari, Kuhn, Uitto [GKU19] showed a (conditional) lower bound on the Maximal Independent Set problem in the MPC model of $\Omega(\log\log n)$ rounds. This is the best known lower bound on Maximal Independent Set in the MPC model.

21.3.3 Fast Fourier Transform and Pattern Matching

Hajiaghayi et al. [HSSS21] obtained a constant-round MPC-algorithm for Fast Fourier Transform and used that to get a constant-round MPC-algorithm for pattern matching (with wildcards). (All results in this section are from that paper.)

Fast Fourier Transform (FFT)
Instance: Complex numbers $x_0, \ldots, x_{n-1}$. (These will have rational real and complex parts so they are finite length.)
Question: Return the n complex numbers $X_0, \ldots, X_{n-1}$ where

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = \{0, \ldots, N-1\}.$$

(We can either use an approximation or keep the roots of unity in symbolic form so that the length is manageable.)

Theorem 21.9. *Let $\varepsilon > 0$. There is a deterministic MPC algorithm for FFT with local memory $O(n^{\varepsilon})$, total memory $O(n polylog(n))$, and $O(\frac{1}{\varepsilon})$ rounds.*

PATTERN MATCHING AND VARIANTS

Instance: A text T and a pattern P. They are both over an alphabet Σ. The text is usually much longer than the pattern. We describe three types of patterns in the QUESTION part.

Question:

- $P \in \Sigma^*$. We want all occurrences of P in T.
- $P \in \{\Sigma \cup ?\}^*$. The pattern occurs if it matches all of the characters in Σ. We do not care what happens at the ? spot. For example, acb?a occurs in **acbaa**bbbbb**acbba** as shown.
- $P \in \{\Sigma \cup +\}^*$. The pattern occurs if it matches all of the characters in Σ. At the + spots a single character can appear many times. For example, acb?a occurs in

  acbaaaaaaaaa bbbbb **acbbbbbbbbbbbbbbbbbbba**

  as shown.
- $P \in \{\Sigma \cup *\}^*$. The pattern occurs if it matches all of the characters in Σ. At the $*$ spots any string can appear. For example, acb?a occurs in

  acbabbcaabbc bbbbb **acabaaaccabbaaadaacadaa**

  as shown.

Theorem 21.10. *Let $0 < \varepsilon < 1$.*

1. *There exists an MPC algorithm for string matching with pattern $P \in \Sigma^*$ with $M = O(n^{\varepsilon})$, $s = O(n^{1-\varepsilon})$, and $r = O(1)$.*
2. *There exists an MPC algorithm for string matching with $P \in \{\Sigma \cup ?\}^*$ with $M = O(n^{\varepsilon} polylog(n))$, $s = O(n^{1-\varepsilon} polylog(n))$, and $r = O(1)$.*

It is not known if pattern matching with $P \in \{\Sigma \cup *\}^*$ has a polylog round MPC algorithm with sublinear M and s.

21.4 Unconditional MPC Lower Bounds

In this section, we are going to present an approach to find unconditional lower bound for problems on the MPC model. This approach was introduced by Roughgarden et al. [RVW18]. (Everything in this section is from that paper.) We will do the following:

1. Define the s-Shuffle model.
2. State (but not prove) a relation between the MPC model and the s-Shuffle model.
3. Show that an s-Shuffle computation can be represented by polynomials.
4. Obtain lower bounds on the s-Shuffle model for some problems by looking at polynomials.
5. Use these lower bounds and the relation between the MPC model and the s-Shuffle model to get lower bounds on the MPC model.

The lower bounds are not tight; however, they are important because they are unconditional.

21.4.1 The s-Shuffle Model

In the s-Shuffle model, we have multiple machines. Each machine has s inputs and is located in one of R levels. These machines may seem like gates in a circuit but they are not. In particular, they can compute arbitrary functions, not just AND, OR, and NOT.

Each input of a machine at level R comes from a machine at a lower level. The inputs are 0, 1, or $\perp$ (we think of $\perp$ as meaning silence—no input). For each machine, at most one of its inputs can be other than $\perp$. In the following definition we describe how each input bit of each machine is determined.

Definition 21.11.

1. The $\perp$-sum of $z_1, z_2, \ldots, z_m \in \{0, 1, \perp\}$ is:
   - 1 if exactly one z_i is 1 and the rest are $\perp$;
   - 0 if exactly one z_i is 0 and the rest are $\perp$;
   - $\perp$ if every z_i is $\perp$;
   - undefined (or invalid) otherwise
2. The $\perp$-sum of s m-tuples is computed component-wise.

For each port of every machine, $\perp$-sum is used on signals received on that port to determine the output of the corresponding port. Now that we understand how each input handles multiple signals, we describe the s-Shuffle more formally.

Definition 21.12. An ***R* rounds *s*-Shuffle** is a model with a set of V machines such that each machine v has a round number $0 \leq r(v) \leq R + 1$. Round 0 indicates an input machine which receives input signals $x_1, x_2, \ldots, x_n$. Round $R + 1$ indicates an output machine which receives output signals $y_1, y_2, \ldots, y_k$.

1. For all pairs $u, v \in V$ so that $r(u) < r(v)$, we have a function $\alpha_{u,v} : \{0, 1, \perp\}^s \to \{0, 1, \perp\}^s$ which defines signals that machine u will send to all inputs of v according to the output of u, i.e. machine u will send $\alpha_{u,v}(g(u))$ where $g(u)$ is the output of machine u. Note that during a computation a machine will get many s-tuples.

2. We now define how the s-Shuffle computes. Recall that a machine v receives many signals (actually s-tuples). So what will machine v take to be its input? It will take the $\perp$-sum of all signals received component-wise.

Definition 21.13. The ***width*** of an s-Shuffle is the maximum number of machines in a round other than rounds 0 and $R + 1$.

We state the connection between the MPC model and the s-Shuffle model.

Theorem 21.14. *An MPC computation that runs on M machines with space s in R rounds can be simulated with an s-Shuffle instance with width M in R rounds.*

21.4.2 Polynomials

Definition 21.15.

1. We will be using polynomials on many variables. We will only care about what they output when the variables are replaced by 0's and 1's. Hence the polynomials will not have any exponents. Moreover, when we say $p \in \mathbb{Z}[x_1, \dots, x_m]$ we will implicitly mean that there are no exponents.

2. The ***degree*** of a polynomial is the number of variables in the largest monomial.

3. Let $f : \{0,1\}^m \to \{0,1\}$. Let $p \in \mathbb{Z}[x_1, \dots, x_m]$. ***$p$ represents f*** if, for all $\vec{b} \in \{0,1\}^m$, $f(\vec{b}) = p(\vec{b})$.

4. If f is a graph property on graphs with n vertices then the variables are $x_{i,j}$ (with $1 \le i < j \le n$) and represent edges.

Theorem 21.16. *For every $f : \{0,1\}^m \to \{0,1\}$ there exists $p \in \mathbb{Z}[x_1, \dots, x_m]$ of degree m such that f represents g.*

Proof. For every $\vec{b} \in \{0,1\}^m$ create a polynomial that is 1 if and only if the input is $\vec{b}$. For example, if $n = 4$ and the bit sequence is 0110 then we associate the polynomial

$$\mathsf{Poly}(\vec{b}) = (1 - x_1)x_2x_3(1 - x_4).$$

The polynomial p is

$$p(x_1, \dots, x_m) = \sum_{\vec{b} :\, f(\vec{b})=1} \mathsf{Poly}(\vec{b}).$$

□

Note We will connect the smallest degree of a polynomial that represents f to the complexity of f.

Exercise 21.17. Let AND$_n$ be the function $x_1 \wedge \cdots \wedge x_n$. Let OR$_n$ be the function $x_1 \vee \cdots \vee x_n$.

1. Give a polynomial of degree n that represents AND$_n$. Same for OR$_n$.

2. Show that any polynomial that represents AND_n has degree $\geq n$. Same for OR_n.

3. Show that any s-Shuffle for AND_n needs at least $\lceil \log_s d \rceil$ rounds. Same for OR_n.

Now, we are going to show that the output of an s-Shuffle computation with n input bits, and k output bits, running in R rounds, can be produced by k polynomials with degree of at most s^R such that each polynomial produces one of the s-Shuffle outputs.

Theorem 21.18. *Let $f : \{0,1\}^n \to \{0,1\}^k$.*

1. *If there is an r-round s-Shuffle for f, then there are k polynomials $\{p_i(x_1, \ldots, x_n)\}_{i=1}^k$ of degree at most s^r such that, for all $x \in \{0,1\}^n$, for $1 \leq i \leq k$, $p_i(x) = f(x)_i$.*

2. *If f cannot be represented by a polynomial with degree less than d, then any s-Shuffle that computes f has least $\lceil \log_s d \rceil$ rounds. (This follows from Part 1).*

Proof. First, for every machine v and value $\vec{z} \in \{0,1,\perp\}^s$, we introduce a polynomial $p_{v,\vec{z}}(x_1, \ldots, x_n)$ which is 1 if the output of machine v will be $\vec{z}$ when the input of our s-Shuffle instance is $\vec{x} = (x_1, \ldots, x_n)$, and otherwise, it is 0. Now, we are going to prove that the degree of $p_{v,\vec{z}}(\vec{x})$ is at most $s^{r(v)}$.

The base of our induction is input machines. For each input x_i, we have a specific machine with polynomial defined as follows:

- $p_{v,\mathbf{z}}(x_1, \ldots, x_n) = 1 - x_i$ for $\mathbf{z} = (0, \perp, \ldots, \perp)$
- $p_{v,\mathbf{z}}(x_1, \ldots, x_n) = x_i$ for $\mathbf{z} = (1, \perp, \ldots, \perp)$
- $p_{v,\mathbf{z}}(x_1, \ldots, x_n) = 0$ otherwise

These polynomials have degree at most $s^0 = 1$. For the induction step, we assume that for all machines in level $0, \ldots, r(v)-1$ like u, there are polynomials with degree at most $s^{r(u)}$ to compute the $p_{u,\mathbf{z}}(\mathbf{x})$ for all $\mathbf{z}$. Now, we want to find polynomials for $p_{v,\mathbf{z}}$. For each bit of $\mathbf{z}$ we have two cases:

1. $z_i \in 0, 1$: In this case, port i of machine v should get z_i from one of the previous machines and $\perp$ from other machines. Therefore, for each machine u with $r(u) < r(v)$ and for each $z' \in \{0,1,\perp\}^s$ such that $\alpha_{u,v}(z')$ has value z_i at entry i, we sum these $p_{u,z'}$. Given that just one of them should send a value other than $\perp$, this summation should not create any problem, i.e. for every input its value should be in $\{0,1\}$.

2. $z_i = \perp$: In the previous case we find polynomials when $z_i = 0$ or 1. So 1 minus these polynomials should give the answer for the case $z_i = \perp$.

Both polynomials in these two cases are made by summation of some polynomials with degree at most $s^{r(v)-1}$. Hence, the final degrees of these polynomials are at most $s^{r(v)-1}$. In the end, for each $\mathbf{z}$, we set $p_{v,\mathbf{z}}$ equal to product of all polynomials which we obtain for each entry of $\mathbf{z}$. This will give a polynomial with degree at most $s^{r(v)}$. □

21.4.3 Lower Bounds in the MPC Model for Monotone Graph Properties

We will use two known results to obtain a lower bound on s-Shuffles for monotone graph properties.

Definition 21.19.

1. A ***monotone graph property*** is a property of graphs such that if more edges are added then the property still holds. *Examples*: connectivity, non-planarity.

2. A ***Decision Tree for a graph property*** is a decision tree for the property where the queries are "$(i,j) \in E$?".

3. In this section (and only this section) a polynomial is over $\mathbb{Z}[x_1,\ldots,x_m]$.

Theorem 21.20. *Let P be a monotone graph property.*

1. *Rosenberg [Ros73] proved that any decision tree for P requires $\Omega(n^2)$ depth. (The constant for the Ω was increased by Rivest & Vuillemin [RV78] and Kahn et al. [KSS84].)*

2. *Buhrman & de Wolf [BdW02] proved that the decision tree complexity of a polynomial in $\mathbb{Z}[x_1,\ldots,x_n]$ of degree d is at most $O(d^4)$.*

3. *The degree of a polynomial representing a monotone graph property is at least $\sqrt{n}$. (This follows from Parts 1 and 2.)*

4. *Any s-Shuffle computation for P has at least $\Omega(\log_s n)$ rounds. (This follows from Part 3 and Theorem 21.18.)*

With more effort, the following has also been proved.

Theorem 21.21. *For the problem of graph connectivity the following hold.*

1. *Every s-Shuffle needs at least $\lceil \log_s \binom{n}{2} \rceil$ rounds.*

2. *Every MPC algorithm with space s needs at least $\lceil \log_s \binom{n}{2} \rceil$ rounds. (This follows from Theorem 21.14.)*

21.5 Conditional MPC Lower Bounds

In this section, we are going to present conditional lower bound for problems using an MPC algorithms. The framework for these lower bounds was introduced by Ghaffari et al. [GKU19]. (Everything in this section is from that paper unless otherwise noted.) We will do the following:

1. Define the 1vs2-Cycle problem and formally state the conjectured lower bound for it in the MPC model.

2. Assuming the conjectured lower bound for the 1vs2-Cycle, and additional assumptions, we will state lower bounds for other problems in the MPC model.

1vs2-Cycle
Instance: An undirected graph $G = (V, E)$ which we are promised is either one cycle or the union of two cycles.
Question: Determine whether the graph is one cycle or the union of two cycles.

The following conjecture is widely believed:

Conjecture 21.22. The 1vs2-Cycle Conjecture *Any MPC algorithm for the 1vs2-Cycle problem using machines with $s = n^\varepsilon$ requires $\Omega(f(\varepsilon)\log n)$ rounds for some f. (Note that since the input is one cycle or two cycles, $m = O(n)$ so strongly sublinear now means $s = n^\varepsilon$.)*

The lower bound is conjectured to hold even for the simpler promise problem of distinguishing whether an input graph is a cycle of length n or two cycles of length $n/2$.

Theorem 21.23. *Assume the 1vs2-Cycle conjecture. Any strongly sublinear MPC for undirected connectivity requires $\Omega(\log n)$ rounds.*

Proof. It is clear that if we have one cycle, the graph is connected, otherwise, the graph is unconnected. Thus, if we find an algorithm for undirected connectivity in $o(\log n)$ rounds, then we will have an algorithm in $o(\log n)$ rounds for 1vs2-Cycle, which contradicts the 1vs2-Cycle conjecture. □

We can try to find conditional lower bounds for other problems by making a reduction from 1vs2-Cycle or other problems that have a known conditional lower bound. But, alas, we need to introduce a restriction on the type of algorithms we can get lower bounds on.

Intuitively, a ***component-stable MPC algorithm*** is one where the output machines from different connected components (viewing the MPC algorithm as a graph) are independent. This notion only makes sense if the output machines are associated to vertices or edges of the graph. We give three examples of problems where we also carefully define the output machines.

Maximal Matching
Instance: An undirected graph $G = (V, E)$
Question: Return an independent set I such that no independent set is a proper superset of I.
Output: For every $v \in V$ there is an output machine. At the end of the computation the machine will have either a 1 (for $v \in I$) or a 0 (for $v \notin I$).

Sinkless Orientation
Instance: An undirected graph $G = (V, E)$
Question: Return an orientation of the graph (a direction for every edge) so that the graph has no vertices of outdegree 0. (Such vertices are called ***sinks***.)
Output: For every $\{u, v\} \in E$ there is an output machine. At the end of the computation the machine will have either (u, v) or (v, u) to indicate the orientation.

$(\Delta + 1)$-Graph Coloring
Instance: An undirected graph $G = (V, E)$ and its max degree Δ.
Question: Return a proper $(\Delta + 1)$-coloring of G (such always exists). Colors are $\{1, \ldots, \Delta + 1\}$.
Output: For every $v \in V$ there is an output machine. At the end of the computation the machine will have the color of v.

Definition 21.24. Let A be a graph problem where the output is data attached to each vertex (a similar definition is used if the output is data attached to each edge). A ***component-stable MPC algorithm*** for A has the following property: Let $u, v \in V$. Let M_u and M_v be the output machines associated to u, v. View the MPC algorithm as a directed graph. If u, v are in different components of G then M_u and M_v are in different components of the MPC algorithm.

The only known MPC algorithms for Maximal Matching, Sinkless Orientation, Δ-Graph Coloring, and many other graph problems are component-stable. Hence it is of interest to get lower bounds on component-stable MPC algorithms for these and other problems.

Theorem 21.25. *Assume the 1vs2-Cycle conjecture. Assume that all MPC's discussed in this theorem are strongly sublinear and use a component stable algorithm.*

1. *Maximal Matching requires $\Omega(\log\log n)$ rounds. (Same holds for a constant approximation for Maximal Matching. This lower bound holds even when restricted to trees.)*
2. *Sinkless Orientation requires $\Omega(\log\log\log n)$ rounds.*
3. *Let c be a constant. c-coloring a cycle requires $\Omega(\log\log^* n)$ rounds.*
4. *Any constant approximation for Vertex Cover requires $\Omega(\log\log n)$ rounds.*
5. *(Using additional assumptions) $(\Delta+1)$-coloring a graph requires $O(\sqrt{\log\log n})$ rounds.*

21.6 The Adaptive Massively Parallel (AMPC) Model

The Adaptive Massively Parallel (AMPC) Model was introduced by Behnezhad et al. [BDE+21]. It is essentially the MPC model but with shared memory. We quote their motivation:

Our model is inspired by the previous empirical studies of distributed graph algorithms [BBD+17, KLM+14], using MapReduce and a distributed hash table service [CDG+08].

We proceed informally.

In the AMPC model the machines have access to a shared memory. This model assumes that all messages sent in a single round are written to a distributed data storage, which all machines can read from within the next round. More specifically, the computation consists of rounds. In the i-th round, each machine can read data from a random access memory D_{i-1} and write to D_i (both D_{i-1} and D_i are common for all machines). Within a round, each machine can make up to S reads (henceforth called queries) and S writes and can perform arbitrary computation. (Note that this S is different from the s parameter for the MPC model.)

Clearly, every MPC algorithm can be easily simulated in the AMPC model. However, the opposite direction is not usually the case. Furthermore, due to known simulations of PRAM algorithms by MPC, the AMPC model can also simulate existing PRAM algorithms. The key property of the AMPC model is that the queries a machine makes in each round may depend on the results of the previous queries it made in that same round, which is why the model is called *adaptive*. For example, if g is a function from X to X and for each $x \in X$, D_{i-1} stores a key-value pair $(x, g(x))$, then in round i a machine can compute $g^k(y)$ in a single round, provided that $k = O(S)$.

21.6.1 AMPC Power: 1vs2cycle Revisited

In this section, we discuss the computation power of the AMPC model. For several fundamental problems, there are AMPC algorithms solving them with significantly lower round complexities than the best-known MPC algorithms. Behnezhad et al. [BDE+19a] includes a number of such algorithms for some of the most fundamental graph problems, such as Graph Connectivity and Maximal Independent Set. We focus on the 1vs2-Cycle problem since this illustrates that the AMPC is likely more powerful than the MPC model.

Recall that in the MPC model, it is conjectured that 1vs2-Cycle requires a logarithmic number of rounds. However, the following theorem (from [BDE+19a] with help from [BDE+21]) shows that this conjecture does not hold in the Adaptive model.

Theorem 21.26. *There is an AMPC algorithm solving the 1vs2-Cycle problem in $O(1/\varepsilon)$ rounds w.h.p. using $O(n^{\varepsilon})$ space per machine and $O(n)$ total space.*

Proof. At first, we discuss the overview of their algorithm. In each round of the algorithm, we sample each vertex with probability $n^{-\varepsilon/2}$. Then, we contract the original graph to the samples by replacing the paths between sampled vertices with single edges. To do so, we traverse the cycle in both directions for each sampled vertex until we hit another sampled vertex, which can be done in a single round using the adaptivity of the model. In each round, with high probability, the number of vertices shrinks by a factor of $n^{\varepsilon/2}$. Therefore, after $O(1/\varepsilon)$ rounds, the number of remaining vertices and edges is reduced to $O(n^{\varepsilon})$. Hence, the graph fits in the memory of a single machine, and we can solve the remaining problem in a single round.

We present the algorithm. We first need a procedure

```
Shrink($G = (V, E), \varepsilon, t$)
Begin Algorithm
  For $i = 0, \ldots, t$
    $V' \leftarrow$ a subset of $V(G)$ s.t. each vertex is included independently with probability $n^{-\varepsilon/2}$
    $E' \leftarrow \emptyset$
  EndFor

  For $v \in V'$
    $l_v \leftarrow$ first sampled vertex that we reach traversing the graph starting with $(v, nei_G^1(v))$
    $r_v \leftarrow$ first sampled vertex that we reach traversing the graph starting with $(v, nei_G^2(v))$
    State $E' \leftarrow E' \cup \{(v, l_v), (v, r_v)\}$
    $G \leftarrow G'(V', E')$
  EndFor
  Return G
End Algorithm
```

And now the algorithm.

```
1vs2-Cycle$GV, E$
Begin Algorithm
  $G' \leftarrow$ Shrink$(G, \varepsilon, O(1/\varepsilon))$
  Solve the 1v2-Cycle problem on $G'$ using a single machine
  (Comment: Note that $|V(G')| \in O(n^{\varepsilon})$ w.h.p.)
```

End Algorithm

Each iteration of the outer loop in Shrink is a single AMPC round, and the correctness of this algorithm is a result of combining the following lemmas from Behnezhad et al. [BDE+21].

Lemma 21.27. Let G be a graph consisting of cycles, and let N be the initial number of vertices in G. Consider a cycle with size $k = \Omega(N^{\varepsilon})$ in some iteration of the loop of Shrink $(G, \varepsilon, O(1/\varepsilon))$. The size of this cycle shrinks by at least a factor of $N^{\varepsilon/2}$ after this iteration w.h.p.

Lemma 21.28. Let G be a N-vertex graph consisting of cycles and let $G' = \text{Shrink}(G, \varepsilon, O(1/\varepsilon))$. Then G' can be obtained from G by contracting edges, and the length of each cycle in G' is $O(n^{\varepsilon})$.

Lemma 21.29. In each round, the total communication of each machine is $O(N^{\varepsilon})$ w.h.p., where N is the initial number of vertices in G.

□

21.6.2 Unconditional AMPC Lower Bounds

Recall that in Section 21.4 we presented the polynomial method for lower bounds on MPC algorithms. Charikar et al. [CMT20] modified the polynomial method to get lower bounds on AMPC algorithms. Everything in this section is from that paper.

They introduced the following extension of degree.

Notation 21.30.

1. In the definitions below $\Delta \subseteq \{0,1\}^n$. For example Δ could be the set of two graphs: one the cycle on n vertices, and the other the disjoint union of two cycles of length $n/2$. We say things like:

   Let g be a partial boolean function that maps Δ to $\{0,1\}$.

   to emphasize that g can be partial.

2. We use the same convention as in Section 21.4 whereby one can input a graph on n vertices into a polynomial on $\binom{n}{2}$ variables by viewing the graph as the a sequence of $\binom{n}{2}$ bits; where $x_{i,j}$ is 1 if (i,j) is an edge and 0 otherwise.

Definition 21.31. Let $g : \Delta \longrightarrow \{0,1\}$. Then

$$deg_{partial}(g) = \min\{deg(p) \mid p(x) = g(x) \text{ for all } x \in \Delta\}.$$

The next theorem reduces lower-bounding the deterministic AMPC round complexity of g to lower-bounding $deg_{partial}(g)$:

Theorem 21.32. *Let g be a partial boolean function that maps Δ to $\{0,1\}$. Let M be a deterministic AMPC algorithm that computes g. Let S be the number of queries and writes allowed per round. The number of rounds is at least*

$$\frac{1}{2}\log_S deg_{partial}(g).$$

In particular, if g is a total Boolean function, then any such algorithm requires $\frac{1}{2}\log_S deg(g)$ rounds.

We omit the proof.

Theorem 21.32 provides lower bounds for deterministic algorithms. What about randomized algorithms? To obtain lower bounds on randomized algorithms we need the following definition.

Definition 21.33. Let $g : \Delta \to \{0, 1\}$.

1. g is approximately represented by a polynomial p if
$$\forall x \in \Delta : |p(x) - g(x)| \le \frac{1}{3}.$$
2. $\widetilde{deg}_{partial}$ is
$$\widetilde{deg}_{partial}(g) = \min\{deg(p) : p \text{ approximately represents } g\}.$$

The following theorem reduces lower-bounding the randomized AMPC round complexity of g to lower-bounding $\widetilde{deg}_{partial}(g)$:

Theorem 21.34. *Let g be a partial boolean function that maps Δ to $\{0, 1\}$. Let M be a randomized AMPC algorithm that computes g with probability of error $\le \frac{1}{3}$. Let S be the number of queries and writes allowed per round. The number of rounds is at least*
$$\frac{1}{2}\log_S \widetilde{deg}_{partial}(g).$$
In particular, if g is a total Boolean function, then any such algorithm requires $\frac{1}{2}\log_S \widetilde{deg}(g)$ rounds.

We omit the proof.

PARITY
Instance: $x \in \{0, 1\}^n$. Let $x = x_1 \cdots x_n$.
Question: What is $\sum_{i=1^n} x_i \pmod 2$?
Note: We denote PARITY on n bits by PARITY$_n$.

Paturi [Pat92] proved the following:

Theorem 21.35. *If p is a polynomial that approximates PARITY$_n$ then* $\deg(p) \ge n$.

By combining Theorems 21.35, 21.32, and 21.34 one obtains the following:

Theorem 21.36.

1. *If M is a deterministic AMPC algorithm for PARITY$_n$ then the number of rounds is $\ge \frac{1}{2}\log_S(n)$.*
2. *If M is a randomized AMPC algorithm with error $\le \frac{1}{3}$ for PARITY$_n$ then the number of rounds is $\ge \frac{1}{2}\log_S(n)$.*
3. *In both of the items above, if $S = n^\epsilon$, the number of rounds is $\Omega(\frac{1}{\epsilon})$*

Now that we have *one* problem with a provable unconditional lower bound we can use a reduction get others!

Theorem 21.37.

1. *If M is a deterministic AMPC algorithm for 1vs2-Cycle then the number of rounds is $\geq \frac{1}{2}\log_S(n/2)$.*

2. *If M is a randomized AMPC algorithm with error $\leq \frac{1}{3}$ for 1vs2-Cycle then the number of rounds is $\geq \frac{1}{2}\log_S(n/2)$.*

3. *In both of the items above, if $S = n^{\epsilon}$, the number of rounds is $\Omega(\frac{1}{\epsilon})$*

Proof. We prove the theorem using a reduction from Parity$_N$ to 1vs2-Cycle. (We use N since we will use n for the number of vertices in a graph.)

Let $x = \{x_1, \ldots, x_N\} \in \{0,1\}^N$ be an instance of Parity$_N$. We construct the graph $G(x)$ as follows:

1. For any x_i we add vertices v_i^1 and v_i^2.

2. For $i = 1, \ldots, N$

   (a) if $x_i = 0$, add edges $(v_i^1, v^1_{(i \mod N)+1})$ and $(v_i^2, v^2_{(i \mod N)+1})$;

   (b) if $x_i = 1$ add edges $(v_i^1, v^2_{(i \mod N)+1})$ and $(v_i^2, v^1_{(i \mod N)+1})$.

Figure 21.1 shows the graph G (the right two vertices are the left two vertices) obtained if the input is 01001. Note that Parity(01001) = 0 and G is the union of two cycles of length n. We leave it to the reader to show the following:

- If Parity$(x) = 0$ then $G(x)$ is the union of 2 cycles of length n.

- If Parity$(x) = 1$ then $G(x)$ is the 1 cycles of length $2n$.

We have a reduction where a string of length N maps to a graph on $2N$ vertices. Hence, by Theorem 21.36, any AMPC algorithm (of the two types we are talking about) for 1vs2-Cycle on $2N$ vertices requires $\geq \frac{1}{2}\log_S(N)$ rounds. Hence any AMPC algorithm (of the two types we are talking about) for 1vs2-Cycle on n vertices requires $\geq \frac{1}{2}\log_S(n/2)$ rounds. □

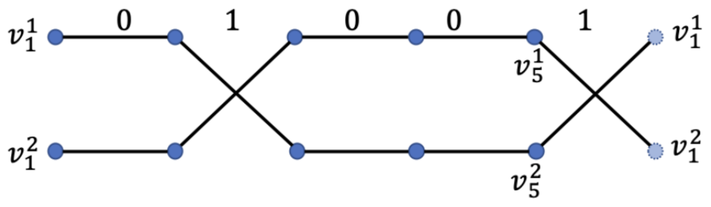


Figure 21.1: Reduction from a Parity instance $x = 01001$ to a 1v2-Cycle instance with 10 vertices.

Theorem 21.38.

1. *If M is a deterministic AMPC algorithm for 1vsk-Cycle then the number of rounds is $\geq \frac{1}{2}\log_S(n/k^2)$.*

2. *If M is a randomized AMPC algorithm with error $\le \frac{1}{3}$ for 1vsk-Cycle then the number of rounds is $\ge \frac{1}{2}\log_S(n/k^2)$.*

3. *In both of the items above, if $k = n^\delta$ and $S = n^\epsilon$, the number of rounds is $\Omega(\frac{1-2\delta}{\epsilon})$*

Proof. We give a sketch of the proof since it is similar to the proof of Theorem 21.37.

We prove the theorem using a reduction from Parity$_N$ to 1vsk-Cycle. (We use N since we will use n for the number of vertices in a graph.)

Let $x = \{x_1, \ldots, x_N\} \in \{0,1\}^N$ be an instance of Parity$_N$. we want to construct a graph $G(x)$ such that

- If Parity$(x) = 0$ then $G(x)$ is the union of k cycles of length n.
- If Parity$(x) = 1$ then $G(x)$ is the 1 cycles of length kn.

We leave it to the reader to use Figure 21.2 to guide their construction and then to finish the proof. □

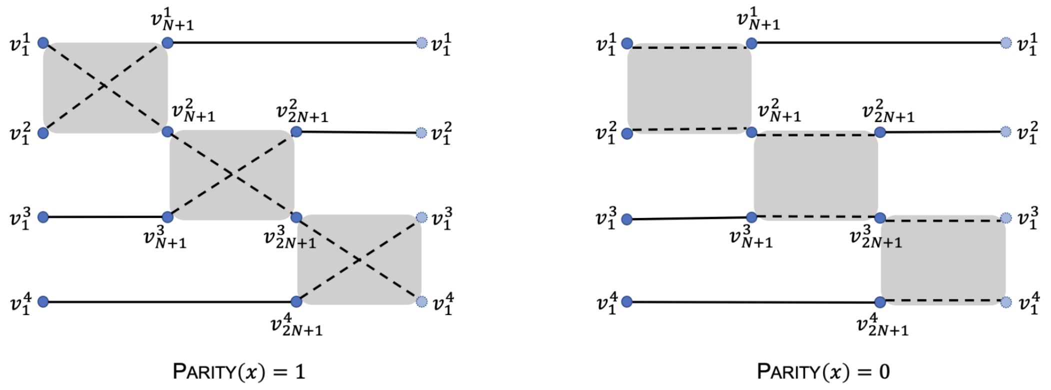


Figure 21.2: Reduction from a Parity instance to a 1vk-Cycle instance with $k = 4$.

21.7 Future Directions

Despite the recent interest in the MPC and AMPC models, there are many fundamental open problems. For example, the MPC-complexity and AMPC-complexity of both 2SAT and directed s-t connectivity remain unknown. These problems are related in the sense that solving 2SAT is usually done by reducing it to directed s-t connectivity.

It is also a challenge to find more problems where AMPC algorithms are provably better than MPC algorithms. s-t connectivity may be such a problem.

Chapter 22

Nash Equilibria and Polynomial Parity Arguments for Directed Graphs

22.1 Introduction

There are problems that are in P for an odd reason. We give an example.

Definition 22.1. Let $G = (V, E)$ be a directed graph. A vertex $v \in V$ is ***unbalanced*** if $\text{indeg}(v) \neq \text{outdeg}(v)$.

Theorem 22.2. *Let G be a directed graph.*

1. $\sum_{v \in V} indeg(v) = \sum_{v \in V} outdeg(v)$.

2. *If there exist an unbalanced vertex then there exists another unbalanced vertex. (This follows from Part 1.)*

Proof. Every edge contributes one to the sum of indegrees and one to the sum of outdegrees. Hence both sums are equal to the number of edges. □

Note that the proof of Theorem 22.2 is ***nonconstructive*** in that it does not help us find the other unbalanced vertex.

Consider the following problem.

UNBALANCED
Instance: A directed graph G and an unbalanced node v.
Question: Is there another unbalanced vertex?

This problem is in P for the odd reason that, by Theorem 22.2, there is *always* another unbalanced vertex. What if we actually want to *find* that other unbalanced vertex? We certainly can find it in polynomial time. But note that the algorithm does not use the proof of Theorem 22.2.

What if we were given an exponentially large graph? We will also restrict the graph to have both indegree and outdegree ≤ 1.

End of Line EOL
Instance: Two circuits P (for Previous) and N (for Next) on $\{0,1\}^n$ such that for all $x \in \{0,1\}^n$, $P(x)$ ()$N(x)$) returns either NO or an element of $\{0,1\}^n$. We interpret the circuits as describing a graph by the following: If $P(x) = y$ ($N(x) = y$) then there is an edge from x to y (y to x). Since these are the only edges, every vertex has indegree and outdegree ≤ 1. Exactly one of $P(0^n)$ and $N(0^n)$ is NO. So 0^n is unbalanced.
Question: We know there is another unbalanced node. Find it!

Figure 22.1 is a picture of a circuit we may use as input, and Figure 22.2 is a picture of a graph that that circuit might describe.

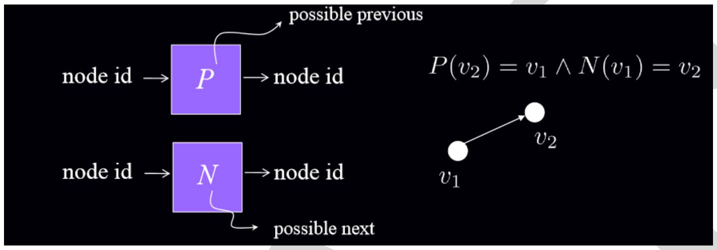


Figure 22.1: The Input to EOL

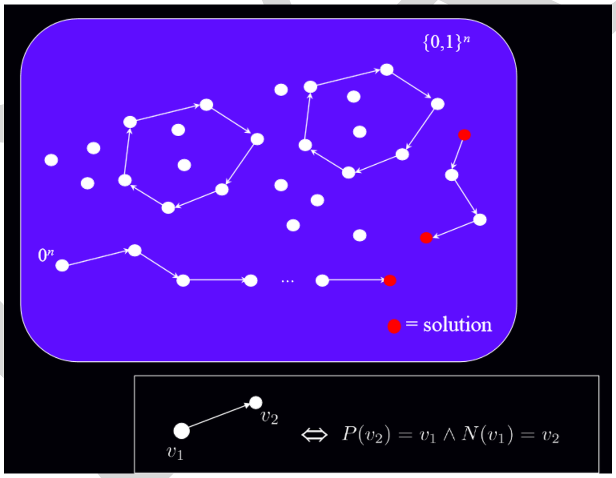


Figure 22.2: The Graph Produced by the EOL Input

Theorem 22.2 tells us that there is another unbalanced VERTEX but, since the proof is non-constructive, it does not help us fine another unbalanced vertex. So it seems that EOL is hard. Is it NP-hard? Unlikely:

Theorem 22.3. *If SAT is reducible to EOL using a polynomial number of queries then NP = coNP.*

Proof. Assume there is such a reduction. Then $\varphi \notin$ SAT if and only if that reduction returns NO.

If the reduction says NO then look at that computation. It will have instructions and queries. The key is that that the queries to EOL all have answers that can be verified (you an verify that a vertex v is unbalanced by looking at $P(v)$ and $N(v)$). So the reduction can be encoded in a string that has the answers to the queries and the verification of those answers. This string is of length polynomial in $|\varphi|$.

Hence if $\varphi \notin$ SAT then there is a short string that verifies the reductions answer of NO. This puts $\overline{SAT}$ into NP, so SAT is in coNP. □

In this chapter we will assume that EOL is hard and use that to show that other problems are hard. The motivation for this will be to show that finding Nash equilibrium for 2-player games is hard. The proof that Nash equilibrium exists is, like that an unbalanced vertex exists, nonconstructive.

We will analyze existence theorems by Nash, Brouwer, and Sperner that arise from game theory, topology and combinatorics, respectively. These three theorems, as dissimilar as they seem, can be related to one another and rely on very basic combinatorial principles. In this section, we will define the setting for each of the theorems and give an overview of their connection.

The motivation to study these problems is that, unlike many problems in computer science, we already know that a solution exists. How efficiently can we find a solution? If the search space is polynomial, this is easy: simply search exhaustively. However, some of these problems, like EOL have an exponential size search space.

22.2 Game Theory

Problems in Game Theory have the following.

1. A set of players (often 2).
2. For each player a choice of strategies.
3. For each players choices of strategies, a payoff for each player.

The key question is to figure out the best strategy. It may be randomized, e.g., flip a coin and based on the coin flip choose a strategy.

We give several examples of problems in game theory.

22.2.1 Prisoners' dilemma

Two prisoners are on trial for a crime and each face the choice of confessing to the crime or remaining silent. If they both remain silent, the authorities will not be able to charge them for this particular crime, and they will both face two years in prison for minor offenses. If one of them confesses, his term will be reduced to one year, but he will have to bear witness against the other, who will be sentenced to five years. If they both confess, they will both get a small break and be sentenced to four years in prison (rather than five). We summarize the four outcomes and the utility with the matrix in Figure 22.3.

The only ***stable solution*** in this game is when both confess. In each of the other three outcomes, a prisoner can switch from being silent to confessing in order to improve his own payoff. The social optimum in this case is when both remain silent; however, this outcome is not stable. In this game, there is a unique optimal selfish strategy for each player, independent of what other players do. A pair of strategies where neither player has an incentive to change, independent of

P2 / P1	*Confess*	*Silent*
Confess	4 4	5 1
Silent	1 5	2 2

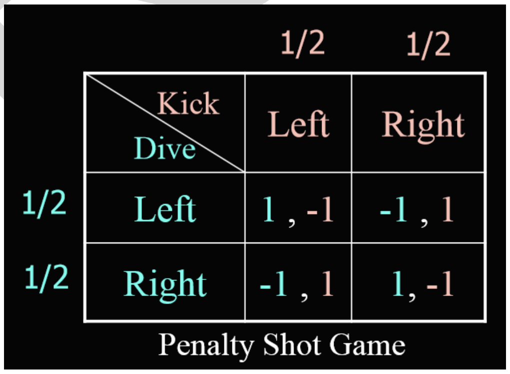
Figure 22.3: Prisoner's Dilemma

the other players strategy, will be called a ***Nash Equilibrium*** (NE). In this case confess-confess is an NE.

One way to specify a game in algorithmic game theory is to explicitly list all possible strategies and utilities of all players. Expressing the game in this form is called the ***standard form*** or matrix form, and it is convenient to represent two-player games with a few strategies in this form, as demonstrated for the Prisoner's dilemma game.

22.2.2 The Penalty Shot Game

Consider the Penalty shot game: Player 1 needs to decide where to shoot a penalty (left or right) and Player 2 needs to decide where to dive (left or right). The payoff of this game is easy to describe: if Player 1 scores, he gets a point and player 2 gets -1 points. If Player 1 misses, Player 2 gets a point and Player 1 gets -1 points. This game is summarized in Figure 22.4



		1/2	1/2
	Kick / Dive	Left	Right
1/2	Left	1 , -1	-1 , 1
1/2	Right	-1 , 1	1, -1

Penalty Shot Game

Figure 22.4: The Penalty Shot Payoff Matrix

Imagine that Player 1 initially decides to shoot left. If Player 2 knows this then he will dive left. If Player 1 knows that Player 2 will dive left he will shoot right. This paragraph could go on forever.

Is there a pair of strategies so that neither player has a reason to deviate from his strategy? If we insist that the strategies are deterministic then no: both players will want to deviate. However, there is a pair of *randomized* strategies: both players flip a fair coin to determine what to do.

22.2.3 Formal Game Theory

Definition 22.4. A ***finite game*** consists of the following elements:

- A set P of r players.
- For each $p \in P$ a set S_p of n pure strategies for p. A pure strategy means a deterministic strategy. In the 2-player case these will be the rows for Player I and the columns for Player II.
- A utility or payoff function that assigning a real value to player p for every possible strategy set. Formally, for every $p \in P$ we have a function

  $$u_p : \times_{q \in P} S_q \rightarrow \mathbb{R}.$$

  We use u for utility. (In the 2-player case this is the matrix of pairs as seen in both the Prisoner's Dilemma (Figure 22.3) and the Penalty Shot Game (Figure 22.4).

Definition 22.5. Let P be a game and p be a player with the set S_p of pure strategies. A ***mixed strategy for player p*** is a distribution over S_p. Formally if the strategies are $s_1, \ldots, s_n$ then a mixed strategy is a set of reals $r_1, \ldots, r_n$ such that $\forall i : 0 \le r_i \le 1$ and $\sum_{i=1}^{n} r_i = 1$.

For the Prisoner's Dilemma and the Penalty Shot game we looked at pairs of strategies where neither player has an incentive to change their mind. We define this formally.

Definition 22.6. A ***Nash Equilibrium (NE) for a game*** is a collection of mixed strategies $s_1, s_2, \ldots, s_n$ such that for every player $p \in P$, for every mixed strategy s'_p for p,

$$E(u_p(s_1, s_2, \ldots, s_p, \ldots, s_n)) \ge E(u_p(s_1, s_2, \ldots, s'_p, \ldots, s_n))$$

Informally, this definition says that a tuple of strategy (one for each player) is NE if no player has incentive (i.e., can't be better off) to change his strategy based on the strategies of the other players, in terms of expected utility.

Example 22.7.

1. In the Prisoner's Dilemma the confess-confess pair is an NE of pure strategies.
2. In the Penalty Shot game, the scenario where both players flip a fair coin to determine their move is an NE of mixed strategies

We are interested in the complexity of finding the NE.

r-Nash
Instance: An r-player game where all of the utilities are integers.
Question: Output an approximation to a NE. The discussion below will clarify why we settle for an approximation. We omit details of how the approximation works; however, you would need to have the error tolerance ε as part of the input. (We use the term Nash to mean r-Nash for some r.)

Is it possible that a NE uses irrational probabilities? Is it possible that a NE uses rationals but the numerator or denominator are exponential the length of the problem? Either of these would make asking for an exact NE a hopeless request. The following is known.

1. In Nash's original paper he gave an example of a 3-player game where all of the NE used irrational numbers.

2. Lemke & Howson [EJ64] showed that every 2-player game with integers utilities has a rational NE.

3. Cottle & Dantzig [CD68] showed that the rational NE for a 2-player game has numerators and denominators that are of size a polynomial in the size of the problem.

The following was known about the complexity of NE before complexity theory was made rigorous.

1. In 1928, von Neumann [Neu28] showed that in two-player zero-sum games (one where if Player i has utility x then Player $1 - i$ has utility $-x$) there is always an NE. His proof was by the minimax theorem which can be interpreted as a polynomial-time algorithm. In fact, this technique is a special case of strong Linear Programming Duality. In our notation we say that 2-Nash restricted to zero-sum games is in P.

2. In 1950, Nash [Nas50] showed that all games have an NE. Unfortunately his proof cannot be made into a polynomial time algorithm. In our notation we say that he was unable to prove that 2-Nash is in P.

What is the complexity of 2-Nash? It seems to be hard to compute. However, by the next exercise, it unlikely to be NP-hard.

Exercise 22.8. 2-Nash is NP-hard then NP = coNP.
Hint: This is similar to the proof of Theorem 22.3.

22.3 Brouwer's Fixed Point Theorem

Brouwer's famous Fixed Point Theorem [Bro50] is as follows.

Theorem 22.9. *Let D be a subset of Euclidean space. Let f be a function from D to D. Assume f, D satisfy the following three properties:*

1. *D is a convex set.*

2. *D is compact.*

3. *f is continuous.*

Then there exists $x \in D$ such that $f(x) = x$.

Figure 22.5 is a picture of an example.

We note that Brouwer's theorem is tight in that if any of the conditions are not met then the theorem is not true. We also note that the proof of Brouwer's theorem is nonconstructive in that the proof will not help you find the fixed point quickly.

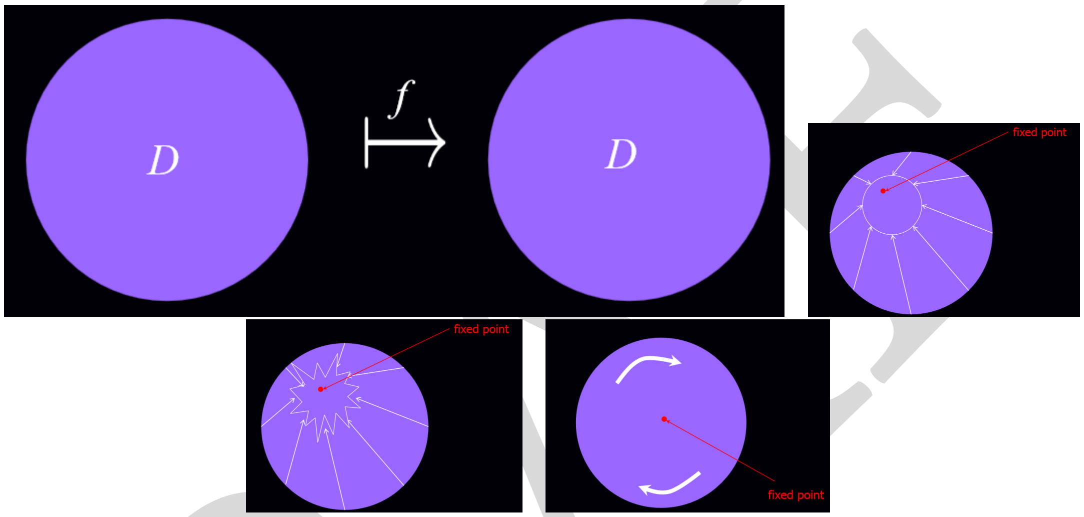
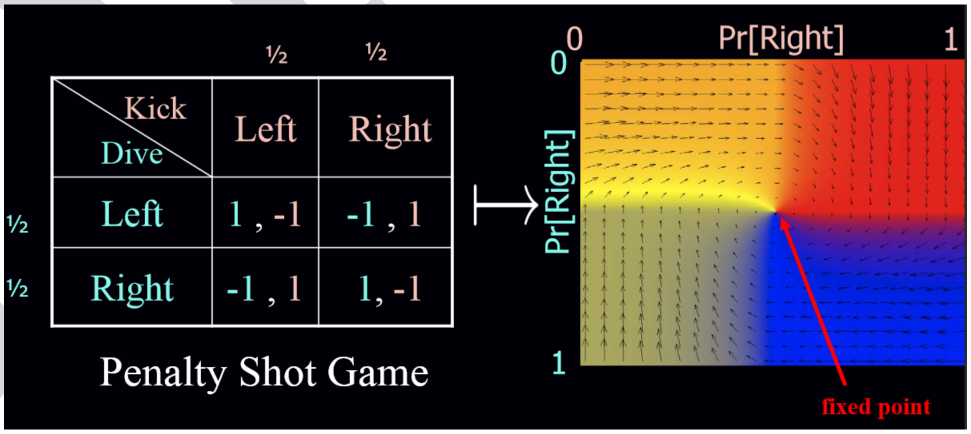


Figure 22.5: Brouwer's Fixed Point Theorem

It is worth noting that Nash used Brouwer's theorem to show his result for general games. Roughly, the proof involves a function $f : [0,1]^n \longrightarrow [0,1]^n$ as a vector field that indicates the motivation a player has to deviate from his current strategy. The NE corresponds to the fixed point of the mapping. Figure 22.6 depicts the function for the Penalty shot game.



		½	½
	Kick / Dive	Left	Right
½	Left	1 , -1	-1 , 1
½	Right	-1 , 1	1, -1

Penalty Shot Game

Figure 22.6: The Proof of Nash Using Brouwer

The computational problem that arises from Brouwer's fixed point theorem is to, given f, D satisfying the conditions of Theorem 22.9, find the fixed point. There is a major problem with this problem. f is continuous! D is a subset of the reals! This entire book has been about discrete problems! Not to worry, there is a discrete version of this problem, suitable for study. We do not define it; however, it was defined by Papadimitriou [Pap94] (Page 511). We will refer to this problem as BROUWER.

22.4 Sperner's Lemma

Sperner's lemma is about coloring of n-dimensional objects; however, we will look at the special case where the dimension is 2.

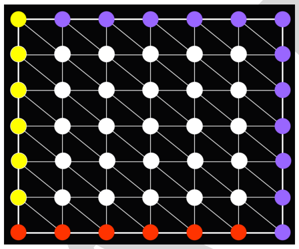


Figure 22.7: Valid Edge Coloring for the Sperner Edge Coloring

Sperner proved the following:

Definition 22.10. Let COL be a 3-coloring of the lattice points of an $n \times n$ grid. For both definitions here see Figure 22.7.

1. COL is ***valid*** if all vertices on the bottom row are RED, all vertices in the leftmost column are YELLOW, and all other boundary nodes are BLUE. (There is no condition on the non-boundary nodes.)

2. For all squares in the grid draw the line from the upper left to the bottom right. Hence we now have many triangles. A ***trichromatic triangle*** is a triangle where all of the vertices have different colors.

Theorem 22.11. *For all valid 3-coloring of the lattice points of an $n \times n$ grid there is a trichromatic triangle. In fact, there will be an odd number of them.*

Proof. We give two proofs.

Proof One

First, we will add an artificial trichromatic triangle by adding a blue vertex next to the bottom left corner of the grid, where the yellow and red boundaries meet (see Figure 22.8). We now define a directed walk on the triangles of the grid graph inductively. We start in the artificial triangle and leave it though the yellow-red edge with yellow to the left. If we arrive in a trichromatic triangle we are done. If not then the other node is red or yellow. Hence there will be a way to leave by going over a yellow-red edge with yellow on the left. Keep doing this: leave a triangle through the yellow-red edge with yellow on the right and either (a) you are in a trichromatic triangle so

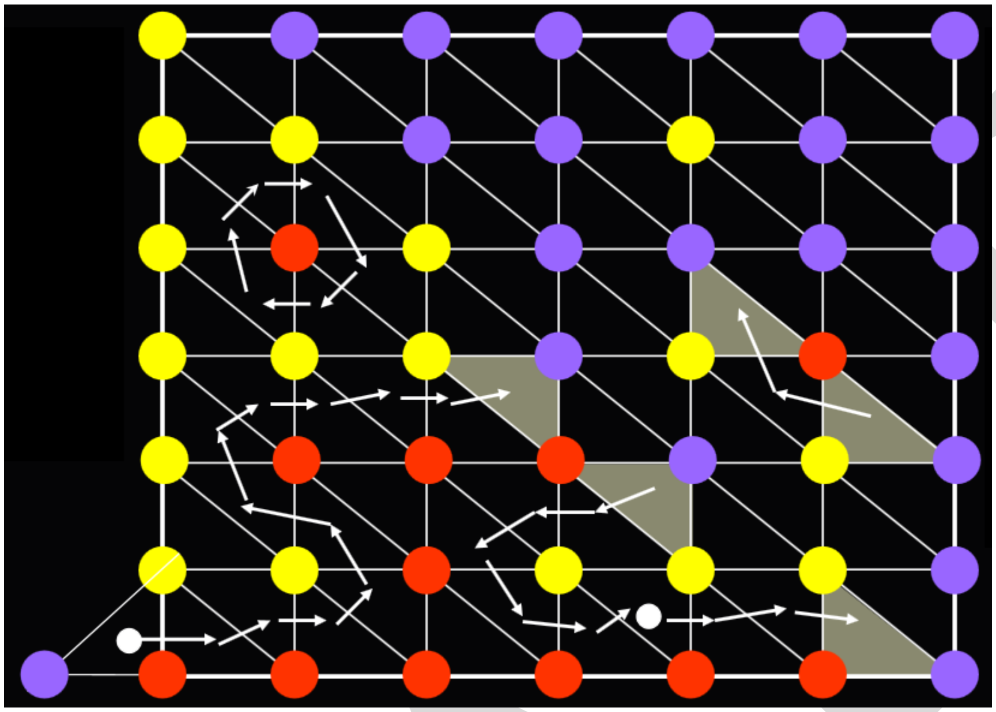


Figure 22.8: A Valid Walk in a Valid Coloring of a Triangulation of the Lattice for Sperner's Lemma

you are done, or (b) the other node is yellow or red so repeat. We claim that this procedure will find another trichromatic triangle.

Note that this walk can not exit the grid graph with legal boundary coloring: the only red-yellow edge on the boundary is the one on the bottom left corner, and in order to cross it we would have to have the yellow node on our right. This is not allowed. Moreover our walk will not produce a cycle. For the sake of contradiction, suppose that it did. Consider a triangle where the loop closes. This triangle must have had a red-yellow edge that we crossed the first time with yellow to the left. However, on our way back in, any edge we cross will either have red to the left or yellow to the right. Neither of these options is admissible. Therefore, there are no cycles and we never leave the grid graph.

This, together with the fact that the number of triangles is finite, implies that at some point we must encounter a trichromatic triangle, since that is when our walk stops. Therefore, at least one such triangle must exist.

What about any other trichromatic triangles? Well, we can perform the same procedure with the other internal trichromatic triangles. Start another walk from one such triangle and, by the same argument above, we will end at another.

Therefore, the total number of trichromatic triangles in our modified graph is an even number that is at least 2. However, one of these triangles was artificially introduced. Therefore, the number of trichromatic triangles inside the grid graph must be an odd number that is at least 1.

End of Proof One

Proof Two

This proof is more basic. Consider a directed graph where each node represents a triangle. There is an edge (u, v) if triangles u, v are adjacent and the edge that they share is a 'crossable' edge (i.e. yellow on the left).

We claim that every vertex must have either indegree and outdegree at most 1. This can be done by an exhaustive analysis of the possibilities. It is important to note that if a triangle has exactly one red-yellow edge (hence it's trichromatic), then its node will have either exactly indegree 1 and outdegree 0 or outdegree 1 and indegree 0.

But what can we say about a directed graph where every node has indegree and outdegree at most 1? Well, there can only be three types of (weakly) connected components: isolated nodes, cycles or directed paths. These paths correspond to the paths we discovered in the walk above and imply that the trichromatic triangles come in pairs. In a more elementary way, the underlying principle is that if a directed graph has an unbalanced (i.e. indegree is not the same as outdegree) node, then there must be another one.

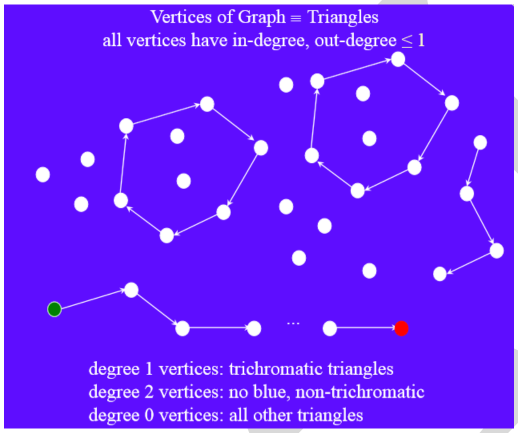

Vertices of Graph = Triangles
all vertices have in-degree, out-degree ≤ 1
...
degree 1 vertices: trichromatic triangles
degree 2 vertices: no blue, non-trichromatic
degree 0 vertices: all other triangles

Figure 22.9: The Triangulation of Sperner's Lemma Turned into a Directed Graph

In our example, the unbalanced nodes correspond exactly to trichromatic triangles. Since in our construction we introduce one such node, we are guaranteed that our graph will contain another one and a total even number of them. This property is also known as the Parity Argument for Directed Graphs, and plays a key role in the definition of the PPAD class.

□

There is also a natural connection between Sperner's and Brouwer's theorems. We can consider the problem of finding approximate fixed points for a function f from the unit square to itself. Given ε we want to find x such that $|f(x) - x| < \varepsilon$. We will create a grid on the unit square where the lines are δ apart (later we will set δ very small). We color each point p of grid as follows:

- Yellow if the vector from p to $f(p)$ points right, though can be at an angle. Note that all of the grid points on the left will be yellow which fits the premise of Sperner's lemma. (We also color yellow if it points straight up or straight down.)

- Red if the vector from p to $f(p)$ points up, though can be at an angle Note that all of the grid points on the bottom will be red which fits the premise of Sperner's lemma. (We also color red if the it points left or right.)

- Blue in all other cases. Note that all of the grid points on the top or the right will be blue which fits Sperner's lemma.

We can then use a compactness argument and let $\delta \to 0$ to prove the existence of an approximate fixed point (which will be inside the trichromatic triangle). This proof however might not preserve the parity or even the number of trichromatic triangles.

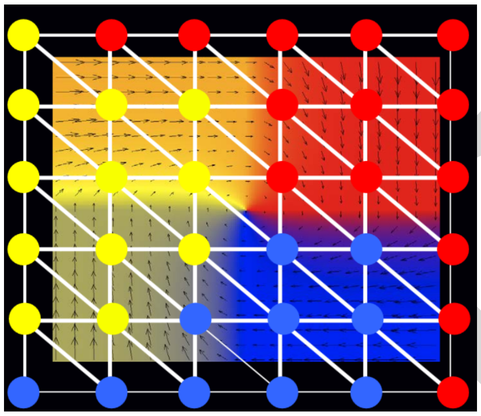


Figure 22.10: The Overlay of Sperner's Coloring on a Function Mapping $[0,1]^2 \to [0,1]^2$

We now define the problem Sperner for 2-dimensions, called 2-Sperner.

2-Sperner
Instance: A circuit that has input $\{0,1\}^n \times \{0,1\}^n$ and output $\{0,1,2\}$ (the three colors). (We do not test if the coloring is valid.)
Question: Either output a trichromatic triangle or output that the coloring is not valid. (It is okay to output a trichromatic coloring even if the coloring is not valid. It is likely that an algorithm will proceed, find an alleged trichromatic coloring, and then test it is trichromatic, output it, if not then output that the coloring is not valid.)

We omit the definition of the general n-dimensional Sperner problem since it is somewhat technical. It can be found in Papadimitriou [Pap94] (page 507-510). We use the term Sperner to refer to the n-dimensional problem for all n.

22.5 Total Search Problems in NP

In the next definition we state two types of problems we have discussed in this book, and two we have not, to show the contrast.

Definition 22.12.

1. ***Decision Problems***: Let A be a set. The problem is, given x, is $x \in A$. Example: SAT. Note that this is NP-hard (actually NP-complete).

2. ***Functions***: Let f be a function. The problem is, given x, find $f(x)$. Example: Given φ output the least lexicographic satisfying assignment if there is one, and 0 if there is not one. Note that this is NP-hard.

3. ***Search Problems*** (this is new): Let R be a relation. The problem is, given x, find *some* y such that $R(x, y)$, or output 0 if there isn't any. Example: Given φ find some y such that $\varphi(y) = \text{TRUE}$, and output 0 if there is no such y. Note that this is NP-hard.

4. ***Total Search Problems*** (this is new): Let R be a relation *where we are promised that, for all* x*, there is a short* y *with* $R(x, y)$. The problem is, given x, find *some* y such that $R(x, y)$. Examples: EOL, 2-Nash, Brouwer, and Sperner. (We do not include 3-Nash or r-Nash since, as noted earlier, these may have irrational NE. We will later look at approximating them.) Are these NP-hard?

It is unlikely that any of EOL, $2 -$ Nash, Brouwer, or Sperner is NP-hard then NP = coNP (we proved this for EOL in Theorem 22.3 and the proofs for the others are similar).

In this chapter we will study problems where (1) you know there is a solution, (2) finding it seems hard, but (3) finding it does not seem to be NP-hard.

Exercise 22.13. Show that if the Search Problem that finds *some* satisfying assignment is in polynomial time, then the function that finds the lexicographic least satisfying assignment is in polynomial time.

We will now define an analog of NP for search problems and total search problems.

To get an intuition for FNP let us first define FP. We follow the definition by Rich [Ric08] (from the section *the problem classes FP and FNP*).

Definition 22.14. A relation R is in FP if the following occur.

1. $R(x, y)$ can be computed in polynomial time.

2. The function that, given x, determines if there is a y such that $R(x, y)$ holds, can be computed in polynomial time.

3. The function that, given x such that there is y with $R(x, y)$, finds such a y, can be computed in polynomial time.

Note that if there is such a y it will be of length bounded by a polynomial in $|x|$. (This definition of FP is not the same as the definition of FP in Section 0.1. The definitions are not equivalent. Use the definition presented in this section, in this section.)

Definition 22.15. If $R \in$ FP then ***the set associated to R*** is

$$\{x \mid \exists y : R(x, y)\}.$$

For FP we often think of the set first and the relation later. For example, 2-coloring corresponds to the following relation in FP.

$$\{(G, \rho) \mid \rho \text{ is a 2-coloring of } G\}.$$

FNP will be the NP-analog of FP. We will not demand that the y (if it exists) can be determined. We will demand that the y (if it exists) will be short since that will no longer follow from the definition.

Definition 22.16. A relation R is in FNP if the following occur.

1. $R(x, y)$ can be computed in polynomial time.
2. There is a polynomial p such that, for all x, if there is a y such that $R(x, y)$ then there is also such a y with $|y| \leq p(|x|)$.

Note that we are not claiming that y can easily be found.

Definition 22.17. If $R \in$ FNP then ***the set associated to R*** is

$$\{x \mid \exists y : R(x, y)\}.$$

For FNP we often think of the set first and the relation later. For example, 3-coloring corresponds to the following relation in FNP.

$$\{(G, \rho) \mid \rho \text{ is a 3-coloring of } G\ \}.$$

We want to capture the fact that the problems we care about are total.

Definition 22.18. A relation R is in TFNP if $R \in$ FNP and the following additional property holds: For every x there is a y such that $R(x, y)$. Note again that we are not claiming that such a y can be found easily.

Note: Unlike FP and FNP it would be silly to associate to $R \in$ TFNP a set since that set would always be Σ^*.

22.6 Reductions and PPAD

We define reductions for FNP.

Definition 22.19. Let $R, R' \in$ FNP and let L, L' be the associated sets. R is polynomial-time reducible to R' if the following occur.

1. There exists a polynomial time computable f such that $x \in L$ if and only if $f(x) \in L'$.
2. There exists a polynomial time computable g such that if $R'(f(x), y)$ holds then $R(x, g(y))$ holds. (So if you have a witness for $f(x)$ you can recover one for x.)

Exercise 22.20. Show that, if R is reducible to R' and $R' \in$ FP, then $R \in$ FP.

We could define a notion of FNP-hard based on this reduction. Alas, it is unlikely that EOL, NASH, SPERNER or BROUWER are FNP-hard. We showed in Theorem 22.3 that if EOL is what we now call FNP hard then NP = coNP. The same holds for the other problems.

These four problems all *seem* hard. Lets turn this around! Lets *assume* that one of them is hard and define a complexity class based on that assumption.

Papadimitriou [Pap94] defined the following class.

Definition 22.21. Let $R \in FNP$.

1. R is in PPAD (Polynomial Parity Arguments on Directed graphs) if R is reducible to EOL.

2. R is PPAD-hard if EOL is reducible to R.

3. R is PPAD-complete if R is both in PPAD and PPAD-hard.

Theorem 22.22. *Sperner, Nash, and Brouwer are all in PPAD.*

Proof. The proof of Sperner's Lemma uses Theorem 22.2 in the case of graphs with indegree and outdegree ≤ 1. This proof can be modified to obtain a reduction of Sperner to EOL.

Brouwer's fixed point theorem can be proven from Sperner's Lemma. That proof can be modified to obtain a reduction of Brouwer to Sperner.

The existence of an NE can be proven from Brouwer's fixed point theorem. That proof can be modified to obtain a reduction of Nash to Brouwer. □

It turns out that all three of these problems are PPAD-complete.

We discuss one more problem that is PPAD-complete.

Definition 22.23.

1. Let C be a cake. Let $P_1, \ldots, P_n$ be n people. They each have a utility function that maps areas of the cake to values. The entire cake maps to 1 and a single point maps to 0. If A and B are disjoint parts of the cake then, for any utility function U, $U(A \cup B) = U(A) + U(B)$.

2. An ***allocation*** of C is a partition $C = C_1 \cup \cdots \cup C_n$ of C where, for all $1 \le i \le n$, P_i gets piece C_i.

3. An allocation is ***Proportional*** if every person, using their own utility function, gets $\ge \frac{1}{n}$.

4. An allocation is ***Envy-Free*** if every person, using their own utility function, think that nobody has a strictly larger piece than they have.

Stromquist [Str80] showed that, given any set of n utility functions there exists an envy-free allocation that only uses n cuts. The cuts could be at irrational points. His proof also yielded an algorithm that, given ε, found the cuts to within ε, in time $O(\log \frac{1}{\varepsilon})$. This kind of problem falls neatly into the PPAD paradigm: we have a proof that something exists but we wonder if we can really find it. Deng et al. [DQS12] showed that the problem of finding an approximate envy-free allocation for n people with $n-1$ cuts is PPAD-complete.

22.7 2-Nash is PPAD-Complete

Daskalakis et al. [DGP09] proved that 3-Nash is PPAD-complete (this was actually in Daskalakis's Ph.D. Thesis). At the time it was thought that perhaps the 2-Nash is in P since (1) 2-Nash always has a rational NE and (2) the zero-sum case is in P. Hence it was a surprise when Chen & Deng [CD06] proved that 2-Nash is PPAD-complete. Later Daskalakis et al. [DGP09] found a way to obtain 2-Nash PPAD-complete from their techniques, and that is in their paper.

We will show part of the proof that 2-Nash is PPAD-complete. We mostly follow the approach of Daskalakis et al [DGP09]. Hence all the theorems we present are due to them unless otherwise noted.

The full reduction requires a sequence of reductions:

1. Reduce a generic PPAD problem to a PPAD-type problem in $[0,1]^3$.
2. Reduce the PPAD-type problem to 3D-Sperner problem.
3. Reduce 3D-Sperner problem to Arith Circuit SAT. We will define this define formally this later. It involves arithmetic circuits and asks if there is a way to set the variable nodes so that the circuit is consistent.
4. Reduce Arith Circuit SAT to Poly Matrix Nash. We will define this formally later. It involves a *restricted* Nash problem, though for *many* players. So, initially, it looks incomparable to 2-Nash.
5. Reduce Poly Matrix Nash to 2-Nash.

We will focus on the reduction of Arith Circuit SAT to Poly Matrix Nash.

22.7.1 PPAD-completeness of 2-Nash (High Level)

We first find cycles and paths and place them in a cube $[0,1]^3$ without intersections. Then we represent this as a Sperner problem, next we convert the problem to an Arithmetic Circuit. Finally, that Arithmetic Circuit is converted into a Nash problem.

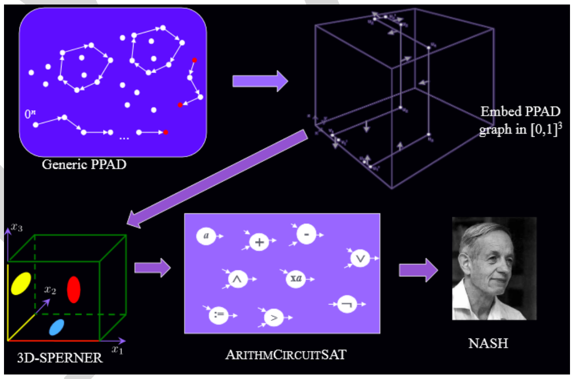


Figure 22.11: A High Level Overview of the Proof that Nash is PPAD-hard

22.7.2 Arithmetic Circuit SAT

An arithmetic circuit is a circuit which has arithmetic Operations at the gates. Normally there would be *input nodes*; however, here we instead have *variable nodes*. The problem will be to find a way to set the variable nodes so that the circuit is consistent.

Definition 22.24. An ***Arithmetic Circuit*** is a circuit with the following types of gates.

- ***Variable nodes*** $x_1, x_2, \ldots, x_n$. These have indegree 1 and outdegree 0 or 1 or 2. (Yes you read that right- the indegree is 1, not 0. These are not input nodes.)

- Gates of 6 types $(:=, +, -, a, xa, >)$ which we describe in the next definition. They are pictured, together with their indegree and outdegree, in Figure 22.12. For the $>$ gate the order of inputs matters, which is why the inputs are labeled 1 and 2. For the other gates the order does not matter.

- Directed edges connecting variables to gates and vice versa.

- Variable nodes have indegree 1; gates have in degree 0,1, or 2 inputs depending on type ; gates and nodes have arbitrary fan out.

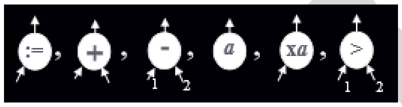


Figure 22.12: The Operation Nodes of ArthmCircuitSAT

Definition 22.25. We use $y \leftarrow f(x_1, \ldots, x_n)$ to mean that if $x_1, \ldots$ are the inputs to the gate, then y is the output.

- Assignment: $y \leftarrow x_1$. Indegree 1.
- Addition: $y \leftarrow \min\{1, x_1 + x_2\}$. Indegree 2.
- Subtraction: $y \leftarrow \max\{0, x_1 - x_2\}$. Indegree 2.
- Equal a constant: $y \leftarrow \max\{0, min\{1, a\}\}$. Indegree 0.
- Multiply by a constant: $y \leftarrow \max\{0, min\{1, ax\}\}$. Indegree 1.
- Greater than:

$$y \leftarrow \begin{cases} 0, & \text{if } x_1 < x_2 \\ 1, & \text{if } x_2 < x_1 \\ \text{any value}, & \text{if } x_2 = x_1 \end{cases} \qquad (22.1)$$

  Indegree 2.

Arith Circuit SAT
Instance: An Arithmetic circuit with variable nodes $x_1, \ldots, x_n$.
Question: Find an assignment of 0's and 1's to the variable nodes so that all of the gate operations are satisfied. In Figure 22.13 we show an example of Arith Circuit SAT.

Node c is assigned to value $1/2$ and thus must have this value.
If $a > c = 1/2$ then b will be set to 0, $a = b$ and $0 < 1/2$ so this is a contradiction.
If $a < c = 1/2$ then b will be set to 1, $a = b$ and $1 > 1/2$ so this is a contradiction.
If $a = c = 1/2$ then b can be set to any value, $a = b = c = 1/2$.

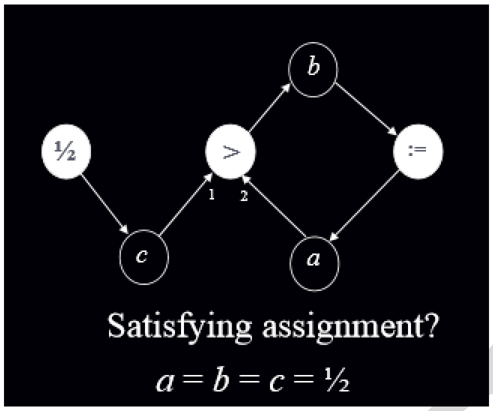


Figure 22.13: The Operation Nodes of Arith Circuit SAT

22.7.3 Graphical Games

Rather than reducing directly to 2-Nash, it will be useful to reduce to a more general kind of game. Kearns et al. [KLS01] defined graphical games.

Definition 22.26. A *graphical game* has the following.

1. A directed graph.

2. Each player's payoff depends only on his own strategy and the strategy of his in-neighbors.

Janovkaya [Jan68] defined a special case of graphical games called polymatrix games.

Definition 22.27. A ***polymatrix game*** is a graphical game with edge-wise separable utility functions. For player v,

$$u_v(x_1, \dots, x_n) = \sum_{(w,v)\in E} u_{w,v}(x_w, x_v) = \sum_{(w,v)\in E} x_v^T A^{(v,w)} x_w.$$

Here, $A^{(v,w)}$ are matrices, x_v is the mixed strategy of v, and x_w is the mixed strategy of w. Now, our strategy for reducing from Arith Circuit SAT to Nash will be via Poly Matrix Nash, so our next goal is to reduce Arith Circuit SAT to Poly Matrix Nash

We define the problem of finding a Nash equilibrium for a polymatrix game.

Poly Matrix Nash *Instance:* A polymatrix game. *Question:* Find an approximation to a Nash equilibrium.

22.7.4 Reduction: Arith Circuit SAT to Poly Matrix Nash

The key to this reduction is the invention of *game gadgets*, small polymatrix games that model arithmetic at their Nash equilibrium. As an example, consider the following gadget for addition.

Addition Gadget Suppose each player has two strategies, call these $\{0, 1\}$. Then, a mixed strategy is a number in $[0, 1]$, i.e. the probability of playing 1. We construct a gadget with players w, x, y, z and edges (x, w), (y, w), (z, w), (w, z). Player w is paid expected:

$$\begin{aligned}&\Pr[x:1] + \Pr[y:1] \text{ for playing } 0\\ &\Pr[z:1] \text{ for playing } 1\end{aligned}$$

It is easy to construct this payoff matrix: $u_w(0) = x + y$ and $u_w(1) = z$.
Player z is paid for "playing the opposite" of w: $u_z(0) = .5$ and $u_z(1) = 1 - w$.

Claim: In any Nash equilibrium of a game containing the above gadget, $\Pr[z:1] = min\{\Pr[x:1] + \Pr[y:1], 1\}$.

Proof. Suppose $\Pr[z:1] < min(\Pr[x:1] + \Pr[y:1], 1)$. Then w will play 0 with probability 1, but then z should have played 1 with probability 1, a contradiction.

Conversely, suppose $\Pr[z:1] > \Pr[x:1] + \Pr[y:1]$. Then w will play 1 with probability 1, but then z should have played 0 with probability 1, again a contradiction.

Thus, $\Pr[z:1] = \Pr[x:1] + \Pr[y:1]$. □

More gadgets We need such a gadget for all the possible gates in Arith Circuit SAT. If z is the output of the gadget and x, y are the inputs, we need (conflating x with $\Pr[x:1]$)

- copy: $z = x$
- addition: $z = min(1, x + y)$
- subtraction: $z = max(0, x - y)$
- set equal to constant: $z = a$
- multiply by a constant: $z = ax$
- comparison: $z = 1$ if $x > y$, $z = 0$ if $x < y$, unconstrained otherwise

Another example: Comparison gadget Players x, y, z. $u_z(0) = y$, $u_z(1) = x$. Then $x > y \Rightarrow \Pr[z:1] = 1$ and similarly for $x < y$.

Now that we have all these gadgets we can sketch a proof of the reduction we seek (in this section).

Theorem 22.28. *Arith Circuit SAT is reducible to Poly Matrix Nash. Hence Poly Matrix Nash is PPAD-complete.*

Proof sketch: We omit the construction of the remaining gadgets, but they are not much more complicated than the addition gadget. From here, given an arbitrary instance of Arith Circuit SAT, we can create a polymatrix game by composing game gadgets corresponding to each of the gates. At any Nash equilibrium of the resulting polymatrix game, the gate conditions are satisfied, which completes the reduction.

By Theorem 22.22 Nash is in PPAD. Since Poly Matrix Nash is a subcase of Nash, it is also in PPAD.

∎

22.7.5 Reduction: Poly Matrix Nash to 2-Nash

Theorem 22.29. *Poly Matrix Nash reduces to 2-Nash.*

Proof. Since we can construct the game gadgets we used to reduce to Poly Matrix Nash using only bipartite game gadgets (input and output players are on one side, and auxiliary nodes are on the other side), without any loss of generality we can also assume the polymatrix game is bipartite. This implies that the graph is 2-colorable, say by two colors 'red' and 'blue'. Now, we can think of this as a two-player game between the 'red lawyer' and the 'blue lawyer', where each lawyer represents all the nodes of their color.

Each lawyer's set of pure strategies is the union of the pure strategy sets of her clients; importantly, this is not the same as having a strategy set equal to the product of the strategy sets of the clients, as this would cause an exponential blowup in the problem size (and thus invalidate the reduction). One way of picturing this is that the red lawyer selects one of her clients to represent, and then selects a strategy for that client.

The payoff of $(u : i), (v : j)$ (the red lawyer plays strategy i of client u, and the blue lawyer plays strategy j of client v) is $A_{i,j}^{(u,v)}$ for u and $A_{j,i}^{(v,u)}$ for v. Informally, the payoff of the lawyers is just the payoffs of the respective clients had they played the chosen strategies. Also, note that the payoff is 0 for either lawyer if their client choice doesn't have an edge incoming from the other lawyer's client choice (and 0 for both lawyers if they choose non-neighboring clients).

Wishful thinking If (x, y) is a Nash equilibrium of the lawyer game, then the marginal distributions that x assigns to the strategies of the red nodes and the marginals that y assigns to the blue nodes comprise a Nash equilibrium.

This doesn't work because lawyers might gravitate only to the 'lucrative' clients - we need to enforce that lawyers will (at least approximately) represent each client equally. This is true not only because a lawyer might choose to *never* represent a client (and hence the marginals are undefined), but because if the red lawyer's clients are not equally represented, the optimal marginals for the blue lawyer are distorted correspondingly.

Enforcing Equal Representation Lawyers play a 'high stakes' game on the side. Without loss of generality, assume that each lawyer represents n clients (can create 'dummy' clients to equalize number of clients). Label each lawyer's clients $1, \ldots, n$ arbitrarily. Payoffs of the high stakes game: Suppose the red lawyer plays any strategy of client j and blue lawyer plays any strategy of client k, then if $j \neq k$ then both players get 0. If $j = k$ then red lawyer gets $+M$ while blue lawyer gets $-M$.

Claim: In any Nash equilibrium of the high stakes game, each lawyer assigns probability (close to) $1/n$ to the set of pure strategies of each of his clients.

Now, the game we need for the reduction is simply the sum of the original lawyer game and the high stakes game. Choose $M >> 2n^2 u_{max}$, where u_{max} is the maximum absolute value of payoffs in the original game. We can show that the total probability mass is distributed almost evenly among the different nodes.

Lemma 22.30. If (x, y) is an equilibrium of the lawyer game, for all u, v:

$$x_u = \frac{1}{n}\left(1 \pm \frac{2u_{max}n^2}{M}\right)$$

$$y_v = \frac{1}{n}\left(1 \pm \frac{2u_{max}n^2}{M}\right)$$

However, within a particular node u, only the original low stakes game matters, since the different strategies of u are all identical from the perspective of the high stakes game.

Lemma 22.31. The payoff difference for the red lawyer from strategies $(u : i)$ and $(u : j)$ is

$$\sum_v \sum_\ell (A_{i,\ell}^{(u,v)} - A_{j,\ell}^{(u,v)}) y_{v:\ell}$$

Corollary If $x_{u:i} > 0$, then for all j:

$$\sum_v \sum_\ell (A_{i,\ell}^{(u,v)} - A_{j,\ell}^{(u,v)}) y_{v:\ell} \geq 0$$

Define $\hat{x}_u(i) := \frac{x_{u:i}}{x_u}$ and $\hat{y}_v(j) := \frac{y_{v:j}}{y_v}$ (these are the marginals given by lawyers to different nodes).

□

Observation: If we had $x_u = 1/n$ for all u and $y_v = 1/n$ for all v, then $\{\{\hat{x}_u\}_u, \{\hat{y}_v\}_v\}$ would be a Nash equilibrium. Since we have $\pm\frac{2u_{max}n^2}{M}$ deviation, we get approximate Nash equilibrium instead. Fortunately, Arith Circuit SAT is still PPAD-hard with ε approximation, so we still get PPAD-hardness for Nash from this reduction.

The Nash really asks for an approximation to the NE. This means that if (x, y) is the NE (where x and y are vectors of probabilities that add to 1) then the algorithm produces (x', y') where x' is close to x and y' is close to y. We briefly discuss a different kind of approximation.

Definition 22.32. An *ε-Nash equilibrium (henceforth just ε-equilibrium*) is a pair of mixed strategies (x, y) such that the following holds.

1. If the row player deviates from x, and the column player still uses y, then the row player benefits by at most ε.

2. If the column player deviates from y, and the row player still uses x, then the column player benefits by at most ε.

3. For each player, the payoff at (x, y) is at most ε less than the optimal.

There are essentially matching upper and lower bounds for the time needed to find an ε-equilibrium:

Theorem 22.33.

1. *Lipton et al. [LMM03] showed that, for all $\varepsilon > 0$, there is an algorithm that finds an ε-equilibrium that runs in time $O(n^{\varepsilon^{-2}\log n})$*

2. *Braverman et al. [BKW15] showed that, assuming ETH, there exists ε^* such that any algorithm that finds an ε^*-equilibrium and requires time $O(n^{\log n})$*

Remark: Consider a 2-player game with payoff matrices R & C. Zero-sum games correspond to having $R + C \equiv 0$, but we can also consider games where the rank of $R + C$ is r, for some constant r. It is known that rank 1 games have a polynomial-time algorithm for finding a Nash equilibrium (just like zero-sum games), but a result of Mehta in [Meh14] shows that, in rank 3 games, it is already PPAD-hard to find a Nash equilibrium.

22.8 Other arguments of existence and resulting complexity classes

The purely combinatorial theorem with a nonconstructive proof at the core of the definition of PPAD was

If a directed graph has an unbalanced vertex, then it has another unbalanced vertex.

This statement lead to problems (EOL, Nash, Brouwer, Sperner) that are in P but finding the witness seems hard. We then defined PPAD to pin down that finding the witness is probably hard.

In this section we discuss other purely combinatorial theorems with nonconstructive proofs, the problems they put in P where witnesses seem hard to find, and the complexity classes they inspired.

22.8.1 There are an Even Number of Vertices of Odd Degree

The oldest theorem in graph theory, due to Euler, is the following (we refashion it for our purposes).

Theorem 22.34. *If a graph has a node of odd degree, then it must have another.*

The proof of Theorem 22.34 is nonconstructive.

The following theorem is due to Smith (unpublished) and is presented in a paper by Thomason [Tho78] (see also a paper by Krawczyk [Kra99] which contains the proof and is not behind paywalls).

Theorem 22.35. *If a 3-regular graph has a Hamiltonian cycle C then, for all edges e in C, there is a second Hamiltonian cycle that uses e.*

The proof uses Theorem 22.34 and hence is nonconstructive.

We use Theorem 22.34 and 22.35 as the inspiration for problems.

> ODDDEG
> *Instance:* A circuit C on $\{0,1\}^n$ that, on input x, outputs a set of elements of $\{0,1\}^n$. We interpret this as a graph on $\{0,1\}^n$ where the circuit outputs a vertices neighbors. We also have that 0^n has an odd number of neighbors.
> *Question:* Find another vertex of odd degree.

> HAMILTONIAN CYCLES IN LARGE 3-REGULAR GRAPHS (HAM-3REG)
> *Instance:* A 3-regular graph G and a Hamiltonian cycle H in G.
> *Question:* Find another Hamiltonian cycle in G.

It is believed that both ODDDEG and HAM-3REG are hard and are equivalent. Papadimitriou[Pap94] defined the following classes to make this statement rigorous (our definition differs from Papadimitriou, however ours and his definitions are equivalent).

Definition 22.36. Let $R \in FNP$.

1. R is in PPA (Polynomial Parity Argument) if R is reducible to ODDDEG.
2. R is PPA-hard if ODDDEG is reducible to R.
3. R is PPA-complete if R is both in PPA and PPA-hard.

Exercise 22.37. Show that PPAD $\subseteq$ PPA.

The proof of Theorem 22.35 easily yields the following theorem.

Theorem 22.38. *HAM-3REG is in PPA.*

It is an open problem to determine whether HAM-3REG is PPA-complete.

Here is a non-graph example.

Definition 22.39. Let D be any integral domain (for our purposes $\mathbb{Z}$ or $\mathbb{Z}_p$).

1. Let m be a monomial in $D[x_1,\ldots,x_n]$. The *degree* of m is the sum of the degrees of its terms. For example, the degree of $x_1^2x_2^3x_4^4$ is $2+3+4=9$.
2. Let $p \in D[x_1,\ldots,x_n]$ The *degree* of p is the max of the degrees of the monomials in p.
3. If $q_1,\ldots,q_L \in D[x_1,\ldots,x_n]$ then we refer to that set of polynomials as *a system*. A *solution to the system* is $(a_1,\ldots,a_nD^n$ such thaof the polynomials are 0 on $(a_1,\ldots,a_n)$.

Chevalley proved the following.

Theorem 22.40. *Let p be a prime. Let $q_1(x_1,\ldots,x_n)$,..., $q_L(x_1,\ldots,x_n) \in \mathbb{Z}_p[x_1,\ldots,x_n]$. Assume q_i is of degree d_i.*

1. *If $\sum_{i=1}^{L} d_i < n$ then the number of solutions to this system is divisible by p.*

2. *(This is an easy corollary of interest to us.) If $p = 2$ and there is a solution, then there is another solution.*

> CHEVALLEY
> *Instance:* A system of polynomial equations with n variables over $\mathbb{Z}_2$ such that the sum of the degrees is $< n$, and one solution.
> *Question:* Find another solution.

Papadimitriou [Pap94] proved the following.

Theorem 22.41. *CHEVALLEY is in PPA.*

It is an open problem to determine whether CHEVALLEY is PPA-complete. The following is known.

1. Goos et al. [GKSZ20] show that a variant of CHEVALLEY is PPA_q-complete (you will define PPA_q in Exercise 22.42). However, they do not think the original CHEVALLEY is PPA-complete (see their note on page 6).

Exercise 22.42.

1. Let $q \in \mathbb{N}$ and let G be a bipartite graph. Show that if there is some vertex of degree $\not\equiv 0 \pmod{q}$ then there must be another one .

2. Define PPA_q and PPA_q-complete using Part 1 as motivation.

3. Read Goos et al. [GKSZ20] which shows several problems are PPA_q-complete. Rewrite their proofs in your own words.

What about natural problems and completeness? The following are known.

1. Filos-Ratsikas & Goldberg [FG18] showed that the consensus-halfing problem, a computational version of the Hobby-Rice Theorem, is PPA-complete.

2. Jerábek [Jer16] showed that the following problems are in PPA (1) finding square roots mod n, (2) finding quadratic residues mod n. Note that this is not a hardness result. He also showed that integer factoring (finding a factor) is randomly reducible to a problem in PPA. Under the General Riemann Hypothesis there is a deterministic reduction. Note that this reduction, randomized or deterministic, is not a hardness result.

22.8.2 Every Directed Acyclic Graph has a Sink

The following is well known.

Theorem 22.43. *Every directed acyclic graph has a vertex v such that $outdeg(v) = 0$. Such a node is called a* sink.

The proof is nonconstructive. Theorem 22.43 inspires the following problem.

> FindSink
> *Instance:* A circuit C on $\{0,1\}^n$ that, on input x, outputs a set of elements of $\{0,1\}^n$. We interpret this as a graph on $\{0,1\}^n$ where the circuit outputs a *potential* set of neighbors. The neighbors of v are the elements $u \in C(v)$ such that $u > v$ (interpreted as numbers in binary). This will ensure that the graph is acyclic.
> *Question:* Find a sink.

Johnson et al. [JPY88] defined the following.

Definition 22.44. Let $R \in FNP$.

1. R is in PLS (Polynomial Local Search) if R reduces to FindSink. (The name "Polynomial Local Search" comes from using this class to classify certain search problems that have local max (or min) making it difficult to find the global max (or min).)

2. R is PLS-hard if FindSink is reducible to R.

3. R is PLS-complete if R is both in PLS and PLS-hard.

> LocalMaxCut
> *Instance:* A weighted graph $G = (V, E, w)$.
> *Question:* Find a partition $V = V_1 \cup V_2$ that is locally optimal (i.e. can't move any single vertex to the other side to increase the cut size).

Johnson et al. [JPY88] showed the following.

Theorem 22.45. *LocalMaxCut is PLS-complete.*

22.8.3 The Pigeonhole Principle

The following is the well known Pigeonhole Principle.

Theorem 22.46. *Let A have n elements and B have $n-1$ elements. For all functions $f : A \longrightarrow B$ there exists $x_1 \neq x_2 \in A$ such that $f(x_1) = f(x_2)$.*

The proof of Theorem 22.46 is nonconstructive.

The following theorem is an easy direct consequence of Theorem 22.46 and hence its proof is nonconstructive.

Theorem 22.47. *If $A \subseteq \{1, \ldots, 2^n - 1\}$ of n elements whose sum is $< 2^n - 1$ then there exist two distinct subsets of A that have the same sum.*

We use Theorem 22.46 and 22.47 as the inspiration for problems.

> Collison
> *Instance:* A circuit C with input and output both $\{0,1\}^n$. (We are not requiring that C has range $< 2^n$.)
> *Question:* Find either an x such that $C(x) = 0^n$ or an x, y such that $x \neq y$ but $C(x) = C(y)$.

Distinct Subsets (DistSubset)
Instance: $A \subseteq \{1, \ldots, 2^n - 1\}$ of n elements whose sum is $< 2^n - 1$.
Question: Find two distinct subsets of A with the same sum.

It is believed that both Collison and DistSubset are hard and are equivalent. Papadimitriou[Pap94] defined the following classes to make this statement rigorous.

Definition 22.48. Let $R \in FNP$.

1. R is in PPP (Polynomial Pigeonhole Principle) if R reduces to Collison.
2. R is PPP-hard if Collison is reducible to R.
3. R is PPP-complete if R is both in PPP and PPP-hard.

Exercise 22.49. Show that PPAD $\subseteq$ PPP.

The proof of Theorem 22.47 easily yields the following theorem.

Theorem 22.50. *DistSubset is in PPP.*

It is an open problem to determine whether DistSubset is PPP-complete.
What about natural problems? The following is known.

1. Sotiraki et al. [SZZ18] showed that a variant of the shortest lattice problem is PPP-complete.
2. Jerábek [Jer16] showed that there is a randomized reduction from integer factorization (finding a nontrivial factor) to a weaker version of Collison where the domain is 2^n and the range is 2^{n-1}. Under the Generalized Riemann Hypothesis the reduction can be derandomized. Note that this reduction, randomized or deterministic, is not a hardness-result for factoring. Nor does it imply that factoring is in PPP.

22.9 How do the Classes Relate?

We summarize what is know about how the classes relate, and what is open.

Exercise 22.51.

1. Show that FP $\subseteq$ PPAD $\subseteq$ PPA $\subseteq$ FNP.
2. Show that FP $\subseteq$ PPAD $\subseteq$ PPP $\subseteq$ FNP.
3. (Open problem) For each subset inclusions in Part 1 and 2 resolve if the inclusion is equal or proper. (It is widely believed that all of the inclusions are proper.)
4. (Open problem) For each subset inclusions in Part 1 and 2 determine whether an equality implies P = NP or some other unlikely conclusion.
5. (Open Problem) Resolve how PPA and PPP compare.

Bibliography

[AAGH21] Pankaj K. Agarwal, Boris Aronov, Tzvika Geft, and Dan Halperin. On two-handed planar assembly partitioning with connectivity constraints. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1740–1756. SIAM, 2021. https://doi.org/10.1137/1.9781611976465.105.

[ABC+20] Zachary Abel, Jeffrey Bosboom, Michael J. Coulombe, Erik D. Demaine, Linus Hamilton, Adam Hesterberg, Justin Kopinsky, Jayson Lynch, Mikhail Rudoy, and Clemens Thielen. Who witnesses the witness? finding witnesses in the witness is hard and sometimes impossible. *Theoretical Computer Science*, 839:41–102, 2020. https://doi.org/10.1016/j.tcs.2020.05.031.

[ABD+04] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S. B. Mitchell, Saurabh Sethia, and Steven Skiena. When can you fold a map? *Compututational Geometry*, 29(1):23–46, 2004. https://doi.org/10.1016/j.comgeo.2004.03.012.

[ABF+02] Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for DOMINATING SET and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002. https://doi.org/10.1007/s00453-001-0116-5.

[ABGS21] Divesh Aggarwal, Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. Fine-grained hardness of CVP(P) - everything that we can prove (and nothing else). In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1816–1835. SIAM, 2021. https://doi.org/10.1137/1.9781611976465.109.

[ABI+09] Eric Allender, Michael Bauland, Neil Immerman, Henning Schnoor, and Heribert Vollmer. The complexity of satisfiability problems: Refining schaefer's theorem. *Journal of Computing and System Science*, 75(4):245–254, 2009. https://www.cs.rutgers.edu/allender/papers/csp.pdf.

[ABS15] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the Association of Computing Machinery (JACM)*, 62(5):42:1–42:25, 2015. https://doi.org/10.1145/2775105.

[ABSS97] Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computing and System Sciences*, 54(2):317–331, 1997. https://doi.org/10.1006/jcss.1997.1472.

[ABV15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. https://doi.org/10.1109/FOCS.2015.14.

[AC06] Sanjeev Arora and Eden Chlamtac. New approximation guarantee for chromatic number. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 215–224. ACM, 2006. https://doi.org/10.1145/1132516.1132548.

[ACMM05] Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. $O((\sqrt{\log n})$ approximation algorithms for min uncut, min 2CNF deletion, and directed cut problems. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 573–581. ACM, 2005. https://doi.org/10.1145/1060590.1060675.

[AD11] Zachary Abel and Erik D. Demaine. Edge-unfolding orthogonal polyhedra is strongly NP-complete. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12, 2011*, 2011. http://www.cccg.ca/proceedings/2011/papers/paper43.pdf.

[AD16] Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486. IEEE Computer Society, 2016. https://doi.org/10.1109/FOCS.2016.58.

[ADD+14] Zachary Abel, Erik D. Demaine, Martin L. Demaine, Takashi Horiyama, and Ryuhei Uehara. Computational complexity of piano-hinged dissections. *IEICE Transactions on Fundamentals of Electronics, Communicationsand Computer Science.*, 97-A(6):1206–1212, 2014. https://doi.org/10.1587/transfun.E97.A.1206.

[ADG14] Greg Aloupis, Erik D. Demaine, and Alan Guo. Classic Nintendo games are NP-hard. In *Fun with Algorithms*, volume abs/1203.1895, 2014. http://arxiv.org/abs/1203.1895.

[ADGV15] Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015. https://doi.org/10.1016/j.tcs.2015.02.037.

[ADHL22] Joshua Ani, Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. In Petra Mutzel, Md. Saidur Rahman, and Slamin, editors, *WALCOM: Algorithms and Computation - 16th International Conference and Workshops, WALCOM 2022, Jember, Indonesia, March 24-26, 2022, Proceedings*, volume 13174 of *Lecture Notes in Computer Science*, pages 187–198. Springer, 2022. https://arxiv.org/abs/2005.03192.

[Adi55] S. I. Adian. Algorithmic unsolvability of problems of recognition of certain properties in groups(in russian). *Doklady Akademii Nauk SSSR*, 103:533–535, 1955.

[AFI+09] Esther M. Arkin, Sándor P. Fekete, Kamrul Islam, Henk Meijer, Joseph S. B. Mitchell, Yurai Núñez-Rodrıguez, Valentin Polishchuk, David Rappaport, and Henry Xiao. Not being (super)thin or solid is hard: A study of grid hamiltonicity. *Computational Geometry: Theory and Applications*, 42(6):582–605, 2009.

[AFM00] Esther M. Arkin, Sándor P. Fekete, and Josephy S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1):25–50, 2000. http://www.ams.sunysb.edu/jsbm/papers/lawn.pdf.

[AGI+19] Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. https://doi.org/10.4230/LIPIcs.ICALP.2019.7.

[AGLar] Daniel Apon, William Gasarch, and Kevin Lawler. An NP-complete problem in grid coloring. *Theory of Computing Systems*, 2023 (to appear). http://arxiv.org/abs/1205.3813.

[AGV15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697. SIAM, 2015. https://doi.org/10.1137/1.9781611973730.112.

[AH77a] Kenneth Appel and Wolfgang Haken. Every planar map is four colorable I: Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977. https://projecteuclid.org/euclid.ijm/1256049011.

[AH77b] Kenneth Appel and Wolfgang Haken. Solution of the four color map problem. *Scientific American*, 237:108–121, 1977.

[AH08] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
https://doi.org/10.1137/060669474.

[AH19] Eric Allender and Shuichi Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. *ACM Trans. Comput. Theory*, 11(4):27:1–27:27, 2019.
https://doi.org/10.1145/3349616.

[AHK77] Kenneth Appel, Wolfgang Haken, and John Koch. Every planar map is four colorable II: Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.
https://projecteuclid.org/euclid.ijm/1256049012.

[Ajt98] Miklós Ajtai. The shortest vector problem in l_2 is *NP*-hard for randomized reductions (extended abstract). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 10–19. ACM, 1998.
https://doi.org/10.1145/276698.276705.

[AK95] Edoardo Amaldi and Viggo Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theor. Comput. Sci.*, 147(1&2):181–210, 1995.
https://doi.org/10.1016/0304-3975(94)00254-G.

[AKI87] H. Adachi, H. Kamekawa, and S. Iwata. Shogi on $n \times n$ board is complete in exponential time. *Transactions IEICE. J70-D*, 88-A:1843–1852, 1987.

[AKS87] Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 132–140. ACM, 1987.
https://doi.org/10.1145/28395.28410.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES in p. *Annals of Mathematics*, 160:781–793, 2004.
http://www.cs.umd.edu/gasarch/BLOGPAPERS/primesinP.pdf.

[AL10] Antonios Antoniadis and Andrzej Lingas. Approximability of edge matching puzzles. In *SOFSEM 2010: Theory and Practice of Computer Science, 36th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 23-29, 2010. Proceedings*, pages 153–164, 2010.

[Alf98] Jorge L. Ramírez Alfonsín. On variations of the subset sum problem. *Discret. Appl. Math.*, 81(1-3):1–7, 1998.
https://core.ac.uk/download/pdf/82533986.pdf.

[All20] Eric Allender. The new complexity landscape around circuit minimization. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors,

Language and Automata Theory and Applications - 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings, volume 12038 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2020.
https://people.cs.rutgers.edu/allender/papers/jones.pdf.

[ALM⁺98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.
https://doi.org/10.1145/278298.278306.

[ALN05] Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 553–562. ACM, 2005.
https://doi.org/10.1145/1060590.1060673.

[AM17] Josh Alman and Dylan McKay. Theoretical foundations of team matchmaking. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1073–1081. ACM, 2017.
http://dl.acm.org/citation.cfm?id=3091277.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Science*, 58(1):137–147, 1999.
https://doi.org/10.1006/jcss.1997.1545.

[AN21] Sepehr Assadi and Vishvajeet N. Graph streaming lower bounds for parameter estimation and property testing via a streaming XOR lemma. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 612–625. ACM, 2021.
https://doi.org/10.1145/3406325.3451110.

[And09] Daniel Andersson. Hashiwokakero is NP-complete. *Information Processing Letters*, 109(19):1145–1146, 2009.

[AR98] Yonatan Aumann and Yuval Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.
https://doi.org/10.1137/S0097539794285983.

[AR20] Sepehr Assadi and Ran Raz. Near-quadratic lower bounds for two-pass graph streaming algorithms. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 342–353. IEEE, 2020.
https://doi.org/10.1109/FOCS46700.2020.00040.

[Aro07] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the Association of Computing Machinery (JACM)*, 45(3):501–555, 2007.
https://dl.acm.org/doi/10.1145/290179.290180.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009.
https://doi.org/10.1145/1502793.1502794.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.
https://doi.org/10.1145/273865.273901.

[ASS+18] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685. IEEE Computer Society, 2018.
https://doi.org/10.1109/FOCS.2018.00070.

[Ass21] Sepehr Assadi. A two-pass lower bound for semi-streaming maximum matching, 2021.
https://arxiv.org/abs/2108.07187.

[Aus07] Per Austrin. Balanced max 2-sat might not be the hardest. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 189–197. ACM, 2007. https://doi.org/10.1145/1250790.1250818.

[AV14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014.
https://doi.org/10.1109/FOCS.2014.53.

[AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014.

[AVY18] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018.
https://doi.org/10.1137/15M1050987.

[AWY15] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. https://doi.org/10.1137/1.9781611973730.17.

[BA18] Moti Ben-Ari. Mastermind is NP-complete, 2018. http://arxiv.org/abs/cs/0512049.

[Bab16] Laszlo Babai. Graph isomorphism is in quasipolynomial time, 2016. https://arxiv.org/abs/1512.03547.

[Bak68] Alan Baker. On the representation of integers by binary forms. *Philosophical Transactions of the Royal Society of London*, 263:173–191, 1968.

[Bar18] Boaz Barak. The Unique game conjecture–halfway there?, 2018. https://windowsontheory.org/2018/01/10/unique-games-conjecture-halfway-there/.

[BB02] Olivier Bournez and Michael S. Branicky. The mortality problem for matrices of low dimensions. *Theory Comput. Syst.*, 35(4):433–448, 2002. https://doi.org/10.1007/s00224-002-1010-5.

[BBD+17] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, Mohammad-Taghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity clustering: Hierarchical clustering at scale. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6864–6874, 2017. https://proceedings.neurips.cc/paper/2017/hash/2e1b24a664f5e9c18f407b2f9c73e821-Abstract.html.

[BBK+16] Kevin Buchin, Maike Buchin, Maximilan Konzack, Wolfgang Mulzer, and Andre Schulz. Fine-grained analysis of problems on curves. In Avrim Blum, editor, *32nd European workshop on Computational Geometry (EuroCG)*, 2016. https://page.mi.fu-berlin.de/mulzer/pubs/kfrechetEWCG.pdf.

[BCC+14] Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R. Fellows, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. *J. Comput. Syst. Sci.*, 80(7):1285–1297, 2014. https://doi.org/10.1016/j.jcss.2014.03.001.

[BCD+18] Jeffrey Bosboom, Spencer Congero, Erik D. Demaine, Martin L. Demaine, and Jayson Lynch. Losing at checkers is hard, 2018. http://arxiv.org/abs/1806.05657.

[BCG82] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, New York, 1982.

[BCH16] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes Theoretical Computer Science*, 322:51–67, 2016.
https://doi.org/10.1016/j.entcs.2016.03.005.

[BCLR96] Valentin E. Brimkov, Bruno Codenotti, Mauro Leoncini, and Giovanni Resta. Strong NP-completeness of a matrix similarity problem. *Theor. Comput. Sci.*, 165(2):483–490, 1996.
https://doi.org/10.1016/0304-3975(96)00103-X.

[BCLS87] T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7:171–191, 1987.

[BDD+01] Therese C. Biedl, Erik D. Demaine, Martin L. Demaine, Rudolf Fleischer, Lars Jacobsen, and J. Ian Munro. The complexity of clickomania, 2001.
https://arxiv.org/abs/cs/0107031.

[BDD+17] Jeffrey Bosboom, Erik D. Demaine, Martin L. Demaine, Adam Hesterberg, Pasin Manurangsi, and Anak Yodpinyanee. Even $1 \times n$ edge-matching and jigsaw puzzles are really hard. *Journal of Information Processing*, 25:682–694, 2017.
https://doi.org/10.2197/ipsjjip.25.682.

[BDE+19a] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab S. Mirrokni. Near-optimal massively parallel graph connectivity. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1615–1636. IEEE Computer Society, 2019.
https://doi.org/10.1109/FOCS.2019.00095.

[BDE+19b] Mahdi Boroujeni, Sina Dehghani, Soheil Ehsani, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Subcubic equivalences between graph centrality measures and complementary problems, 2019.
http://arxiv.org/abs/1905.08127.

[BDE+21] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. Massively parallel computation via remote memory access. *ACM Trans. Parallel Comput.*, 8(3):13:1–13:25, 2021.
https://doi.org/10.1145/3470631.

[BDH+04] Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14(1-2):41–68, 2004.
https://doi.org/10.1142/S0218195904001354.

[BDP08] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008.
https://doi.org/10.1007/s00453-007-9036-3.

[BdW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.
https://www.sciencedirect.com/science/article/pii/S030439750100144X.

[BEP05] Cristina Bazgan, Bruno Escoffier, and Vangelis Th. Paschos. Completeness in standard and differential approximation classes: Poly-(D)APX- and (D)PTAS-completeness. *Theoretical Computer Science*, 339(2-3):272–292, 2005.
http://dx.doi.org/10.1016/j.tcs.2005.03.007.

[Ber66] Robert Berger. *The undecidability of the domino problem*. Memoirs of the American Math Society. American Math Society, 1966.

[BF13] Karl Bringmann and Tobias Friedrich. Parameterized average-case complexity of the hypervolume indicator. In Christian Blum and Enrique Alba, editors, *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, pages 575–582. ACM, 2013.
https://doi.org/10.1145/2463372.2463450.

[BGL+18] Davide Bilò, Luciano Gualà, Stefano Leucci, Guido Proietti, and Mirko Rossi. On the PSPACE-completeness of peg duotaire and other peg-jumping games. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy*, volume 100 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
https://doi.org/10.4230/LIPIcs.FUN.2018.8.

[BGLR93] Mihir Bellare, Shafi Goldwasser, Carsten Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 294–304. ACM, 1993.
https://doi.org/10.1145/167088.167174.

[BGMW20] Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). *ACM Trans. Algorithms*, 16(4):48:1–48:22, 2020.
https://doi.org/10.1145/3381878.

[BGRS13] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1):6:1–6:33, 2013.
https://doi.org/10.1145/2432622.2432628.

[BGS98] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps, and nonapproximability-towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.
https://www.cs.umd.edu/gasarch/TOPICS/pcp/bgsfreebits.pdf.

[BGSD20] Huck Bennett, Sasha Golovnev, and Noah Stephens-Davidowitz. Fine-grained hardness of lattice problems: Open questions, 2020. This is on the Simons Institute Website.

[BH01] Gill Barequet and Sariel Har-Peled. Polygon containment and translational min-hausdorff-distance between segment sets are 3sum-hard. *Int. J. Comput. Geom. Appl.*, 11(4):465–474, 2001.
https://doi.org/10.1142/S0218195901000596.

[BHP12] Paul C. Bell, Mika Hirvensalo, and Igor Potapov. Mortality for 2×2 matrices is NP-hard. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 148–159. Springer, 2012.

[BHZ87] Ravi Boppana, Johan Hastad, and Stathis Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25, 1987.

[BI06] Jonathan F. Buss and Tarique Islam. Simplifying the weft hierarchy. *Theoretical Computer Science*, 351(3):303–313, 2006.
https://doi.org/10.1016/j.tcs.2005.10.002.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015.
https://doi.org/10.1145/2746539.2746612.

[BI16] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466. IEEE Computer Society, 2016.
https://doi.org/10.1109/FOCS.2016.56.

[Bid20] Stella Biderman. Magic: the gathering is as hard as arithmetic, 2020.
https://arxiv.org/abs/2003.05119.

[BJ04] Aleksander Bachman and Adam Janiak. Scheduling jobs with position-dependent processing times. *J. Oper. Res. Soc.*, 55(3):257–264, 2004.
https://doi.org/10.1057/palgrave.jors.2601689.

[BK98] Therese C. Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. *Computational Geometry: Theory and Applications*, 9(3):159–180, 1998.

[BK99] Piotr Berman and Marek Karpinski. On some tighter inapproximability results. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, pages 200–209, 1999.

[BK09] Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 453–462. IEEE Computer Society, 2009.
https://doi.org/10.1109/FOCS.2009.23.

[BK10] Nikhil Bansal and Subhash Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 250–261. Springer, 2010.

[BKP$^+$19] Marthe Bonamy, Lukasz Kowalik, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. Tight lower bounds for the complexity of multicoloring. *ACM Trans. Comput. Theory*, 11(3):13:1–13:19, 2019.
https://doi.org/10.1145/3313906.

[BKS17] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the Assocation of Computing Machinery*, 64(6):40:1–40:58, 2017.
https://doi.org/10.1145/3125644.

[BKW15] Mark Braverman, Young Kun-Ko, and Omri Weinstein. Approximating the best nash equilibrium in $n^{o(\log n)}$-time breaks the exponential time hypothesis. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 970–982. SIAM, 2015.
https://doi.org/10.1137/1.9781611973730.66.

[BM08] Benjamin E. Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.
https://doi.org/10.1145/1360443.1360462.

[BM20] Édouard Bonnet and Tillmann Miltzow. Parameterized hardness of art gallery problems. *ACM Trans. Algorithms*, 16(4):42:1–42:23, 2020.
https://doi.org/10.1145/3398684.

[Boo58] William Boone. The word problem. *Proceedings of the National Academy of Sciences*, 44:1061–1065, 1958.

[BOS94] David Bremner, Joseph O'Rourke, and Thomas Shermer. Motion planning amidst movable square blocks is PSPACE-complete, 1994. Not published and does not seem to be online, hence is lost to humanity.

[Bou85] Jean Bourgain. On Lipshitz embeddings of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[BP15] Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *Journal of the Association of Computing Machinery*, 62(3):19:1–19:52, 2015. https://arxiv.org/pdf/1011.2894.pdf.

[BP22] Huck Bennett and Chris Peikert. Hardness of the (approximate) shortest vector problem: A simple proof via Reed-Solomon codes, 2022. https://arxiv.org/abs/2202.07736.

[BPT92] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6):555–581, 1992. https://doi.org/10.1007/BF01758777.

[BRG89] David Bernstein, Michael Rodeh, and Izidor Gertner. On the complexity of scheduling problems for parallel/pipelined machines. *IEEE Trans. Computers*, 38(9):1308–1313, 1989. https://doi.org/10.1109/12.29469.

[Bri14] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. https://doi.org/10.1109/FOCS.2014.76.

[Bro41] Rowland Leonard Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37:194–197, 4 1941.

[Bro50] L.E.J. Brouwer. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[Bur] Burke. I want an easy gadget to prove Planar Hamiltonian Cycle is NP-complete (from Hamiltonian Cycle). https://cstheory.stackexchange.com/questions/9587/i-want-an-easy-gadget-to-prove-planar-hamiltonian-cycle-np-co

[BW91] Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8:225–242, 1991. https://link.springer.com/content/pdf/10.1007/BF00383444.pdf.

[BW05] Graham R. Brightwell and Peter Winkler. Counting eulerian circuits is #p-complete. In Camil Demetrescu, Robert Sedgewick, and Roberto Tamassia, editors, *Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005*, pages 259–262. SIAM, 2005.
http://www.siam.org/meetings/analco05/papers/09grbrightwell.pdf.

[Can88] John F. Canny. Some algebraic and geometric computations in PSPACE. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 460–467. ACM, 1988.
https://doi.org/10.1145/62212.62257.

[Car15] Jean Cardinal. Computational geometry column 62. *SIGACT News*, 46(4):69–78, 2015.
https://doi.org/10.1145/2852040.2852053.

[CBH21] Alex Churchill, Stella Biderman, and Austin Herrick. Magic: The gathering is turing complete. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
https://doi.org/10.4230/LIPIcs.FUN.2021.9.

[CC03] Miroslav Chlebík and Janka Chlebíková. Approximation hardness for small occurrence instances of NP-hard problems. In *Algorithms and Complexity, 5th Italian Conference, CIAC 2003, Rome, Italy, May 28-30, 2003, Proceedings*, pages 152–164, 2003.

[CCHM15] Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1234–1251. SIAM, 2015.
https://doi.org/10.1137/1.9781611973730.82.

[CD68] Richard W. Cottle and George B. Dantzig. Complentary pivot theory of mathematical programming. *Linear Algebra and its applications*, 1:103–125, 1968.

[CD06] Xi Chen and Xiaotie Deng. Settling the complexity of two-player nash equilibrium. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 261–272. IEEE Computer Society, 2006.
https://doi.org/10.1109/FOCS.2006.69.

[CDE+20] Jean Cardinal, Erik D. Demaine, David Eppstein, Robert A. Hearn, and Andrew Winslow. Reconfiguration of satisfying assignments and subset sums: Easy to find, hard to connect. *Theoretical Computer Science*, 806(2):332–343, February 2020.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, 2008.
https://doi.org/10.1145/1365815.1365816.

[CDP08] Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal of Discrete Mathematics*, 22(1):124–138, 2008.
http://dx.doi.org/10.1137/060652063.

[CDR86] Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986.
https://doi.org/10.1137/0215006.

[CE91] Leizhen Cai and John A. Ellis. NP-completeness of edge-colouring some restricted graphs. *Discrete Applied Mathmeatics*, 30(1):15–27, 1991.
https://doi.org/10.1016/0166-218X(91)90010-T.

[CE04] Mark Cieliebak and Stephan J. Eidenbenz. Measurement errors make the partial digest problem NP-hard. In Martin Farach-Colton, editor, *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*, volume 2976 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2004.

[Ces03] Marco Cesati. The Turing way to parameterized complexity. *Journal of Computing and System Science*, 67(4):654–685, 2003.
https://doi.org/10.1016/S0022-0000(03)00073-4.

[CFG+17] Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *Journal of the Association of Computing Machinery (JACM)*, 64(3):18:1–18:22, 2017.
https://doi.org/10.1145/3051094.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. https://doi.org/10.1007/978-3-319-21275-3.

[CH99] Nadia Creignou and Miki Hermann. On #p-completeness of some counting problems, 1999.
https://www.cs.umd.edu/gasarch/BLOGPAPERS/3colsharpp.pdf.

[Cha20] Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median, +)-convolution, and some geometric 3sum-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020.
https://doi.org/10.1145/3363541.

[Che06] Hubie Chen. A rendezvous of logic, complexity, and algebra. *SIGACT News*, 37(4):85–114, 2006.
https://arxiv.org/abs/cs/0611018.

[Che09] Hubie Chen. A rendezvous of logic, complexity, and algebra. *ACM Comput. Surv.*, 42(1):2:1–2:32, 2009.
https://dl.acm.org/doi/pdf/10.1145/1592451.1592453.

[CHH11] Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. The complexity of request-response games. In Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*, volume 6638 of *Lecture Notes in Computer Science*, pages 227–237. Springer, 2011.

[CHHN14] Julien Cassaigne, Vesa Halava, Tero Harju, and Francois Nicolas. Tighter undecidability bounds for matrix mortality, zero-in-the-corner problems, and more, 2014.
http://arxiv.org/abs/1404.0644.

[CHK$^+$21] Jianer Chen, Qin Huang, Iyad Kanj, Qian Li, and Ge Xia. Streaming algorithms for graph k-matching with optimal or near-optimal update time. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 48:1–48:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
https://doi.org/10.4230/LIPIcs.ISAAC.2021.48.

[CHKX06] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computing and System Science*, 72(8):1346–1367, 2006.
https://doi.org/10.1016/j.jcss.2006.04.007.

[Cho16] Amy Chou. NP-hard triangle packing problems, 2016.
http://www.cs.umd.edu/gasarch/BLOGPAPERS/nphtripack.pdf.

[Chr76] Nicos Christofides. Worst case analysis of a new heuristic for the Travelling Salesman Problem, 1976.
https://apps.dtic.mil/sti/pdfs/ADA025602.pdf.

[Chu12] Alex Churchill. Magic: The gathering is turing complete, 2012.
http://www.toothycat.net/hologram/Turing/.

[Chv79] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[CJ03] Liming Cai and David W. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003. https://core.ac.uk/download/pdf/81180403.pdf.

[CJMS12] Raphaël Clifford, Markus Jalsenius, Ashley Montanaro, and Benjamin Sach. The complexity of flood filling games. *Theory Comput. Syst.*, 50(1):72–92, 2012. https://doi.org/10.1007/s00224-011-9339-2.

[CKK⁺06] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006. https://doi.org/10.1007/s00037-006-0210-9.

[CKS01] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*, volume 7 of *SIAM monographs on discrete mathematics and applications*. SIAM, 2001.

[CKST99] Pierluigi Crescenzi, Viggo Kann, Riccardo Silvestri, and Luca Trevisan. Structure in approximation classes. *SIAM Journal on Computing*, 28(5):1759–1782, 1999. http://dx.doi.org/10.1137/S0097539796304220.

[CKX10] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. https://doi.org/10.1016/j.tcs.2010.06.026.

[CMT20] Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional lower bounds for adaptive massively parallel computation. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 141–151. ACM, 2020. https://doi.org/10.1145/3350755.3400230.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third IEEE Symposium on the Foundations of Computer Science*, pages 151–158, 1971.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press, 1990. https://doi.org/10.1016/b978-0-444-88074-1.50010-x.

[CR18] Daniel W. Cranston and Landon Rabern. Planar graphs are 9/2-colorable. *J. Comb. Theory, Ser. B*, 133:32–45, 2018. https://doi.org/10.1016/j.jctb.2018.04.002.

[CS14] Michael S. Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPIcs*, pages 96–104.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2014.96.

[CT97] Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Inf. Process. Lett.*, 64(4):165–171, 1997.
www.sciencedirect.com/science/article/pii/S0020019097001646.

[Cul96] Karel Culik. An aperiodic set of 13 wang tiles. *Discret. Math.*, 160(1-3):245–251, 1996.
https://doi.org/10.1016/S0012-365X(96)00118-5.

[Cul98] J. C. Culberson. Sokoban is PSPACE-complete. In *Proceedings International Conference on Fun with Algorithms (FUN98)*, pages 65–76, Waterloo, Ontario, Canada, June 1998. Carleton Scientific.

[CW09] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214. ACM, 2009.
https://doi.org/10.1145/1536414.1536445.

[CW19] Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40. SIAM, 2019.
https://doi.org/10.1137/1.9781611975482.2.

[CW21] Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021.
https://doi.org/10.1145/3402926.

[Dal99] Víctor Dalmau. A dichotomy theorem for learning quantified boolean formulas. *Mach. Learn.*, 35(3):207–224, 1999.
https://doi.org/10.1023/A:1007582729656.

[Dav73] Martin Davis. Hilbert's tenth problem is unsolvable. *American Mathematical Monthly*, pages 233–2695, 1973.
https://www.math.umd.edu/laskow/Pubs/713/Diophantine.pdf.

[dBK12] Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal of Computational Geometry*, 22(3):187–206, 2012.
https://www.win.tue.nl/mdberg/Papers/2010/bk-obspp-10.pdf.

[DD07] Erik D. Demaine and Martin L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23(Supplement-1):195–208, 2007.
https://doi.org/10.1007/s00373-007-0713-4.

[DDD18] Andreas Darmann, Janosch Docker, and Britta Dorn. The monotone satisfiability problem with bounded variable appearances. *International Journal of Foundations of Computer Science*, 29:979–993, 2018.

[DDE00] Erik D. Demaine, Martin L. Demaine, and David Eppstein. Phutball endgames are hard. *Computing Research Repository*, cs.CC/0008025, 2000. https://arxiv.org/abs/cs/0008025.

[DDE+11] Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow. Algorithms for solving Rubik's cubes. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 689–700, 2011.

[DDHO01] Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O'Rourke. Pushing blocks is hard. *Computational Geometry: Theory and Applications*, 26:21–36, 2001.
https://www.sciencedirect.com/science/article/pii/S0925772102001700.

[DDO00] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2d. In *Proceedings of the 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick, Canada, August 16-19, 2000*, 2000.
http://www.cccg.ca/proceedings/2000/26.ps.gz.

[DDST16] Benjamin Doerr, Carola Doerr, Reto Spöhel, and Henning Thomas. Playing mastermind with many colors. *Journal of the Association of Computing Machinery*, 63(5):42:1–42:23, 2016.
https://doi.org/10.1145/2987372.

[DE11] Erik D. Demaine and Sarah Eisenstat. Flattening fixed-angle chains is strongly NP-hard. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *12th International Workshop on Algorithms and Data Structures (WADS), 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, volume 6844 of *Lecture Notes in Computer Science*, pages 314–325. Springer, 2011.

[Der10] Dariusz Dereniowski. Phutball is PSPACE-hard. *Theoretical Computer Science*, 411(44-46):3971–3978, 2010. https://doi.org/10.1016/j.tcs.2010.08.019.

[Der14] Franck Dernoncourt. TrackMania is NP-complete, 2014.
https://arxiv.org/pdf/1411.5765v1.pdf/.

[DF86] Martin E. Dyer and Alan M. Frieze. Planar 3DM is NP-complete. *Journal of Algorithms*, 7(2):174–184, 1986. https://www.math.cmu.edu/af1p/Texfiles/3DM.pdf.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.
https://doi.org/10.1007/978-1-4612-0515-9.

[DFHT05] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the Association of Computing Machinery (JACM)*, 52(6):866–893, 2005.
http://www-math.mit.edu/hajiagha/hminorfreeJ15.pdf.

[DFL10] Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard, 2010.
http://arxiv.org/abs/1008.1224.

[DFR98] Rodney G. Downey, Michael R. Fellows, and Kenneth W. Regan. Parameterized circuit complexity and the W hierarchy. *Theoretical Computer Science*, 191(1-2):97–115, 1998.
https://doi.org/10.1016/S0304-3975(96)00317-9.

[DGH⁺02] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-sat based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.
https://doi.org/10.1016/S0304-3975(01)00174-8.

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.
https://doi.org/10.1137/S0097539701398363.

[DGKR05] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM J. Comput.*, 34(5):1129–1146, 2005.
https://doi.org/10.1137/S0097539704443057.

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal of Computing*, 39(1):195–259, 2009.
https://doi.org/10.1137/070699652.

[DGPT91] Josep Díaz, Alan Gibbons, Mike Paterson, and Jacobo Torán. The MINSUMCUT problem. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 2nd Workshop WADS '91, Ottawa, Canada, August*

14-16, 1991, Proceedings, volume 519 of *Lecture Notes in Computer Science*, pages 65–89. Springer, 1991.
https://doi.org/10.1007/BFb0028251.

[DH01] Erik D. Demaine and Michael Hoffmann. Pushing blocks is NP-complete for noncrossing solution paths. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 65–68, 2001.
http://www.cccg.ca/proceedings/2001/eddemaine-24711.ps.gz.

[DHH02] Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann. Push-2f is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, pages 31–35, Lethbridge, Alberta, Canada, August 2002. http://www.cs.uleth.ca/wismath/cccg/papers/31.ps.

[DHH04] Erik Demain, Michael Hoffmann, and Markus Holzer. Pushpush-k is PSPACE complete. In *Third international conference on fun with algorithms*, pages 159–170, 2004.
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.1272&rep=rep1&type=pdf.

[DHHL22] Erik D. Demaine, Robert A. Hearn, Dylan Hendrickson, and Jayson Lynch. PSPACE-completeness of reversible deterministic systems, 2022.
https://arxiv.org/abs/2207.07229.

[DHL20] Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 62:1–62:42. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
https://doi.org/10.4230/LIPIcs.ITCS.2020.62.

[DHT04] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensional theory of bounded-genus graphs. *SiAM Journal of Discrete Mathematics*, 18(3):501–511, 2004.
http://www-math.mit.edu/hajiagha/handle.pdf.

[DKK+18a] Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. On non-optimally expanding sets in grassmann graphs. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 940–951. ACM, 2018.
https://doi.org/10.1145/3188745.3188806.

[DKK+18b] Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. Towards a proof of the 2-to-1 games conjecture? In Ilias Diakonikolas, David Kempe, and

Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 376–389. ACM, 2018.
https://doi.org/10.1145/3188745.3188804.

[DKL20] Erik D. Demaine, Justin Kopinsky, and Jayson Lynch. Recursed is not recursive: A jarring result. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 50:1–50:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
https://doi.org/10.4230/LIPIcs.ISAAC.2020.50.

[DKS19] Yuval Dagan, Gil Kur, and Ohad Shamir. Space lower bounds for linear prediction in the streaming model. In Alina Beygelzimer and Daniel Hsu, editors, *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, volume 99 of *Proceedings of Machine Learning Research*, pages 929–954. PMLR, 2019.
http://proceedings.mlr.press/v99/dagan19b.html.

[DLL$^+$02] Xiaotie Deng, Guojun Li, Zimao Li, Bin Ma, and Lusheng Wang. A PTAS for distinguishing (sub)string selection. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 740–751. Springer, 2002.
www.cs.umd.edu/users/gasarch/BLOGPAPERS/dsubptas.pdf.

[DM99] Mauro Dell'Amico and Silvano Martello. Reduction of the three-partition problem. *J. Comb. Optim.*, 3(1):17–30, 1999.
https://doi.org/10.1023/A:1009856820553.

[dM21] Carl de Marcken. Computational complexity of air travel planning, 2021.
https://www.cs.umd.edu/gasarch/BLOGPAPERS/airtravel.pdf.

[DMR09] Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009.
https://doi.org/10.1137/07068062X.

[DMS$^+$18] Erik D. Demaine, Fermi Ma, Ariel Schvartzman, Erik Waingarten, and Scott Aaronson. The fewest clues problem. *Theor. Comput. Sci.*, 748:28–39, 2018.
https://doi.org/10.1016/j.tcs.2018.01.020.

[DOUU14] Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno. Computational complexity and an integer programming model of shakashaka. *IEICE Transactions*, 97-A(6):1213–1219, 2014.
https://www.cs.umd.edu/users/gasarch/BLOGPAPERS/shak.pdf.

[DPPS01] Josep Díaz, Mathew D. Penrose, Jordi Petit, and Maria J. Serna. Approximating layout problems on random geometric graphs. *J. Algorithms*, 39(1):78–116, 2001. https://doi.org/10.1006/jagm.2000.1149.

[DPR61] Martin Davis, Hillary Putnam, and Julia Robinson. The decision problem for exponential diophantine equations. *Annal of Mathematics*, 74:425–436, 1961.

[DPS02] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, September 2002. http://doi.acm.org/10.1145/568522.568523.

[DQS12] Xiaotie Deng, Qi Qi, and Amin Saberi. Algorithmic solutions for envy-free cake cutting. *Oper. Res.*, 60(6):1461–1476, 2012. https://doi.org/10.1287/opre.1120.1116.

[DR17] Erik D. Demaine and Mikhail Rudoy. Hamiltonicity is hard in thin or polygonal grid graphs, but easy in thin polygonal grid graphs. arXiv:1706.10046, 2017. https://arxiv.org/abs/1706.10046.

[DS02] Irit Dinur and Shmuel Safra. The importance of being biased. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 33–42. ACM, 2002. https://doi.org/10.1145/509907.509915.

[DS05] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, pages 439–485, 2005. http://www.wisdom.weizmann.ac.il/dinuri/mypapers/vc.pdf.

[DS13] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the 46th annual ACM Sympsosium on Theory of Computing*, pages 624–633, 2013. https://dl.acm.org/doi/pdf/10.1145/2591796.2591884.

[DSA18] Ali Dehghan, Mohammad-Reza (Rafsanjani) Sadeghi, and Arash Ahadi. Not-all-equal and 1-in-degree decompositions: Algorithmic complexity and applications. *Algorithmica*, 80(12):3704–3727, 2018. https://doi.org/10.1007/s00453-018-0412-y.

[DvM14] Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the Association of Computing Machinery (JACM)*, 61(4):23:1–23:27, 2014. https://doi.org/10.1145/2629620.

[DVW16] Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Brothers is harder-easier than we thought. In *Fun with algorithms*, 2016.

[EG04] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004. https://doi.org/10.1016/j.tcs.2004.05.009.

[EGLM19] David Eppstein, Michael T. Goodrich, James A. Liu, and Pedro Matias. Tracking paths in planar graphs. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, i December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 54:1–54:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. https://doi.org/10.4230/LIPIcs.ISAAC.2019.54.

[EHL+18] Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. *ACM Trans. Algorithms*, 14(4):48:1–48:23, 2018. https://doi.org/10.1145/3230819.

[EJ64] Calrton E.Lemke and J.T.Howson. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics*, 12:412–423, 1964. https://epubs.siam.org/doi/pdf/10.1137/0112033.

[ENSS98] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[EP06] Bruno Escoffier and Vangelis Th. Paschos. Completeness in approximation classes beyond APX. *Theoretical Computer Science.*, 359(1-3):369–377, 2006. http://dx.doi.org/10.1016/j.tcs.2006.05.023.

[Epp87] David Eppstein. On the NP-completeness of cryptarithms. *SIGACT News*, 18(3):38–40, 1987.

[ES75] Guy Even and Yossi Shiloach. NP-comleteness of several arrangement problems, 1075. Tech Report 43, Dept of CS, Technion, Haifia.

[FB02] Gary William Flake and Eric B. Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1-2):895–911, 2002. https://doi.org/10.1016/S0304-3975(01)00173-6.

[FdSSPdS15] Michael R. Fellows, Uéverton dos Santos Souza, Fábio Protti, and Maise Dantas da Silva. Tractability and hardness of flood-filling games on trees. *Theor. Comput. Sci.*, 576:102–116, 2015. https://doi.org/10.1016/j.tcs.2015.02.008.

[Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the Association of Computing Machinery (JACM)*, 45:634–652, 1998.

[Fei04] Uriel Feige. Approximating maximum clique by removing subgraphs. *SIAM J. Discret. Math.*, 18(2):219–225, 2004. https://epubs.siam.org/doi/pdf/10.1137/S089548010240415X.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FG18] Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 51–64. ACM, 2018. https://arxiv.org/abs/1711.04503.

[FGGP12] Stephen Fenner, William Gasarch, Charles Glover, and Semmy Purewal. Rectangle free colorings of grids, 2012. http://arxiv.org/abs/1005.3750.

[FGJ+78] A. S. Fraenkel, M. R. Garey, D. S. Johnson, T. Schaefer, and Y. Yesha. The complexity of checkers on an $n \times n$ board. In *19th Annual Symposium on Foundations of Computer Science, 1978*, pages 55–64. IEEE Computer Society, 1978.

[FGL+96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. https://doi.org/10.1145/226643.226652.

[FGLS10] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized with clique-width. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 493–502. SIAM, 2010. https://doi.org/10.1137/1.9781611973075.42.

[FHM+20] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785. SIAM, 2020. https://arxiv.org/abs/1912.10497.

[FHRV09] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. https://doi.org/10.1016/j.tcs.2008.09.065.

[Fil19a] Ivan Tadeu Ferreira Antunes Filho. Characterizing Boolean satisfiability variants. Master's thesis, Massachusetts Institute of Technology, 2019. http://erikdemaine.org/theses/ifilho.pdf.

[Fil19b] Ivan Tadeu Ferreira Antunes Filho. Characterizing Boolean satisfiability variants. Master's thesis, Massachusetts Institute of Technology, September 2019.

[FK02] Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002. https://doi.org/10.1137/S0097539701387660.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. www.cs.cmu.edu/sleator/papers/competitive-paging.pdf.

[FL81] Avierzi S. Fraenkel and David Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *Journal of Combinatorial Theory (Series A)*, 31:199–214, 1981. http://www.ms.mff.cuni.cz/truno7am/slozitostHer/chessExptime.pdf.

[For10] Michal Forišek. Computational complexity of two-dimensional platform games. In *Fun with Algorithms*, pages 214–227. Springer, 2010.

[FR92] Martin Fürer and Balaji Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In Greg N. Frederickson, editor, *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida, USA*, pages 317–324. ACM/SIAM, 1992. http://dl.acm.org/citation.cfm?id=139404.139469.

[FR04] Uriel Feige and Daniel Reichman. On systems of linear equations with two variables per equation. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, volume 3122 of *Lecture Notes in Computer Science*, pages 117–127. Springer, 2004.

[FRdSS18] Michael R. Fellows, Frances A. Rosamond, Maise Dantas da Silva, and Uéverton S. Souza. A survey on the complexity of flood-filling games. In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovic on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 357–376. Springer, 2018.

[Fre76] Michael Fredmann. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976. https://epubs.siam.org/doi/pdf/10.1137/0205006.

[Fri02a] Erich Friedman. Corral puzzles are NP-complete, 2002. http://www.cs.umd.edu/gasarch/BLOGPAPERS/corral.pdf.

[Fri02b] Erich Friedman. Pushing blocks in gravity is NP-hard, 2002. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.6066&rep=rep1&type=pdf.

[Fri02c] Erich Friedman. Spiral galaxies puzzles are NP-complete, 2002. https://slideplayer.com/slide/254461/.

[FT82] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, pages 219–253, 1982. https://link.springer.com/article/10.1007/BF01857727.

[FW90] Michael Formann and Frank Wagner. The VLSI layout in various embedding models. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science, 16rd International Workshop, WG '90, Berlin, Germany, June 20-22, 1990, Proceedings*, volume 484 of *Lecture Notes in Computer Science*, pages 130–139. Springer, 1990.

[FZ14] Nathanaël Fijalkow and Martin Zimmermann. Parity and streett games with costs. *Log. Methods Comput. Sci.*, 10(2), 2014. https://doi.org/10.2168/LMCS-10(2:14)2014.

[Gas09a] William Gasarch. The 17×17 challenge. Worth $289.00. This is not a joke, 2009. November 30, 2009 entry on ComplexityBlog (Google Fortnow Blog).

[Gas09b] William Gasarch. Whats your game Mr. Bond? Nim?, 2009. https://blog.computationalcomplexity.org/2009/12/whats-your-game-mr-bond-nim.html.

[Gas09c] William Gasarch. Whats your game Mr. Bond?-The sequel!, 2009. https://blog.computationalcomplexity.org/2010/06/whats-your-game-mr-bond-sequel.html.

[Gas19] William Gasarch. Complexity Theory Column 100: The P=NP poll. *SIGACT News*, 50(1):28–34, 2019. https://www.cs.umd.edu/users/gasarch/papers/poll3.pdf.

[Gas21] William Gasarch. Hilbert's 10th problem for fixed d and n, 2021.

[Gav77] F. Gavril. Some NP-complete problems on graphs. In *Proceedings of the 11th Conference on Information Science. John Hopkins University*, pages 91–95, 1977.

[GB07] Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. https://doi.org/10.1007/s00453-007-0010-x.

[GGJ20] Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC algorithms for MIS, matching, and coloring on trees and beyond. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 34:1–34:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
https://doi.org/10.4230/LIPIcs.DISC.2020.34.

[GGJK78] Michael R. Garey, Ronald Graham, David S. Johnson, and Donald E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal of Applied Mathematics*, 34:477–495, 1978.

[GGN06] Jens Gramm, Jiong Guo, and Rolf Niedermeier. Parameterized intractability of distinguishing substring selection. *Theory Comput. Syst.*, 39(4):545–560, 2006.
www.cs.umd.edu/users/gasarch/BLOGPAPERS/dsub.pdf.

[GHM$^+$11] Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011.
https://doi.org/10.1137/090756144.

[GHS02] Venkatesan Guruswami, Johan Håstad, and Madhu Sudan. Hardness of approximate hypergraph coloring. *SIAM J. Comput.*, 31(6):1663–1686, 2002.
https://doi.org/10.1137/S0097539700377165.

[GJ77] Michael R. Garey and David S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.
https://www.jstor.org/stable/pdf/2100192.pdf.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, 1979.

[GJ83] Michael R. Garey and David S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[GJS76] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.
https://www.sciencedirect.com/science/article/pii/0304397576900591.

[GJT76] Michael R. Garey, David S. Johnson, and Robert E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.
http://dx.doi.org/10.1137/0205049.

[GKK12] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012.
https://doi.org/10.1137/1.9781611973099.41.

[GKM+12] Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 478–490. Springer, 2012.

[GKM+19] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 26–37. IEEE Computer Society, 2019. https://doi.org/10.1109/FOCS.2019.00011.

[GKMP09] Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009. https://doi.org/10.1137/07070440X.

[GKR00] Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000. https://doi.org/10.1006/jagm.2000.1096.

[GKSZ20] Mika Göös, Pritish Kamath, Katerina Sotiraki, and Manolis Zampetakis. On the complexity of modulo-q arguments and the chevalley - warning theorem. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 19:1–19:42. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. https://doi.org/10.4230/LIPIcs.CCC.2020.19.

[GKU19] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1650–1663. IEEE Computer Society, 2019. https://doi.org/10.1109/FOCS.2019.00097.

[GLS81] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. https://doi.org/10.1007/BF02579273.

[GMKS95] P.W. Goldberg, M.C.Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. *Journal of Computational Biology*, 2(1):139–152, 1995.

[GO12] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 45(4):140–152, 2012. https://doi.org/10.1016/j.comgeo.2011.11.006.

[Goe97] Michel X. Goemans. Semidefinite programming in combinatorial optimization. *Math. Program.*, 79:143–161, 1997.
https://doi.org/10.1007/BF02614315.

[Gol78] E. Mark Gold. Complexity of automaton identification from given data. *Information and Computation*, 37:302–320, 1978.

[Gol97] P.A. Golovach. The total vertex separation number of a graph (in Russian). *Diskretnaya Matematika*, 10(1):87–94, 1997.

[Goo09] Michael T. Goodrich. On the algorithmic complexity of the mastermind game with black-peg results. *Inf. Process. Lett.*, 109(13):675–678, 2009.
https://doi.org/10.1016/j.ipl.2009.02.021.

[GP18] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *Journal of the Association of Computing Machinery (JACM)*, 65(4):22:1–22:25, 2018.
https://doi.org/10.1145/3185378.

[Gre67] Sheila Greibach. A note on undecidable properties of formal languages, 1967. SDC Document TM-738/038/00.

[Gre21] Bogdan Grechuk. Diophantine equations: a systematic approach, 2021.
https://arxiv.org/abs/2108.08705.

[GS81] Fritz J. Grunewald and Dan Segal. How to solve a quadratic equations in integers. *Mathematical Proceedings of the Cambridge Philosophical Society*, 89:1–5, 1981.

[GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653. SIAM, 2019.
https://doi.org/10.1137/1.9781611975482.99.

[Gus93] Jens Gustedt. On the pathwidth of chordal graphs. *Discret. Appl. Math.*, 45(3):233–248, 1993.
https://doi.org/10.1016/0166-218X(93)90012-D.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
https://dl.acm.org/doi/pdf/10.1145/227683.227684.

[Had75] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.*, 4(3):221–225, 1975.
http://www.cs.umd.edu/users/gasarch/BLOGPAPERS/planarmaxcut.pdf.

[Ham14] Linus Hamilton. Braid is undecidable, 2014.
https://arxiv.org/pdf/1412.0784.pdf.

[Has99] Johan Hastad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 1999.
https://www.csc.kth.se/johanh/cliqueinap.pdf.

[Has01] Johan Hastad. Some optimal inapproximability results. *Journal of the Association of Computing Machinery (JACM)*, 48(4):798–859, 2001.
https://dl.acm.org/doi/10.1145/502090.502098.

[HD09] Robert A. Hearn and Erik D. Demaine. *Game, Puzzles, and Computation*. A K Peters/CRC Press, 1st edition, July 2009.

[Hea06] Robert A. Hearn. *Games, puzzles, and computation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.
http://hdl.handle.net/1721.1/37913.

[Hea09] Robert A. Hearn. Amazons, konane, and cross purposes are PSPACE-complete. *Games of No Chance 3*, 56:287–306, 2009.
http://library.msri.org/books/Book56/files/31hearn.pdf.

[Hem04] Edith Hemaspaandra. Dichotomy theorems for alternation-bounded quantified boolean formulas, 2004.
http://arxiv.org/abs/cs/0406006.

[Her14] Timon Hertli. 3-sat faster and simpler - Unique-SAT bounds for PPSZ hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014.
https://arxiv.org/abs/1103.2165.

[HHHK05] Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumaki. Skolem's problem—on the border between decidable and undecidable. Technical Report 683, Turku Centre for Computer Science, 2005.

[HIN+12] Takashi Horiyama, Takehiro Ito, Keita Nakatsuka, Akira Suzuki, and Ryuhei Uehara. Packing trominoes is NP-complete, #P-complete and ASP-complete. In *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG 2012, Charlottetown, Prince Edward Island, Canada, August 8-10, 2012*, pages 211–216, 2012.
http://2012.cccg.ca/papers/paper67.pdf.

[HJ12] Markus Holzer and Sebastian Jakobi. On the complexity of rolling block and Alice mazes. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 210–222. Springer, 2012.

[HJW85] John E. Hopcroft, Deborah Joseph, and Sue Whitesides. On the movement of robot arms in 2-dimensional bounded regions. *SIAM Journal on Computing.*, 14(2):315–333, 1985.
https://doi.org/10.1137/0214025.

[HL94] Steven Homer and Luc Longpré. On reductions of NP sets to sparse sets. *J. Comput. Syst. Sci.*, 48(2):324–336, 1994.
https://doi.org/10.1016/S0022-0000(05)80006-6.

[Hof00] Michael Hoffmann. Motion planning admidst movable square blocs: Push-* is NP-hard. In *Proceedings of the 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick, Canada, August 16-19, 2000*, 2000.
https://people.inf.ethz.ch/hoffmann/pub/h-psnh-00.pdf.

[Hop69] John E. Hopcroft. On the equivalence and containment problems for context-free languages. *Math. Syst. Theory*, 3(2):119–124, 1969.
https://doi.org/10.1007/BF01746517.

[Hor51] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[HR97] Magnús M. Halldórsson and Jaikumar Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.
https://doi.org/10.1007/BF02523693.

[HR12] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. *Theory of Computing*, 8(1):513–531, 2012.
https://doi.org/10.4086/toc.2012.v008a023.

[HSSS21] MohammadTaghi Hajiaghayi, Hamed Saleh, Saeed Seddighin, and Xiaorui Sun. String matching with wildcards in the massively parallel computation model. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 275–284. ACM, 2021.
https://doi.org/10.1145/3409964.3461793.

[HT74] John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.
https://doi.org/10.1145/321850.321852.

[INKT21] Fumitaka Ito, Masahiko Naito, Naoyuki Katabami, and Tatsuie Tsukiji. EXPTIME hardness of an n by n custodian capture game. *Algorithms*, 14(3):70, 2021.
https://doi.org/10.3390/a14030070.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Science*, 62(2):367–375, 2001.
https://doi.org/10.1006/jcss.2000.1727.

[IPS82] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computing and System Science*, 63(4):512–530, 2001.
https://cseweb.ucsd.edu/russell/ipz.pdf.

[IW05] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208. ACM, 2005.
https://doi.org/10.1145/1060590.1060621.

[IZ89] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 248–253. IEEE Computer Society, 1989.
https://doi.org/10.1109/SFCS.1989.63486.

[Jan68] E. Janovkaya. Equilibrium points in polymatrix games. *Litovskii Matematicheskii Sbornik*, 8:381–384, 1968.

[JDU+74] David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, Michael R. Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal of Computing*, 3(4):299–325, 1974.
https://doi.org/10.1137/0203025.

[Jer94] Mark Jerrum. Counting trees in a graph is #p-complete. *Inf. Process. Lett.*, 51(3):111–116, 1994.
https://doi.org/10.1016/0020-0190(94)00085-9.

[Jer16] Emil Jerábek. Integer factoring and modular square roots. *Journal of Computing and System Sciences*, 82(2):380–394, 2016.
https://arxiv.org/pdf/1207.5220.pdf.

[Joh87] David S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 8(3):438–448, 1987.
https://doi.org/10.1016/0196-6774(87)90021-6.

[Jon82] James Jones. Universal diophantine equations. *The Journal of Symbolic Logic*, 47(3):549–571, 1982.
http://www.jstor.org/stable/2273588?seq=1#metadata infotabcontents.

[Jon98] Peter Jonsson. Near-optimal nonapproximability results for some NPO PB-complete problems. *Information Processing Letters*, 68(5):249–253, 1998.
http://dx.doi.org/10.1016/S0020-0190(98)00170-7.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. https://doi.org/10.1016/0022-0000(88)90046-3.

[JT09] Marcin Jurdzinski and Ashutosh Trivedi. Reachability-time games on timed automata, 2009. http://arxiv.org/abs/0907.3414.

[JZZZ12] Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *J. Comb. Optim.*, 23(4):519–527, 2012. https://doi.org/10.1007/s10878-010-9370-2.

[KAI79] Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM Journal of Computing*, 8(4):574–586, 1979.

[Kap21] Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1874–1893. SIAM, 2021. https://doi.org/10.1137/1.9781611976465.112.

[Kar68] Jerome J Karaganis. On the cube of a graph. *Canadian Mathematical Bulletin*, 11:295–296, 1968.

[Kar72] Richard Karp. Reducibilities among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of computer computations*, pages 85–103. Plenum Press, 1972.

[Kar88] Richard Karp. A survey of parallel algorithms for shared-memory machines, 1988. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/CSD-88-408.pdf.

[Kay00] Richard Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000.

[KC00] Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 73–79. ACM, 2000. https://doi.org/10.1145/335305.335314.

[Kho01] Subhash Khot. Improved inaproximability results for maxclique, chromatic number and approximate graph coloring. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 600–609. IEEE Computer Society, 2001. https://doi.org/10.1109/SFCS.2001.959936.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002*, page 25. IEEE Computer Society, 2002. https://dl.acm.org/doi/pdf/10.1145/509907.510017.

[Kho05] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005. https://doi.org/10.1145/1089023.1089027.

[Kho06] Subhash Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006. https://epubs.siam.org/doi/10.1137/S0097539705447037.

[Kho10a] Subhash Khot. On the unique games conjecture (invited survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9-12, 2010*, pages 99–121. IEEE Computer Society, 2010. https://doi.org/10.1109/CCC.2010.19.

[Kho10b] Subhash Khot. On the unique games conjecture (invited survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9-12, 2010*, pages 99–121. IEEE Computer Society, 2010. https://doi.org/10.1109/CCC.2010.19.

[Kin15] William B. Kinnersley. Cops and robbers is exptime-complete. *J. Comb. Theory, Ser. B*, 111:201–220, 2015. https://doi.org/10.1016/j.jctb.2014.11.002.

[KK82] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 312–320. IEEE Computer Society, 1982. https://doi.org/10.1109/SFCS.1982.61.

[KKMO07] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. https://www.cs.cmu.edu/odonnell/papers/maxcut.pdf.

[KKOG20] Anna Karlin, Nathan Klein, and Shayan Oveis-Gharan. A (slightly) improved approximation algorithm for metric TSP, 2020. https://arxiv.org/abs/2007.01409.

[Kle43] Stephen Kleene. Recursive predicates and quantifiers. *Transactions of the the American Mathematics Society*, 53:41–73, 1943.

[KLM+14] Raimondas Kiveris, Silvio Lattanzi, Vahab S. Mirrokni, Vibhor Rastogi, and Sergei Vassilvitskii. Connected components in mapreduce and beyond. In Ed Lazowska, Doug Terry, Remzi H. Arpaci-Dusseau, and Johannes Gehrke, editors, *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 3-5, 2014*, pages 18:1–18:13. ACM, 2014.
https://doi.org/10.1145/2670979.2670997.

[KLS00] Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.
https://doi.org/10.1007/s004930070013.

[KLS01] Michael Kearns, Michael L. Littman, and Satinder P. Singh. Graphical models for game theory. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, 2001.
http://arxiv.org/abs/1301.2281.

[KM94] Jan Kratochvíl and Jirí Matousek. Intersection graphs of segments. *J. Comb. Theory, Ser. B*, 62(2):289–315, 1994.
https://doi.org/10.1006/jctb.1994.1071.

[KMN11] Anna R. Karlin, Claire Mathieu, and C. Thach Nguyen. Integrality gaps of linear and semi-definite programming relaxations for knapsack. In Oktay Günlük and Gerhard J. Woeginger, editors, *Integer Programming and Combinatoral Optimization - 15th International Conference, IPCO 2011, New York, NY, USA, June 15-17, 2011. Proceedings*, volume 6655 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 2011.
https://homes.cs.washington.edu/karlin/papers/knapsack.pdf.

[KMS98] David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998.
https://doi.org/10.1145/274787.274791.

[KMS17] Subhash Khot, Dor Minzer, and Muli Safra. On independent sets, 2-to-2 games, and grassmann graphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 576–589. ACM, 2017.
https://doi.org/10.1145/3055399.3055432.

[KMS18] Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 592–601. IEEE Computer Society, 2018.
https://doi.org/10.1109/FOCS.2018.00062.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, England, 1997.

[KO09] Subhash Khot and Ryan O'Donnell. SDP gaps and ugc-hardness for max-cut-gain. *Theory Comput.*, 5(1):83–117, 2009. https://doi.org/10.4086/toc.2009.v005a004.

[KP03] Phokion G. Kolaitis and Jonathan Panttaja. On the complexity of existential pebble games. In Matthias Baaz and Johann A. Makowsky, editors, *Computer Science Logic, 17th International Workshop, CSL 2003, 12th Annual Conference of the EACSL, and 8th Kurt Gödel Colloquium, KGC 2003, Vienna, Austria, August 25-30, 2003, Proceedings*, volume 2803 of *Lecture Notes in Computer Science*, pages 314–329. Springer, 2003.

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. https://doi.org/10.1137/1.9781611974331.ch89.

[KPS10] Subhash Khot, Preyas Popat, and Rishi Saket. Approximate lasserre integrality gap for unique games. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2010.

[KR92] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992. https://doi.org/10.1137/0405033.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2 - \varepsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[Kra99] Adam Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *Journal of Computing and System Science*, 58(3):641–647, 1999. https://doi.org/10.1006/jcss.1998.1611.

[Kri75] Mukkai S. Krishnamoorthy. An NP-hard problem in bipartite graphs. *ACM SIGACT News*, 7(1):26–26, 1975. https://dl.acm.org/doi/10.1145/990518.990521.

[KS92] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5:545–557, 1992. https://epubs.siam.org/doi/pdf/10.1137/0405044.

[KS20] Young Kun Ko and Min Jae Song. Hardness of approximate nearest neighbor search under l-infinity, 2020. https://arxiv.org/abs/2011.06135.

[KSS84] J. Kahn, M. Saks, and D. Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984.

[KV15] Subhash Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1. *Journal of the ACM*, 62(1):8:1–8:39, 2015.
https://doi.org/10.1145/2629614.

[KVV90] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990.
https://doi.org/10.1145/100216.100262.

[KW19] Daniel M. Kane and Richard Ryan Williams. The orthogonal vectors conjecture for branching programs and formulas. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 48:1–48:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
https://doi.org/10.4230/LIPIcs.ITCS.2019.48.

[KZ97] Howard Karloff and Uzi Zwick. A 7/8-approximation algorithm for MAX 3-SAT? In *38th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 406–415. IEEE Computer Society, 1997.
http://www.cs.umd.edu/gasarch/BLOGPAPERS/max3satu.pdf.

[Lad75] Richard Ladner. On the structure of polynomial time reducibilities. *Journal of the Association of Computing Machinery*, 22:155–171, 1975.

[Lam11] Michael Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Information Processing Letters*, 111(23-24):1089–1091, 2011.
https://doi.org/10.1016/j.ipl.2011.09.003.

[Las01a] Jean B. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In Karen Aardal and Bert Gerards, editors, *Integer Programming and Combinatorial Optimization, 8th International IPCO Conference, Utrecht, The Netherlands, June 13-15, 2001, Proceedings*, volume 2081 of *Lecture Notes in Computer Science*, pages 293–303. Springer, 2001.

[Las01b] Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, 11(3):796–817, 2001.
https://doi.org/10.1137/S1052623400366802.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.
https://doi.org/10.1016/0020-0190(83)90044-3.

[LC89] K. Li and K. H. Cheng. Complexity of resource allocation and job scheduling problems on partitionable mesh connected systems. In *Proceedings of the 1st Annual IEEE Symposium on Parallel and Distributed Processing*, pages 358–365, May 1989.

[Len81] Thomas Lengauer. Black-white pebbles and graph separation. *Acta Informatica*, 16:465–475, 1981.
https://doi.org/10.1007/BF00264496.

[Lev73] Leonid Levin. Universal search problems. *Problems of Information Transmission*, 9:115–116, 1973.

[Lew78] Harry Lewis. Renaming a set of clauses as a Horn set. *Journal of the Association for Computing Machinery*, 25(1):134–135, 1978.

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. https://epubs.siam.org/doi/pdf/10.1137/0211025.

[Lin02] Nathan Linial. Finite metric spaces: combinatorics, geometry and algorithms. In Ferran Hurtado, Vera Sacristán, Chandrajit Bajaj, and Subhash Suri, editors, *Proceedings of the 18th Annual Symposium on Computational Geometry, Barcelona, Spain, June 5-7, 2002*, page 63. ACM, 2002.
https://doi.org/10.1145/513400.513441.

[LLL82] A. Lenstra, H. Lenstra, and L Lovasz. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:513–534, 1982.

[LLM+03] J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003.
www.sciencedirect.com/science/article/pii/S0890540103000579.

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Comb.*, 15(2):215–245, 1995.
https://doi.org/10.1007/BF01200757.

[LLZ02] Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-sat and MAX DI-CUT problems. In William J. Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, volume 2337 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.
https://dl.acm.org/doi/10.5555/645591.660076.

[LMM03] Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In Daniel A. Menascé and Noam Nisan, editors, *Proceedings 4th ACM Conference on Electronic Commerce (EC-2003), San Diego, California, USA, June 9-12, 2003*, pages 36–41. ACM, 2003.
https://doi.org/10.1145/779928.779933.

[Lov75] László Lovász. On the ratio of optimal integral and fractional covers. *Discret. Math.*, 13(4):383–390, 1975.
https://doi.org/10.1016/0012-365X(75)90058-8.

[LPV20] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
https://doi.org/10.4230/LIPIcs.ITCS.2020.53.

[LR99] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
https://doi.org/10.1145/331524.331526.

[LTW+90] Joseph Y.-T. Leung, Tommy W. Tam, C. S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.
https://doi.org/10.1016/0743-7315(90)90019-L.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.
https://doi.org/10.1137/0215074.

[LY89] Ming Li and Yaacov Yesha. New lower bounds for parallel computation. *J. ACM*, 36(3):671–680, 1989.
https://doi.org/10.1145/65950.65959.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.
https://dl.acm.org/doi/pdf/10.1145/185675.306789.

[Mah82] Stephen R. Mahaney. Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.

[Mai78] David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, 1978.
https://doi.org/10.1145/322063.322075.

[Mar12] Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012.

[Mar13] Daniel Marx. The square root phenomenon in planar graphs (slides), 2013. http://www.cs.bme.hu/dmarx/papers/marx-faw-2013-planar.pdf.

[Mat] Yuri Matijasevic. Hilbert's tenth problem: What can we do with diophantine equations? https://logic.pdmi.ras.ru/yumat/personaljournal/H10history/H10histe.pdf.

[Mat70] Yuri Matijasevic. Enumerable sets are diophantine (Russian). *Doklady Academy Nauk, SSSR*, 191:279–282, 1970. Translation in Soviet Math Doklady, Vol 11, 1970.

[Mat93] Yuri Matijasevic. *Hilbert's Tenth Problem.* MIT Press, Cambridge, 1993.

[Mat14] Jirí Matousek. Intersection graphs of segments and $\exists r$, 2014. http://arxiv.org/abs/1406.2636.

[McG78] Richard McGregor. *On partitioning a graph: a theoretical and empirical study.* PhD thesis, University of California at Berkeley, 1978.

[McP05] Brandon McPhail. Light up is NP-complete, 2005. Manuscript.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, volume 2 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. https://doi.org/10.1007/978-3-642-69897-2.

[Meh14] Ruta Mehta. Constant rank bimatrix games are PPAD-hard. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014*, pages 545–554. ACM, 2014. https://doi.org/10.1145/2591796.2591835.

[Mel21] Erika Melder. A chronology of set cover inapproximability results, 2021. https://arxiv.org/abs/2111.08100.

[MG92] Jayadev Misra and David Gries. A constructive proof of vizing's theorem. *Inf. Process. Lett.*, 41(3):131–133, 1992. https://doi.org/10.1016/0020-0190(92)90041-S.

[Mic12] Daniele Micciancio. Inapproximability of the shortest vector problem: Toward a deterministic reduction. *Theory of Computing*, 8(1):487–512, 2012. https://doi.org/10.4086/toc.2012.v008a022.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computing and System Science*, 13(3):300–317, 1976. https://doi.org/10.1016/S0022-0000(76)80043-8.

[Min67] Marvin Minsky. *Computation: Finite and Infinite Machines*, pages 255–258. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1st edition, 1967.

[Mit99] Joseph Mitchell. Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal of Computing*, 28(4):1298–1309, 1999. https://epubs.siam.org/doi/abs/10.1137/S0097539796309764.

[MNV12] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi R. Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012. https://doi.org/10.1016/j.tcs.2010.05.034.

[Mon86] Burkhard Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM Journal of Algebraic and Discrete Methods*, 7:505–512, 1986.

[Mor88] Bernard Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, 1988. https://dl.acm.org/doi/pdf/10.1145/49097.49099.

[Mos15] Dana Moshkovitz. The projection games conjecture and the NP-hardness of ln n-approximating set-cover. *Theory Comput.*, 11:221–235, 2015. https://doi.org/10.4086/toc.2015.v011a007.

[MPS85] Fillia Makedon, Christos Papadimitriou, and Ivan Sudborough. Topological bandwidth. *SIAM Journal of Algebraic and Discrete Methods*, 6(3):418–444, 1985.

[MR75] Yuri Matijasevic and Julia Robinson. Reduction of an arbitrary diophantine equation to one in 13 unknowns. *Acta Arithmetica*, pages 521–553, 1975.

[MR07] Anna Moss and Yuval Rabani. Approximation algorithms for constrained node weighted steiner tree problems. *SIAM Journal of Computing*, 37(2):460–481, 2007. https://doi.org/10.1137/S0097539702420474.

[MR08] Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *Journal of the Association of Computing Machinery JACM*, 55(2):11:1–11:29, 2008. https://doi.org/10.1145/1346330.1346336.

[MS88] Burkhard Monien and Ivan Hal Sudborough. Min cut is NP-complete for edge weighted treees. *Theoretical Computer Science*, 58:209–229, 1988. https://doi.org/10.1016/0304-3975(88)90028-X.

[MS08] Luke Mathieson and Stefan Szeider. The parameterized complexity of regular subgraph problems and generalizations. In James Harland and Prabhu Manyem, editors, *Theory of Computing 2008. Proc. Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 22-25, 2008. Proceedings*, volume 77 of *CRPIT*, pages 79–86, 2008. http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV77Mathieson.html.

[MS13] Colin McDiarmid and Fiona Skerman. Modularity in random regular graphs and lattices. *Electron. Notes Discret. Math.*, 43:431–437, 2013. https://doi.org/10.1016/j.endm.2013.07.063.

[MT20] Shohei Mishiba and Yasuhiko Takenaga. QUIXO is exptime-complete. *Inf. Process. Lett.*, 162:105995, 2020. https://doi.org/10.1016/j.ipl.2020.105995.

[MTY13] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing the HSSW algorithm for 3-sat. *Algorithmica*, 67(2):112–124, 2013. https://doi.org/10.1007/s00453-012-9741-4.

[MV19] Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019. https://doi.org/10.1007/s00224-018-9878-x.

[Nas50] John Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[Neu28] John Von Neumann. Zur theorie der gesellschaftsspiele. *Math. Annalen.*, 100:295–320, 1928.

[New00] Atlantha Newman. Approximating the maximum acyclic subgraph. Master's thesis, MIT, 2000.

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[Nik] Nikoli. Nikoli puzzles. https://www.nikoli.co.jp/en/puzzles/.

[Nik08] Nikoli. *Akari & Suoku.* Sterling, 2008.

[Nov55] Pytor Novikov. On the algorithmic unsolvability of the word problem in group theory (in russian). *Proceedings of the Steklov Institute of Mathematics*, 44:1–143, 1955.

[NSS95] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. https://doi.org/10.1109/SFCS.1995.492475.

[Opa79] Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979. https://epubs.siam.org/doi/pdf/10.1137/0208008.

[OW08] Ryan O'Donnell and Yi Wu. An optimal sdp algorithm for max-cut, and equally optimal long code tests. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 335–344. ACM, 2008.
https://doi.org/10.1145/1374376.1374425.

[OW12] Ryan O'Donnell and John Wright. A new point of np-hardness for unique games. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 289–306. ACM, 2012.
https://doi.org/10.1145/2213977.2214005.

[Pap76] Christos H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.
https://doi.org/10.1007/BF02280884.

[Pap85] Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and Systems Sciences*, 48(3):498–532, 1994.
https://doi.org/10.1016/S0022-0000(05)80063-7.

[Par91] Abhay K. Parekh. Analysis of a greedy heuristic for finding small dominating sets in graphs. *Information Processing Letters*, 39(5):237–240, 1991.
https://doi.org/10.1016/0020-0190(91)90021-9.

[Pat92] Ramamohan Paturi. On the degree of polynomials that approximate symmetric boolean functions (preliminary version). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 468–474. ACM, 1992.
https://doi.org/10.1145/129712.129758.

[Păt10] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010.
https://doi.org/10.1145/1806689.1806772.

[PB83] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.
https://doi.org/10.1137/0212053.

[Pee03] René Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003. https://doi.org/10.1016/S0166-218X(03)00333-0.

[Pie03] Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and Systems Science*, 67(4):757–771, 2003. https://doi.org/10.1016/S0022-0000(03)00078-3.

[Pil18] Alexander Pilz. Planar 3-SAT with a clause/variable cycle. In David Eppstein, editor, *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 101 of *LIPIcs*, pages 31:1–31:13, Malmö, Sweden, June 2018.

[Ple79] Ján Plesník. The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. *Information Processing Letters*, 8(4):199–201, 1979.

[Plo13] Serge Plotkin. Online algorithms, 2013. http://web.stanford.edu/class/cs369/files/cs369-notes.

[Poo14] Bjorn Poonen. Undecidable problems: A sampler, 2014. https://arxiv.org/pdf/1204.0299.pdf.

[Pos46] Emil Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

[PPZ99] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999, 1999. http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html.

[PS76] Christos H. Papadimitriou and Kenneth Steiglitz. Some complexity results for the traveling salesman problem. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1976.

[PV84] Christos Papadimitriou and Umesh V Vazirani. On two geometric problems related to the travelling salesman problem. *Journal of Algorithms*, 5(2):231–246, 1984.

[PW10] Mihai Pǎtraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010. https://doi.org/10.1137/1.9781611973075.86.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computing and System Sciences*, 43(3):425–440, 1991. https://doi.org/10.1016/0022-0000(91)90023-X.

[Rab58] Michael Rabin. Recursive unsolvability of group theoretic problems. *Annals of Mathematics*, 67:172–194, 1958.

[Rab80] Michael Rabin. A probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.
http://www.cs.umd.edu/gasarch/BLOGPAPERS/rabinprime.pdf.

[Raz92] Alexander Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106:385–390, 1992.

[Raz95] Ran Raz. A parallel repetition theorem. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 447–456. ACM, 1995.
https://doi.org/10.1145/225058.225181.

[Raz98] Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.
https://dl.acm.org/doi/10.1145/225058.225181.

[Ren00a] Paul Rendell. A Turing macine in Conway's game of life, 2000.
https://www.ics.uci.edu/welling/teaching/271fall09/Turing-Machine-Life.pdf.

[Ren00b] Paul Rendell. A Turing macine in Conway's game of life:implementation, 2000.
http://rendell-attic.org/gol/tm.htm.

[Ric53] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the the American Mathematics Society*, 74:358–366, 1953.

[Ric08] Elaine Rich. *Automata, computability, and complexity: theory and application*. Prentice Hall, 2008.

[Rob83] John M. Robson. The complexity of Go. *Proceedings of the International Federation for Information Processing*, pages 413–417, 1983.

[Rob84] John M. Robson. N by N checkers is Exptime complete. *SIAM Journal on Computing*, 13:252–267, 1984.
https://epubs.siam.org/doi/10.1137/0213018.

[Roe06] Thomas A. Roemer. A note on the complexity of the concurrent open shop problem. *J. Sched.*, 9(4):389–396, 2006.
https://doi.org/10.1007/s10951-006-7042-y.

[Ros73] Arnold L. Rosenberg. On the time required to recognize properties of graphs: a problem. *SIGACT News*, 5(4):15–16, 1973.

[Rot13a] Thomas Rothvob. The Lasserre hierachy in approximation algorithms, 2013. https://sites.math.washington.edu//rothvoss/lecturenotes/lasserresurvey.pdf.

[Rot13b] Thomas Rothvoß. Approximating bin packing within o(log OPT * log log OPT) bins. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 20–29. IEEE Computer Society, 2013. https://doi.org/10.1109/FOCS.2013.11.

[RSST97] Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four color theorem. *Journal of Combinatorial Theory (Series B)*, 70:2–44, 1997.

[RU81] Kari-Jouko Räihä and Esko Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theor. Comput. Sci.*, 16:187–198, 1981. https://doi.org/10.1016/0304-3975(81)90075-X.

[Rub18] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1260–1268. ACM, 2018. https://doi.org/10.1145/3188745.3188916.

[RV78] R. Rivest and S. Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3:371–384, 1978.

[RV13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524. ACM, 2013. https://doi.org/10.1145/2488608.2488673.

[RVW18] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6):41:1–41:24, 2018. https://dl.acm.org/doi/10.1145/3232536.

[RW90] Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990. https://dl.acm.org/doi/10.1016/S0747-7171%2808%2980001-6.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[SC79] Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, May 1979. http://www.oocities.org/stockmeyer@sbcglobal.net/games.pdf.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, San Diego, California, May 1978.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.
https://doi.org/10.1016/0304-3975(87)90064-8.

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In David S. Johnson, editor, *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148, San Francisco, January 1990.

[Sch12] Marcus Schaefer. *Thirty essays on geometric graph theory*, chapter Realizability of graphs and linkages, pages 461–482. Springer, 1st edition, 2012.

[SEO03] Michael A. Soss, Jeff Erickson, and Mark H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry*, 26(3):235–246, 2003.
https://doi.org/10.1016/S0925-7721(02)00156-6.

[Ser78] Anatoliy Serdyukov. On some extremal walks in graphs (in Russian). *Upravlyaemye Sisetmy*, 17:76–79, 1978.

[Sey95] Paul D. Seymour. Packing directed circuits fractionally. *Comb.*, 15(2):281–288, 1995.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.
https://doi.org/10.1109/SFCS.1994.365700.

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999.
https://doi.org/10.1137/S0036144598347011.

[Sie72] Carl Ludwig Siegel. Zur theorie der quadratischen formen. *Nachrichten der Akademie der Wissenschaften in Göttingen, Mathematisch-Physikalische Klasse*, 3:21–46, 1972.

[Sim77] Janos Simon. On the difference between one and many. In *Automata, Languages, and Programming (ICALP 1977)*, volume 52 of *Lecture Notes in Computer Science*, pages 480–491. Springer-Verlag, 1977.
https://www.cs.umd.edu/users/gasarch/BLOGPAPERS/sharp.pdf.

[Sip83] Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 330–335. ACM, 1983.
https://doi.org/10.1145/800061.808762.

[Sla97] Peter Slavík. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997.
https://doi.org/10.1006/jagm.1997.0887.

[Soa96] Robert Soare. Computability and recursion. *Bulletin of Symbolic Logic*, 2(4), 1996.

[Soa13] Robert Soare. Why Turing's thesis is not a thesis. In *Turing centenary volume*. Cambridge University Press, 2013.

[SP12a] Bernd Steinbach and Christian Posthoff. Extremely complex 4-colored rectangle-free grids: Solution of open multiple-valued problems. In D. Michael Miller and Vincent C. Gaudet, editors, *42nd IEEE International Symposium on Multiple-Valued Logic, ISMVL 2012, Victoria, BC, Canada, May 14-16, 2012*, pages 37–44. IEEE Computer Society, 2012. https://doi.org/10.1109/ISMVL.2012.12.

[SP12b] Bernd Steinbach and Christian Posthoff. The solution of ultra large grid problems. In *21st International Workshop on Post-Binary USLI Systems*, 2012.

[SP12c] Bernd Steinbach and Christian Posthoff. Utilization of permutation classes for solving extremely complex 4-colorable rectangle-free grids. In *Proceedings of the IEEE 2012 international conference on systems and informatics*, 2012.

[SP13a] Bernd Steinbach and Christian Posthoff. Rectangle-free colorings of extremely complex grids using 4 colors. *J. Multiple Valued Log. Soft Comput.*, 21(5-6):609–625, 2013.
http://www.oldcitypublishing.com/journals/mvlsc-home/mvlsc-issue-contents/mvlsc-volume-21-number-5-6-2013/mvlsc-21-5-6-p-609-625/.

[SP13b] Bernd Steinbach and Christian Posthoff. Solution of the last open four-colored rectangle-free grid: An extremely complex multiple-valued problem. In *43rd IEEE International Symposium on Multiple-Valued Logic, ISMVL 2013, Toyama, Japan, May 22-24, 2013*, pages 302–309. IEEE Computer Society, 2013.
https://doi.org/10.1109/ISMVL.2013.51.

[SP15] Bernd Steinbach and Christian Posthoff. The last unsolved four-colored rectangle-free grid: The solution of extremely complex multiple-valued problems. *J. Multiple Valued Log. Soft Comput.*, 25(4-5):461–490, 2015.
http://www.oldcitypublishing.com/journals/mvlsc-home/mvlsc-issue-contents/mvlsc-volume-25-number-4-5-2015/mvlsc-25-4-5-p-461-490/.

[SRG20] Matthew Stephenson, Jochen Renz, and Xiaoyu Ge. The computational complexity of angry birds. *Artif. Intell.*, 280:103232, 2020.
https://doi.org/10.1016/j.artint.2019.103232.

[SS13] Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1698–1709. SIAM, 2013.
https://doi.org/10.1137/1.9781611973105.122.

[SS17] Marcus Schaefer and Daniel Stefankovic. Fixed points, nash equilibria, and the existential theory of the reals. *Theory Comput. Syst.*, 60(2):172–193, 2017.
https://doi.org/10.1007/s00224-015-9662-0.

[SSvR11] Allan Scott, Ulrike Stege, and Iris van Rooij. Minesweeper may not be NP-complete but it is hard nontheless. *The Mathematical Intelligencer*, 33:5–17, 2011.
http://www.cs.umd.edu/gasarch/BLOGPAPERS/msnotnpc.pdf.

[ST00] Michael Soss and Godfried T. Toussaint. Geometric and computational aspects of polymer reconfiguration. *Journal of Mathematical Chemistry*, 27(4):303–318, 2000.

[ST13] Uwe Schöning and Jacob Toran. *The satisfiability problem*. Lehaman's Media, 2013.

[Ste14] Bernd Steinbach, editor. *Recent progress in the boolean domain*. Cambridge Scholars Publishing, Newcastle upon Tyne,UK, 2014.

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, October 1976.

[Str80] Walter Stromquist. How to cut a cake fairly. *American Mathematics Monthly*, 87(8):640–644, 1980.

[SU08] Marcus Schaefer and Christopher Umans. Completeness in the Polynomial-Time hierarchy, 2008.
http://ovid.cs.depaul.edu/documents/phcom.pdf.

[Sun20] Zhi-Wei Sun. Further results on Hilbert's tenth problem. *Science China Mathematics*, This Journal Does not have Volumes:1–26, 2020.

[SY11] Arezou Soleimanfallah and Anders Yeo. A kernel of order $2k - c$ for vertex cover. *Discrete Mathematics*, 311(10-11):892–895, 2011.
https://doi.org/10.1016/j.disc.2011.02.014.

[SZ06] Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *INFOCOMP Journal of Computer Science*, pages 25–28, 2006.
http://arxiv.org/abs/cs/0512049.

[SZZ18] Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. Ppp-completeness with connections to cryptography. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 148–158. IEEE Computer Society, 2018.
https://arxiv.org/abs/1808.06407.

[Tar48] Alfred Tarski. A decision method for elemenary algebra and geometry, 1948. http://www.rand.org/content/dam/rand/pubs/reports/2008/R109.pdf.

[Tho78] Andrew B. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Annals of Discrete Mathematics*, 3:259–268, 1978.

[Tip16] Simon Tippenhauer. *On planar 3-SAT and its variants.* PhD thesis, Fachbeheich Mathematik and Informatik der Freien Universitat Berlin, 2016. https://www.cs.umd.edu/gasarch/BLOGPAPERS/Tippenphd.pdf.

[Tod91] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal of Computing*, 20(5):865–877, 1991. https://doi.org/10.1137/0220053.

[Tov84] Craig Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8:85–89, 1984. https://www.sciencedirect.com/science/article/pii/0166218X84900817.

[Tra84] Boris Trakhenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6:384–400, 1984.

[Tre11] Luca Trevisan. Lecture 8: A linear programming relaxation of set cover, 2011. http://theory.stanford.edu/trevisan/cs261/lecture08.pdf.

[TSSW00] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal of Computing*, 29(6):2074–2097, 2000. https://doi.org/10.1137/S0097539797328847.

[Tut46] William T Tutte. On Hamiltonian circuits. *The Journal of the London Mathematical Society*, pages 98–101, 1946.

[Tut56] William T Tutte. A theorem on planar graphs. *Transactions of the American Mathematical Society*, pages 99–116, 1956. https://www.ams.org/journals/tran/1956-082-01/S0002-9947-1956-0081471-8/S0002-9947-1956-0081471-8.pdf.

[UL97] Christopher Umans and William Lenhart. Hamiltonian cycles in solid grid graphs. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 496–505. IEEE, 1997. https://ieeexplore.ieee.org/document/646138.

[Und78] P. Underground. On graphs with hamiltonian squares. *Discrete Mathematics*, 21(3):323, 1978.
https://doi.org/10.1016/0012-365X(78)90164-4.

[US15] Akihiro Uejima and Hiroaki Suzuki. Fillmat is NP-complete and APSP-complete. *J. Inf. Process.*, 23(3):310–316, 2015.
https://doi.org/10.2197/ipsjjip.23.310.

[Vad01] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.
https://doi.org/10.1137/S0097539797321602.

[Val79a] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
http://dx.doi.org/10.1016/0304-3975(79)90044-6.

[Val79b] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
http://dx.doi.org/10.1137/0208032.

[Val79c] Leslie G. Valiant. Negative results on counting. In *Theoretical Computer Science, 4th GI-Conference, Aachen, Germany, March 26-28, 1979, Proceedings*, pages 38–46, 1979.

[Vaz01] Vijay Vazirani. *Approximation Algorithms*. Springer, 2001.

[vEB81] Peter van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Tech Report 8104, University of Amsterdam, Dept of Mathematics, Neverthlands, 1981.

[Vig12] Giovanni Viglietta. Hardness of mastermind. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 2012.

[Vig14] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.
http://dx.doi.org/10.1007/s00224-013-9497-5.

[Viz64] Vadim Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 3:25–30, 1964.

[vKdB91] Marc J. van Kreveld and Mark de Berg. Finding squares and rectangles in sets of points. *BIT*, 31(2):202–219, 1991.
https://doi.org/10.1007/BF01931281.

[VW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the Association of Computing Machinery (JACM)*, 65(5):27:1–27:38, 2018.
https://doi.org/10.1145/3186893.

[VY11] Elad Verbin and Wei Yu. The streaming complexity of cycle counting, sorting by reversals, and other problems. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 11–25. SIAM, 2011. https://doi.org/10.1137/1.9781611973082.2.

[Wag13] Samuel Wagstaff. *The joy of factoring*. AMS, Providence, 2013.

[Wan60] Hao Wang. Proving theorems by pattern recognition I. *Communications of the ACM*, 3(4):220–234, 1960. https://doi.org/10.1145/367177.367224.

[Wika] Wikipedia. Discrete logarithm. https://en.wikipedia.org/wiki/Discretelogarithm.

[Wikb] Wikipedia. Eternity II puzzle. https://en.wikipedia.org/wiki/EternityIIpuzzle.

[Wikc] Wikipedia. Eternity puzzle. https://en.wikipedia.org/wiki/Eternitypuzzle.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. https://doi.org/10.1016/j.tcs.2005.09.023.

[Wil18] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. https://doi.org/10.1137/15M1024524.

[WS11] David Williamson and David Shmoys. *Design of Approximation Algorithms*. Cambridge University Press, 2011.

[Yao77] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227. IEEE Computer Society, 1977. https://doi.org/10.1109/SFCS.1977.24.

[Yat00] Takayuki Yato. On the NP-completeness of the Slither link puzzle. *IPSJ SIGNotes ALgorithms*, 74:25–32, 2000.

[YG80] Mihalis Yannakakis and Fanica Gavril. Edge domination sets in graphs. *SIAM Journal of Applied Mathematics*, 38(3):364–372, 1980.

[YJ94] Lin Yixun and Yuan Jingiang. Profile minimization problem for matrices and graphs. *Acta Mathematicae Applicattae Sinica*, 10:107–112, 1994. https://link.springer.com/content/pdf/10.1007/BF02006264.pdf.

[YLLW98] Jinjang Yuan, Yixun Lin, Yan Liu, and Shiying Wang. NP-completeness of the profile problem and the fill-in problem on co-bipartite graphs. *Journal of Mathematical Studies*, Project: Labeling and Embedding of Graphs, 1998.

[YS03] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles, 2003.

[YSI05] Takayuki Yato, Takahiro Seta, and Tsuyoshi Ito. Finding yozume of generalized tsume-shogi is exptime-complete. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 88-A(5):1249–1257, 2005.
https://doi.org/10.1093/ietfec/e88-a.5.1249.

[Yu16] Jingjin Yu. Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics Autom. Lett.*, 1(1):33–40, 2016.
https://doi.org/10.1109/LRA.2015.2503143.

[Zuc07] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007.
http://theoryofcomputing.org/articles/v003a006/index.html.

[Zwi98] Uri Zwick. All pairs shortest paths in weighted directed graph $\frac{3}{4}$ exact and almost exact algorithms. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 310–319. IEEE Computer Society, 1998.
https://doi.org/10.1109/SFCS.1998.743464.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.
https://doi.org/10.1145/567112.567114.