Algorithms and Resource Requirements for Fundamental Problems

R. Ryan Williams

August 2007

CMU-CS-07-147

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Manuel Blum, Chair
Ryan O'Donnell
Steven Rudich
Russell Impagliazzo (UCSD)
Dieter van Melkebeek (U. Wisconsin-Madison)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant no. CCR-0122581 and fellowship no. DGE-0234630. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: satisfiability, lower bounds, time-space tradeoffs, automated theorem-proving, exact algorithms, maximum cut

to my parents

Abstract

We establish more efficient methods for solving interesting classes of NP-hard problems exactly, as well as methods for proving limitations on how quickly those and other problems can be solved.

- On the negative side, we prove that a number of NP-hard problems cannot be solved too efficiently by algorithms that only use a small amount of additional workspace. Building on prior work in the area, we prove that the Boolean satisfiability problem and other hard problems require $\Omega(n^{2\cos(\pi/7)-o(1)}) \geq \Omega(n^{1.801})$ time to solve by any algorithm that uses $n^{o(1)}$ space. Stronger lower bounds are proved for solving quantified Boolean formulas with a fixed number of quantifiers. Our results are essentially model-independent, in that they hold for all reasonable random-access machine models. Furthermore, we present a formal proof system that captures all prior time-space lower bounds for satisfiability (including our own), and demonstrate how the search for better lower bounds can be *automated*, in not only our particular setting but also other lower bounds that follow a certain high-level pattern. We describe an implementation of an automated theorem prover and provide experimental results which strongly suggest that further improvements on the above time lower bound will require new tools and ideas.

- On the positive side, we give a general methodology for solving a large class of NP-hard problems much faster than exhaustive search. In particular, for a problem in the class where exhaustive search of all possible solutions takes $\Theta(N)$ time, our algorithm solves the problem in $O(N^{\delta})$ time, for a universal constant $\delta < 0.792$ that depends on the complexity of multiplying two matrices over a ring. We also provide theoretical evidence that a much larger class of problems admits a similar type of algorithm.

To illustrate our results, consider the MAX CUT problem, where one is given a graph $G = (V, E)$ and integer K, and one wishes to determine if G has a subset of vertices such that the number of edges leaving the subset is at least K. The obvious algorithm for MAX CUT runs in $O(\text{poly}(n) \cdot 2^n)$ time, where $n = |V|$. Despite the problem's importance, no better algorithm was known for the general case of MAX CUT, prior to our work. Our results imply that MAX CUT *can* be solved in $O(\sqrt{3}^n)$ time but *cannot* be solved in $O(n^{1.801})$ time and $n^{o(1)}$ space.

Acknowledgements

Any success I have had as a graduate student owes credit to my advisor, Manuel Blum. He is one of the most generous people I have ever met, and he maintains an unwavering focus on the truly important problems. He has helped me learn that optimism and bravery are paramount in research (and life as well– but I'm being redundant). I owe many thanks to the rest of my thesis committee (Ryan, Steven, Russell, and Dieter) too, for their all of their gracious help and penetrating insights which improved this work tremendously.

Here at Carnegie Mellon, I've made many friends that I'm grateful to have been around. In particular, if it were not for Virginia Vassilevska, I might have truly lost my mind at certain stages of my thesis work. Her constant cheer and unconditional support kept me going. Maverick Woo has been a great buddy since freshman year at Cornell – he and Manuel smooth-talked me into Carnegie Mellon, and I'm so glad they did. I also thank Sue Ann, Himanshu, Pedro, Vlad, and Mahim for tolerating me as an officemate, and Bubba Beasley for his camaraderie in watching college football.

From the times of my undergraduate and M.Eng at Cornell, I am most grateful to Juris Hartmanis for his highly influential mentoring, along with the dynamic duo of Carla Gomes and Bart Selman for their wonderful support over the years. And whereas I am graduating, be it resolved that everyone affiliated with Telluride House from 2000 to 2002 should take this acknowledgement as an excuse to toast themselves, wherever they may be. Special thanks are due to Intergraph Corporation for their use of me as a COBOL-ing Y2K summer intern– that experience drove me to research as much as anything else.

Of course I should also mention my family– there, I just did. Seriously, I don't know why Mom and Dad let me go all the way to yankeefied Ithaca, New York for college, even with my skills of persuasion ("no, it isn't pronounced EYE-THACKA"), but their decision changed my life for the better. My brother and sister have always been there for me, through the good, the bad, and the XBox.

There are so many other people to thank, and with another hundred pages I would only have listed a tiny fraction of them. (*For instance, I can imagine that around page 79, I'd start thanking the 2004 NCAA Football National Champions of Auburn University one by one, for their inspirational perfect season that fictionally led me to prove Corollary 58 of my 4th paper, bla bla bla...*) Permit me to thank you all at once, and I sincerely hope you enjoy reading.

Contents

Chapter 1

Introduction

Computational complexity theory is the formal study of how resource limitations affect computational ability, both negatively and positively. Its mathematical framework unites the elegance of recursive function theory (which studies what can be computed in principle) with the highly constrained reality of our world. The resources most widely considered are *time* and *space*, where the primary questions are "what can be computed within a short amount of time?" and "what can be computed with a small amount of storage?".

Through years of research, practitioners of complexity theory have identified leagues of various *complexity classes* that intuitively capture those problems which are solvable in an efficient amount of time and space. Four central classes among these are P, NP, L, and PSPACE.

- For the resource of time, the class P contains those problems which can be solved by a computer in polynomial time. That is to say, any input instance of the problem can be solved in an amount of time that is not extravagantly longer than the length of the input itself.
- The class NP contains those problems whose solutions can be verified by a computer in polynomial time. That is, if one is given the solution to a problem upfront, then the solution can be efficiently verified as a valid one. It is easy to see that $\mathsf{P} \subseteq \mathsf{NP}$, since any problem that can be *solved* in polynomial time can also have its solutions *verified as correct* in polynomial time. The question of whether or not $\mathsf{P} = \mathsf{NP}$ is among the most important problems in mathematics– it is one of the seven Clay Math Millennium Problems, for which a solution merits \$1 million USD [CJW06]. It is widely believed that $\mathsf{P} \neq \mathsf{NP}$.
- For the resource of space, the class L (also known as $\mathsf{LOGSPACE}$) contains those problems which can be solved by a computer whose working storage size is roughly the logarithm of the input size. Problems in L can be solved with very little extra storage, beyond that used to house the input.
- Similarly, the class PSPACE contains those problems solvable by a computer with a polynomial-sized working storage. PSPACE is the analogue of P in the space-bounded setting.

It is well-known that $\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$, but also that $\mathsf{L} \neq \mathsf{PSPACE}$. Therefore, we know that either $\mathsf{L} \neq \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{PSPACE}$, but we do not know how to prove either of the two inequalities.

The $\mathsf{L} \mathrel{?}= \mathsf{NP}$ and $\mathsf{P} \mathrel{?}= \mathsf{PSPACE}$ problems are also major open problems in complexity theory, and since most researchers believe $\mathsf{P} \neq \mathsf{NP}$, it is widely conjectured that both $\mathsf{L} \neq \mathsf{NP}$ and $\mathsf{P} \neq \mathsf{PSPACE}$.

Each of the separation problems $\mathsf{P} \mathrel{?}= \mathsf{NP}$, $\mathsf{L} \mathrel{?}= \mathsf{NP}$, and $\mathsf{P} \mathrel{?}= \mathsf{PSPACE}$ are fundamental to our understanding of feasible computation. From time to time, it is sometimes said that despite their past significance, these problems are no longer relevant from a practical standpoint, and amount to mere puzzles for pure mathematicians. This viewpoint is badly mistaken. These problems form the underpinning of all our shared knowledge about what is possible with computers and what is not. Practically every prominent area of computer science has a share of expressive problems known as NP-*complete* and PSPACE-*complete* problems, many of which are central to their area. A complete problem in a class $\mathcal{C}$ has the remarkable property that every other problem in $\mathcal{C}$ can be efficiently expressed as a special case of the complete problem. Therefore if an efficient algorithm exists for a $\mathcal{C}$-complete problem, then there is an efficient algorithm for *every* problem in $\mathcal{C}$. A demonstration that a problem is NP-complete or PSPACE-complete is strong evidence that the problem is difficult to solve– for if not, then a major breakthrough would occur in complexity theory.

When a new area in computer science arises, one of the first orders of business is often to figure out what's NP-complete and what's not. (We have personally experienced this in database research, where a data privacy problem turned out to be unexpectedly NP-complete [MW05], forcing us to design heuristic approximations for it.) The abundance of complete problems has forced thousands of computer science workers to refine their research agendas in the hopes of working around them. As purely mathematical problems, P vs NP and her relatives are already among the most celebrated and sought-after. But as a computer science problem, P vs NP has attained the dual stature of being both a fundamental physical law and an insurmountably unverifiable religious belief. No patch of the computer science landscape can escape the long shadow it has cast, and yet we do not have the slightest idea of how to prove that $\mathsf{P} \neq \mathsf{NP}$ despite our firmest intuitions that it must be true.

1.1 Lower Bounds for Solving NP Problems

The L versus NP question is not quite as famous as P vs NP, but it is still extremely important for our understanding of the deep difference between *proof verification* versus *proof generation.* For example, given a Boolean formula and an assignment to its variables, one can check if the assignment makes the formula true, using only logarithmic space. However, the problem of devising such an assignment from scratch (*i.e.* the satisfiability problem, abbreviated as SAT) requires at least the power of NP. How difficult are these two problems, comparatively? With a realistic computing model, the first problem can be solved in $O(n \cdot \text{poly}(\log n))$ time and $O(\log n)$ space. In contrast, a precise classification of the time-space complexity of the second problem would settle the L vs NP question.

We give some partial progress towards the separation of L and NP, building on the pioneering work of others from the late 1990's. Our main result is that SAT requires at least $n^{2\cos(\pi/7)-\varepsilon} \geq n^{1.8019}$ time for any realistic algorithm that uses $n^{o(1)}$ space, for all $\varepsilon > 0$. Therefore, there is a substantial difference between the time-space complexity of checking if a particular assignment makes a formula true, and checking if *no assignment at all* makes the formula true. (Note that a full proof that $\mathsf{L} \neq \mathsf{NP}$ would be equivalent to proving that SAT requires at least n^k time and

$O(\log n)$ space, for *all* $k > 1$.) In fact, we can show a *time-space tradeoff*: for every $a < 2\cos(\pi/7)$, there is a $b > 0$ such that SAT cannot be solved in n^a time and n^b space. This improves upon previous results by Fortnow, Lipton, Van Melkebeek, and Viglas, who showed an $n^{\phi} \geq n^{1.618}$ time lower bound in the same setting.

Our time-space lower bound for SAT holds for many other well-studied NP-complete problems such as VERTEX COVER, HAMILTON PATH, and MAX 2-SAT. We also generalize our results to prove new lower bounds for the Quantified Boolean Formula (QBF) problem on instances with a fixed number of quantifiers, showing that quantified Boolean formulas with k quantifier blocks require at least $\Omega(n^{k+1-\varepsilon_k})$ time to solve for any realistic algorithm using $n^{o(1)}$ space, where $\varepsilon_k \to 0$ as $k \to \infty$. The overall proof strategy used is very general, and can be adapted to improve upon superlinear time limitations for SAT on other computational models as well, such as off-line one-tape Turing machines. (However, since the off-line one-tape machine is not considered to be a very general computational model, these results will not be the main focus in this thesis.) Our high-level strategy follows prior work in striving for proofs by contradiction: we show that one can apply an ultra-efficient satisfiability algorithm to build new algorithms that we already know cannot exist in computational complexity. Thus, while our approach yields negative results, the proofs of results are algorithmic in nature. In this way we use old non-existence proofs to help us prove new ones.

For a long time, we believed that it should be possible to extend our ideas to prove an $\Omega(n^{2-\varepsilon})$ time lower bound for solving SAT with $n^{o(1)}$ space on a random access machine. Indeed, this was a folklore conjecture among most researchers studying time-space tradeoffs. A quadratic time lower bound is already known for multitape Turing machines with $n^{o(1)}$ space (cf. Santhanam [San01]), but that result crucially depends on the access limitations of tapes and does not generalize. However, we now have evidence that any time lower bound better than ours will require substantially new ideas. After countless days of failing to find a better lower bound than our $n^{1.8019}$ result, we discovered a way to efficiently formalize the high-level strategy taken by all prior results (including our own), so that a computer program can automatically and *feasibly* find proofs of any lower bound that falls within the existing framework. This framework includes a host of lower bound arguments, including time limitations for solving problems on multitape machines, off-line one-tape machines, and nondeterministic space-bounded RAMs.

We have implemented a small-scale version of a program that can automatically discover new time lower bounds. Our computer experiments suggest a surprising and challenging conjecture: that our $\Omega(n^{2\cos(\pi/7)-\varepsilon})$ lower bound is actually the best possible using the existing tools. In particular, a variety of possible proof strategies (as formalized in our approach) all lead to lower bound exponents below 1.8019, and exhaustive search of all proofs up to a certain length indicates that the best possible proofs have a very similar form to our proof strategy achieving the $2\cos(\pi/7)$ exponent. Therefore, to improve upon our work in this area, it appears that truly new ideas are required– better results do not seem possible with the existing tools.

1.2 Upper Bounds for Solving NP Problems

In the second part of this thesis, we investigate the extent to which NP-complete problems *can* be solved exactly. While the general consensus is that no polynomial time algorithms exist for these problems, it is perhaps less clear that subexponential ($2^{n^{o(1)}}$ time) algorithms do not exist. For some problems, we know of no better algorithms than the trivial brute-force search that tries all possible solutions. It is of great intellectual interest to know if solution enumeration is truly the best one can do for NP problems in general, and indeed this question was one of the initial motivations for the P vs NP question. However, just like the lower bound setting, it seems very difficult to find good *general* methods– substantially better algorithms that work for a variety of hard problems. Our notion of "substantially better" is the following: we define an *accelerated algorithm* for a problem to be one that solves any instance of the problem in $O(N^{\delta})$ time for some $\delta < 1$, where N is the number of possible solutions. This is a much weaker requirement than showing $\mathsf{P} = \mathsf{NP}$ – in that case, we would be solving instances in $O((\log N)^c)$ time – but the question of whether there are accelerated algorithms for NP problems is still a mathematically deep one that has the potential for practical impact. In the second half of this thesis, we report progress on the search for accelerated algorithms for general NP-hard problems.

Our work in this direction is motivated by a general research question: computer science has uncovered remarkably efficient algorithms for many problems in P, such as matrix multiplication, integer/polynomial multiplication, and minimum spanning trees – what implications do these ingenious algorithms have for solving NP-hard problems? Our approach is to reformulate an NP problem in such a way that it becomes *exponentially larger* than before, but now solvable in time *polynomial* in its new size. Then we apply a fast polynomial time algorithm to the new exponential-size problem, obtaining an accelerated algorithm. For example, a trivial reformulation would be to append a list of all possible solutions to the input– a solution can now be found in "polynomial time" by checking each possible solution produced, but of course the resulting algorithm yields no improvement over brute-force search. We use more intricate reductions combined with a polynomial time problem having a surprisingly fast algorithm, *matrix multiplication*, the problem of multiplying two $n \times n$ matrices over a ring. For years, people believed that the obvious $O(n^3)$ algorithm for matrix multiplication was the fastest possible– that all possible pairs of rows and columns had to be multiplied separately in $\Theta(n)$ time. This intuition was shattered with Strassen's breakthrough algorithm which uses only $O(n^{\log_2 7})$ ring operations [Str69]. The current fastest (ring) matrix multiplication algorithm is by Coppersmith and Winograd [CW90] and runs in $O(n^{2.376})$ ring operations.

The main positive result of this work is that a large class of NP-hard problems can be solved significantly faster than exhaustive search, by connecting the standard brute-force algorithm for these problems to one that can be solved quickly with fast matrix multiplication. The "large class" is called $\mathsf{Weighted\ 2\text{-}CSP}$, and it consists of constraint satisfaction problems having at most two variables per constraint, with small weights on the variables and constraints. The problem includes many well-studied optimization problems as special cases, such as Max Cut, Max 2-Sat, Sparsest Cut, and Min Bisection. We prove that, if there is a matrix multiplication algorithm for $N \times N$ matrices that runs in $O(N^{\omega})$ time, then instances of Weighted 2-CSP with n variables can be solved in $O(\mathrm{poly}(n) \cdot 2^{\frac{\omega n}{3}})$ time, whereas $O(2^n)$ is the runtime of exhaustive search. Hence, a fast algorithm for an *easy* problem can be used to design accelerated algorithms for Max Cut, Max 2-Sat, Sparsest Cut, and Min Bisection, which are all *difficult* problems. Prior to our work, none of the above four problems were known to have better worst case algorithms than the trivial one. While our algorithms are still exponential, and rely on fast matrix multiplication algorithms which are not yet practical, it is nevertheless intellectually interesting that something better than solution enumeration is possible.

Finally, we study the question of whether there exists an accelerated algorithm for SAT for formulas in conjunctive normal form. We give lines of attack that suggest that an accelerated SAT algorithm may be possible, for formulas where the number of clauses is bounded by a polynomial in the number of variables. Similar to the results of the previous paragraph, we give connections between SAT and three polynomial time solvable problems, showing that if the polynomial time problems have sufficiently faster algorithms, then SAT has an accelerated algorithm. One of these problems is k-DOMINATING SET, which is to determine if a given graph contains a k-set S of vertices such that every node is either in S or has a neighbor in S. We show that if there exists a $k \geq 3$ and $\varepsilon > 0$ such that k-DOMINATING SET is in $O(n^{k-\varepsilon})$ time, then SAT is in $O\left(\text{poly}(m,n)2^{\delta_\varepsilon n}\right)$ time, for some $\delta_\varepsilon < 1$. (It is known that, for $k > 7$, k-DOMINATING SET is in $n^{k+o(1)}$ time.) We are currently developing an abstract notion of our reductions, in the hopes of defining a complexity class that captures the underlying phenomenon.

1.3 Outline and Bibliographic Information

This thesis is structured as follows. The next chapter provides some background in computational complexity. Chapter 3 introduces prior work on time-space tradeoffs, including their basic tools and the style of argument used by them. Chapter 4 discusses our time-space lower bounds for SAT, giving a chronological account of successively better time lower bounds for SAT and other hard problems on random access machines with subpolynomial ($n^{o(1)}$) space. Most of the results in this chapter appear in the papers [Wil05b, Wil07]. Preliminary versions of these papers appeared in the 20th and 22nd Annual IEEE Conference on Computational Complexity; both received the Ronald V. Book Best Student Paper award. Chapter 5 describes our automated approach to proving lower bounds, giving a formal model with which old lower bound proofs can be verified and new lower bounds can be proved.

Chapter 6 turns to the task of finding *accelerated algorithms* for NP problems, deriving a general method for solving WEIGHTED 2-CSP and its variants, which capture practically any combinatorial optimization problem whose constraints can be represented by degree-two polynomials. The results in this chapter extend our prior work presented in [Wil05a]. A preliminary version was presented at the 31st International Colloquium on Automata, Languages, and Programming, and received the Best Student ICALP Paper award. Chapter 7 discusses the possibility of an accelerated algorithm for SAT. Three plausible hypotheses are given, each of which would imply the existence of a SAT algorithm substantially faster than brute-force search. The final chapter outlines an assortment of future goals and directions for further work.

Chapter 2

Background

In this chapter, we review some basic facts from algorithms and complexity theory that are required for our work. Our treatment is neither intended to be completely rigorous, nor rigorously complete. For further background, we invite the reader to try Papadimitriou's *Computational Complexity* [Pap94].

2.1 Asymptotics

We start with a quick review of some asymptotic notation. Assume $f, g : \mathbb{N} \to \mathbb{N}$.

- f is $O(g)$ *if and only if* there are $c_1, c_2 \geq 0$ such that for all $n \geq 1$, $f(n) \leq c_1 g(n) + c_2$. Thus f is "bounded from above" by g, modulo constants.

- f is $\Omega(g)$ *if and only if* g is $O(f)$. So f is "bounded from below" by g.

- f is $\text{poly}(g)$ *if and only if* there is a constant $c \geq 1$ so that f is $O(g^c)$. For instance, n^{10} is $\text{poly}(n^2)$.

- f is $O^*(g)$ *if and only if* f is $O(\text{poly}(n) \cdot g)$. In other words, the O^* omits polynomial factors from the runtime bound. This is a convenient notation for expressing exponential bounds.

  Example. $2^n n^5 \log n + 1.9^n$ is $O^*(2^n)$.

- f is $o(g)$ *if and only if* $\lim_{n\to\infty} f(n)/g(n) = 0$.

  Example. A function f is $o(1)$ when $\lim_{n\to\infty} f(n) = 0$, and f is $n^{o(1)}$ if f is $O(n^{\varepsilon})$ for all $\varepsilon > 0$. The functions $\text{poly}(\log n)$, $2^{\sqrt{\log n}}$, and $(\log n)^{\log\log n}$ are all $n^{o(1)}$, but $n^{1/100}$ is not $n^{o(1)}$.

2.2 Boolean Formulas

The concept of a Boolean formula is central to our work. To ensure our notation is clear, we review some elementary notions. For us, Boolean variables take the value either 0 or 1; as we consider finite sets of Boolean variables, we typically name variables to be $x_1, \ldots, x_n$ for some positive integer n.

Three operators on variables are paramount: the NOT, AND, and OR operators, which are written formally as $\neg$, $\wedge$. and $\vee$, respectively.

Definition 2.2.1 *We define the set of Boolean formulas over n variables inductively:*

- *For all $i = 1, \ldots, n$, the variable x_i is a formula.*
- *If ϕ and ψ are formulas, then $\neg\phi$, $(\phi \wedge \psi)$, and $(\phi \vee \psi)$ are also formulas.*

The size *of a Boolean formula is the total number of variables and occurrences of $\neg$, $\wedge$, and $\vee$ in the formula.*

We extensively study special types of formulas. A *literal* is a formula of the form x_i or $\neg x_i$, for some variable x_i. A *clause* or *disjunction of literals* is a formula of the form $(\ell_1 \vee \cdots \vee \ell_k)$ for some literals ℓ_i. Similarly, a *conjunction of literals* is of the form $(\ell_1 \wedge \cdots \wedge \ell_k)$.

An extremely convenient representation for Boolean formulas is *conjunctive normal form*, abbreviated as CNF. A Boolean formula is in CNF if and only if it is of the form

$$(c_1 \wedge c_2 \wedge \cdots \wedge c_m),$$

where the c_i are clauses. That is, a CNF formula is an AND of ORs. It is well-known that every function $f : \{0,1\}^n \to \{0,1\}$ can be written in conjunctive normal form. However the size of this representation is often exponential in the number of variables. If a formula is in CNF and has at most k literals in each clause, we further say that the formula is in k-CNF.

Similarly, any Boolean function can be represented in *disjunctive normal form* (DNF), which is

$$(c_1 \vee c_2 \vee \cdots \vee c_m),$$

where the c_i are conjunctions (*i.e.* the formula is an OR of ANDs). As with CNF, the DNF representation of a Boolean function can be exponential in the number of variables.

2.3 The Computational Model and Computational Problems

Throughout the thesis, we assume the *random-access machine* (a.k.a. *RAM*), to be the underlying computational model. The RAM model is realistic and powerful, which makes it appealing to study in the context of algorithms, since the design of efficient algorithms becomes easier as the strength of the model increases. However in the context of lower bounds, RAM computations often become considerably more difficult to analyze than Turing machines and other weaker models. We shall not require a formal definition of a RAM, as our results hold under many perturbations of the model. The salient features of the RAM that we need are:

- it accesses its input in a read-only fashion,
- it has an auxiliary working storage of registers that it accesses in a read-write fashion, and
- it can access an arbitrary bit of the input or working storage in $O(1)$ time, independently of the last location accessed.

The third requirement is the crucial difference between RAMs and weaker computational models. The fact that a RAM can skip through its storage in constant time makes it very difficult to prove limitations on RAMs, even for those that run in linear time in the length of the input. For example, it is a celebrated open problem to determine if SAT, VERTEX COVER, or INDEPENDENT SET requires asymptotically more than *linear time* on a RAM. Our inability to resolve even this problem is a strong testament to the difficulty of the P vs NP question.

Since the above computational model is fixed throughout, all references to "algorithm" or "RAM" refer to a procedure of the above type.

The notion of a computational problem is central to computer science. A *decision problem* is a collection $\mathcal{P}$ of strings drawn from a subset $S \subseteq \{0,1\}^*$. Since our primary focus shall be on decision problems (rather than *function problems*, where the number of possible answers can be larger than two), we refer to "decision problems" as merely "problems." An *instance* of a problem is just a string from S. A string x is a *yes-instance* if $x \in \mathcal{P}$, and is a *no-instance* otherwise. We typically take $S = \{0,1\}^n$, so that strings not in S are automatically no-instances. Intuitively, the instances of a problem represent *questions*, and the questions contained in $\mathcal{P}$ are those with a yes answer.

We define the solution of a problem by an algorithm as follows:

Definition 2.3.1 *An algorithm A* solves *or* decides *$\mathcal{P}$ if for all $x \in S$,*

- *if $x \in \mathcal{P}$ then $A(x) = 1$, and*
- *if $x \notin \mathcal{P}$ then $A(x) = 0$.*

It is also common to say that algorithm A *rejects* x when $A(x) = 0$, and it *accepts* x when $A(x) = 1$. Note the "output" of an algorithm solving a computational problem is a single bit, *yes* or *no*. This limited choice of outputs is generally not a major issue; one can typically solve problems whose solutions are of arbitrary length, given an algorithm for a corresponding computational problem. Let us just give three examples of computational problems, defined by the yes-no questions they address.

- BIT-SUM: given an input (a, b, i) where a, b are two n-bit numbers and $i \in \{1, \ldots, n+1\}$, is the ith bit of $(a + b)$ a 1?
  This problem has a *linear time* algorithm: on inputs with n-bit numbers, the algorithm takes $O(n)$ steps. From such an algorithm, one can easily construct another algorithm that solves the *function problem* of printing the sum of $a + b$ on input (a, b); for example, one could run the BIT-SUM algorithm on $(a, b, 1)$, $(a, b, 2)$, $(a, b, 3)$, *etc.*

- TRIANGLE: given a graph $G = (V, E)$ as an adjacency matrix, does it contain vertices u, v, w such that $\{u, v\}, \{v, w\}, \{w, u\}$ are edges?
  Note that the input length is $N = |V|^2$. By checking every triple of vertices, TRIANGLE can be solved in $O(N^{3/2})$ steps. In Chapter 6, we use a faster algorithm for TRIANGLE as a stepping stone to building algorithms for harder-to-solve problems.

- SAT, the satisfiability problem: given a Boolean formula F in conjunctive normal form, is there an assignment to the variables of F that makes the formula true?
  As mentioned in the introduction, SAT is of paramount importance in modern computer science. Unlike the above two problems, it is unlikely to have an algorithm that always answers its question correctly in $O(n^c)$ steps, for any constant $c \geq 1$. In Chapter 4, we prove new limitations on how well SAT can be solved, and in Chapter 7, we introduce an attack that may hold promise for finding better SAT algorithms in the future.

2.4 Complexity Classes

As mentioned in the Introduction, L, P, NP, and PSPACE are classes of computational problems, defined by the amount of resources that an algorithm needs to successfully answer the questions for a problem. L is the class of problems solvable by a RAM that (on all inputs of length n) stores at most $O(\log n)$ bits in its auxiliary storage. For example, by using an algorithm that maintains a binary counter, the following problem ONES is in L:

$$\text{ONES} = \{(x, i) | \text{ the number of 1's in } x \text{ equals } i\}.$$

PSPACE is the class of problems that can be solved by a RAM that stores at most n^k bits in its auxiliary storage, for some constant $k \geq 1$. P is the class of problems that can be solved in n^k steps for some constant $k \geq 1$. NP is the class of problems for which the "yes" instances have solutions that can be verified in n^k steps for some $k \geq 1$. More precisely, a problem $\mathcal{P}$ is in NP when there is a polynomial time algorithm A and constant $c \geq 1$ whereby $x \in \mathcal{P}$ if and only if there is a $y \in \{0,1\}^{|x|^c}$ satisfying $A(xy) = 1$. Here, the y constitutes a *proof* that x is a "yes" instance of the problem, and once y is known, the proof can be checked in polynomial time via A.

The class coNP is also of significance. A problem is contained in coNP if its *complement* is in NP. (That is, for a coNP problem, the "no" instances have solutions that can be efficiently verified.) The following inclusions are straightforward:

$$\mathsf{L} \subseteq \mathsf{P} \begin{array}{c} \subseteq \mathsf{NP} \subseteq \\ \subseteq \mathsf{coNP} \subseteq \end{array} \mathsf{PSPACE}.$$

Hundreds of other complexity classes have been defined in the literature, many of which are actively studied, each of which quantifies some aspect of resource-bounded computation.[1] In this thesis we focus primarily on the above five classes, along with some other intermediate classes of problems lying between NP and PSPACE, namely the classes of the *polynomial time hierarchy*, written as $\Sigma_k\mathsf{P}$ and $\Pi_k\mathsf{P}$ for $k \geq 1$. Intuitively, the problems in $\Sigma_k\mathsf{P}$ and $\Pi_k\mathsf{P}$ are those for which the yes-instances can be defined via logical sentences with k quantifiers, where each quantifier is over polynomially

[1]For an ongoing catalog of these classes, we invite the reader to do a web search for "Complexity Zoo".

long strings and the predicate can be evaluated in polynomial time. The Σ_k classes start with an existential quantifier, and the Π_k classes start with a universal one. For example, consider the problem of determining whether a given formula is as small as it can possibly be:

FORMULA MINIMIZATION: *Given a Boolean formula F on n variables and an integer k, is there a formula F' of size at most k that agrees with F on all 2^n variable assignments?*

This problem is in $\Sigma_2\mathsf{P}$, as the yes-instances can be defined with the sentence:

$$(\exists \text{ formula } F' \ : \ |F'| < k)(\forall\ x \in \{0,1\}^n)[F(x) = F'(x)].$$

Despite our belief that normal algorithms cannot efficiently solve such problems, suppose we tried to develop a hypothetical computational model that *could* solve $\Sigma_k\mathsf{P}$ problems using only polynomial time. Understanding this hypothetical model could give us insight into the difference between $\Sigma_k\mathsf{P}$ and P. To this end, we define a nondeterministic algorithm A' for an NP problem to be a device that on input x "guesses" a proof $y \in \{0,1\}^{|x|^c}$ and runs $A(xy)$, where A is the polytime algorithm guaranteed by the definition of NP. We say that A' solves the problem if for all x there is a proper y so that $A(xy) = 1$. That is, A' accepts x if and only if there is a guess y that makes A accept xy. (The above shows, at least informally, that $\Sigma_1\mathsf{P} = \mathsf{NP}$.)

Similarly, one can define a co-nondeterministic algorithm A' as one that accepts x if and only if all guesses y make A accept. (This is tantamount to saying $\mathsf{coNP} = \Pi_1\mathsf{P}$.) Combining the two, we can also define algorithms that *alternate* between guessing nondeterministically ("existentially") and co-nondeterministically ("universally") – we call such an algorithm an *alternating machine*, and the problems solvable in polynomial time with a bounded number of alternations define the classes of the polynomial hierarchy. For more details on nondeterminism, the polynomial hierarchy, and alternating machines, the reader is invited to consult Papadimitriou [Pap94].

2.4.1 Time Bounded Classes

For each of the complexity classes discussed above, there is a counterpart that only considers fixed polynomial time bounds:

- $\mathsf{DTIME}[t(n)]$, $\mathsf{NTIME}[t(n)]$, $\mathsf{coNTIME}[t(n)]$, $\Sigma_k\mathsf{TIME}[t(n)]$ are the classes of problems that can be solved in $O(t(n))$ steps by a (deterministic) algorithm, a nondeterministic algorithm, a co-nondeterministic algorithm, and a Σ_k-algorithm, respectively.

- $\mathsf{SPACE}[s(n)]$ is the class of problems that can be solved using $O(s(n))$ auxiliary space.

- $\mathsf{DTISP}[t(n), s(n)]$ is the class of problems that can be solved by a single algorithm that takes $O(t(n))$ time and uses $O(s(n))$ space.

In this thesis, we focus on the cases where $t(n)$ and $s(n)$ are both bounded by polynomials in n. For this reason we shall not worry with issues like time and space constructibility (*i.e.* whether or not the algorithm can efficiently count how many steps and space it is going to take, in advance).

2.5 Complete Problems

One of the most astounding phenomena in computer science is the abundance and pervasiveness of *complete* problems. Intuitively speaking, a problem $\mathcal{P}$ is complete for a class of problems when $\mathcal{P}$ is in the class, and every other problem in the class can be efficiently expressed as a special case of $\mathcal{P}$. Therefore, having an efficient algorithm for $\mathcal{P}$ is tantamount to having an efficient algorithm for *every* problem in the class. *A priori*, a complete problem for NP might need to appear in a very general form, and one would not always run into such expressive and powerful problems in everyday practice. On the contrary it is precisely the opposite that is true. Complete problems for NP and other classes spring up all over computer science. These problems constitute serious challenges to everyone involved in computing, and it is their ubiquity that makes the P versus NP question so important: either all of these problems can be solved in polynomial time, or none of them.[2]

Definition 2.5.1 *A problem $\mathcal{P}$ is* NP-complete *if*

- $\mathcal{P} \in \mathsf{NP}$*, and*
- *for every $\mathcal{P}' \in \mathsf{NP}$, there is a polynomial time algorithm A that given instances of $\mathcal{P}'$ outputs an instance of $\mathcal{P}$, with the property:*

$$(\forall \textit{ instances } x)[x \in \mathcal{P}' \iff A(x) \in \mathcal{P}].$$

A problem satisfying this second condition is called NP-hard.

That is, $\mathcal{P}$ is complete if it lies in NP, and for every other $\mathcal{P}'$ in NP there is an efficient way to express instances of $\mathcal{P}'$ as instances of $\mathcal{P}$. Below is a short list of some extensively studied NP-complete problems from logic and graph theory.

- BOUNDED HALTING PROBLEM: given $(\langle A \rangle, x, 1^k)$, where $\langle A \rangle$ is the code for an algorithm A and k is an integer, is there a $y \in \{0,1\}^k$ such that $A(xy) = 1$ in at most k steps?
  This is a "canonical" complete problem, in that one can easily reduce any instance of a problem $\mathcal{P}'$ in NP to a BOUNDED HALTING PROBLEM instance, by taking the algorithm A for $\mathcal{P}'$ and setting $k = |x|^c$ for a suitably large constant c.
- SAT: given a Boolean formula F in conjunctive normal form, is there an assignment to the variables of F that makes the formula true? In a seminal paper laying the groundwork for NP-completeness, Cook [Coo71] showed that SAT is NP-complete.

[2] Along these lines, A. K. Dewdney eloquently states a good reason why $\mathsf{P} \neq \mathsf{NP}$ seems so likely: "for all the hundreds of NP-complete problems, the thousands of person-hours spent on their solution are, in a sense, cumulative, as if all this time had been spent trying to discover a polynomial time algorithm for the satisfiability problem alone." [Dew81]

- k-SAT, for any constant $k > 2$: given a Boolean formula F in k-CNF, does the question of the SAT problem hold for it? While k-SAT looks like a restriction of SAT, Karp [Kar72] showed that k-SAT is NP-complete as well, along with the remaining problems on this list. Interestingly, it is known that the 2-SAT problem is in P, and consequently is not known to be NP-complete.

- VERTEX COVER: given a graph $G = (V, E)$ and integer K, is there a K-set $S \subseteq V$ whereby for every $\{u, v\} \in E$, at least one of u, v is in S? Intuitively, the subset S *covers* all edges in G.

- INDEPENDENT SET: given a graph $G = (V, E)$ and integer K, is there a K-set $S \subseteq V$ whereby for every distinct pair $u, v \in S$, $\{u, v\} \notin E$? Intuitively, each vertex in S is *independent* of the others, in that there are no edges between them.

- MAX CUT: given a graph $G = (V, E)$ and integer K, is there a set $S \subseteq V$ such that the number of $\{u, v\} \in E$ with $u \in S$ and $v \notin S$ is at least K? In other words, we are asking if there is a cut of the vertices into two parts so that the number of edges *crossing* the cut S (from one part to the other) is at least K. The number of edges crossing a cut S is also called the *cut value of* S.

- MAX 2-SAT: given a Boolean formula F in 2-CNF and integer K, is there an assignment to its variables that makes at least K clauses of F true? Notice that the formula F can be a no-instance of 2-SAT, but (F, K) can be a yes-instance of MAX 2-SAT for small enough K. Garey, Johnson, and Stockmeyer [GJS76] proved that MAX 2-SAT is NP-complete.

For a much larger list of NP-complete problems, the reader is invited to consult Garey and Johnson's classic text [GJ79]. The first part of this thesis presents time lower bounds on all the above problems (and many other NP complete problems). The second part of the thesis presents novel algorithms for MAX CUT and MAX 2-SAT, among others.

We remark in passing that for every NP-complete problem, there is a "complementary" co-NP-complete problem. The complementary problem for SAT is TAUTOLOGY: given a Boolean formula in DNF, does it evaluate to true on every possible assignment?

2.5.1 The Robust Completeness of SAT and Other NP Problems

Our ability to prove limitations on solving specific NP-complete problems originates with a strengthening of Cook's Theorem. Define

$$\mathsf{NQL} := \bigcup_{c \geq 0} \mathsf{NTIME}[n \cdot (\log n)^c] = \mathsf{NTIME}[n \cdot \mathrm{poly}(\log n)].$$

The letters in NQL stand for *Nondeterministic Quasi-Linear time.*

Definition 2.5.2 *A problem $\mathcal{P}$ is* robustly complete for NQL *if, for every problem $L \in \mathsf{NQL}$, there is a random access machine M_L and constant k such that:*

- *For all inputs x of length n and integers $i = 1, \ldots, kn\log^2 n$, $M_L(x,i)$ outputs a bit, and runs in $O(poly(\log n))$ time and $O(\log n)$ space, simultaneously.*

- *$x \in L$ iff $M_L(x,1) \cdot M_L(x,2) \cdot M_L(x,3) \cdots M_L(x, k|x|\log^2|x|) \in \mathcal{P}$, where '$\cdot$' is concatenation.*

We also refer to the machine M_L as a robust reduction.

That is, for any problem in NQL, there is a quasilinear time reduction to problem $\mathcal{P}$ with the property that each bit of the reduction can be computed in polylogarithmic time and logarithmic space. Building on work of Gurevich-Shelah [GS89] and Schnorr [Sch78], Fortnow *et al.* proved that SAT is robustly complete.

Theorem 2.5.1 (Fortnow-Lipton-Van Melkebeek-Viglas [FLvMV05]) SAT *for formulas in conjunctive normal form is robustly complete for* NQL.

The robustness of SAT has the following significant corollary.

Corollary 2.5.1 *Let $t(n)$ be a polynomial. If $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[t(n), n^{o(1)}]$, then there is a $c > 0$ such that* SAT $\notin \mathsf{DTISP}[t(n) \cdot (\log t(n))^c, n^{o(1)}]$.

That is to say, if one can show $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[t(n), n^{o(1)}]$, then one can name an *explicit, natural problem* that is not in the class, modulo polylogarithmic factors. All lower bound proofs in this line of work establish that $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[t(n), n^{o(1)}]$ for large $t(n) = n^c$, concluding that SAT is not solvable in $n^{c-o(1)}$ time and $n^{o(1)}$ space.

Since our proofs will show results of the form $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[t(n), n^{o(1)}]$, our arguments actually show that not only does SAT have such a time-space limitation, but *any* problem that is robustly complete for NQL also is limited in an identical way. In work showing time lower bounds for off-line one-tape machines, Van Melkebeek and Raz [vMR05] have stated that almost all known NP-complete problems are robustly complete for NQL. Indeed, in a series of papers, Dewdney [Dew81, Dew82] showed that a variety of NP-complete problems are also complete for NQL under linear time reductions, and it appears very likely that all of his reductions can also be turned into robust ones. For completeness of our presentation, we sketch proofs of robust completeness for 3-SAT, VERTEX COVER, INDEPENDENT SET, MAX CUT, and MAX 2-SAT, drawing from known proofs in the literature. It turns out that natural reductions from SAT to these problems construct "gadgets", which are small sub-instances of the problem, in such a way that there is an explicit and simple correspondence between the clauses and variables of the original formula and the gadgets in the reduced instance.

Theorem 2.5.2 3-SAT *on formulas with n variables and $n \cdot \mathrm{poly}(\log n)$ clauses is robustly complete for* NQL.

Proof. Due to Karp [Kar72]. The reduction in Theorem 2.5.1 can be made to output a pair (L, F), where F is a CNF formula on $m \leq n \cdot \text{poly}(\log n)$ clauses, and L is a list of m integers, where $L[i]$ is the number of literals in the ith clause of F. Now consider the standard reduction from SAT to 3-SAT that takes the ith clause of literals $c_i = (\ell_1 \vee \ldots \vee \ell_{L[i]})$ and produces the clause group

$$(\ell_1 \vee \ell_2 \vee y_c^1), (\neg y_c^1 \vee \ell_3 \vee y_c^2), \ldots, (\neg y_c^{L[i]-4} \vee \ell_{L[i]-2} \vee y_c^{L[i]-3}), (\neg y_c^{L[i]-3} \vee \ell_{L[i]-1} \vee \ell_{L[i]}),$$

where y_c^1, y_c^2, $\ldots$, $y_c^{L[i]}$ are new variables. For example, when $k = 5$ the resulting clause group is $(\ell_1 \vee \ell_2 \vee y_c^1)$, $(\neg y_c^1 \vee \ell_3 \vee y_c^2)$, $(\neg y_c^2 \vee \ell_4 \vee \ell_5)$.

In order to efficiently produce the jth clause in the ith group of clauses, the reduction only needs to read at most two literals from the ith clause of the formula F, and $O(\log n)$ bits from the list L. With a reasonable encoding (*e.g.* encoding each literal with $\Theta(\log n)$ bits), producing the jth clause from the ith group can be done in $\text{poly}(\log n)$ time. □

Theorem 2.5.3 VERTEX COVER *is robustly complete for* NQL.

Proof. Due to Garey and Johnson [GJ79]. We give a reduction from 3-SAT formulas F to VERTEX COVER instances G_F, with the property that for all appropriate i and j, the ith edge and jth node of G_F can be computed in polylogarithmic time via random access to F.

Let F be a 3-CNF formula on n variables and m clauses. Without loss of generality, by Theorem 2.5.2, $m \leq n \cdot \text{poly}(\log n)$ and the formula F has exactly three literals in each clause (if a clause has fewer literals, we can always put an extra copy of the literal in the clause).

One classical reduction from 3-SAT to VERTEX COVER [GJ79] produces a graph with $3m + 2n$ nodes: three (named u_i^1, u_i^2, u_i^3) for each clause c_i, and two (named v_x and $v_{\neg x}$) for each variable x of the formula. The edges in the graph are $\{v_x, v_{\neg x}\}$ for all variables x, and $\{u_i^j, v_\ell\}$ for all i, j, ℓ such that ℓ is the jth literal in clause i.

For example, for a formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_4 \vee x_6) \wedge \cdots$ on n variables, the corresponding graph is

$$\{v_{x_1}, \neg v_{x_1}\}, \ldots, \{v_{x_n}, v_{\neg x_n}\}, \{u_1^1, v_{x_1}\}, \{u_1^2, v_{\neg x_2}\}, \{u_1^3, v_{x_3}\}, \{u_2^1, v_{\neg x_1}\}, \{u_2^2, v_{\neg x_4}\}, \{u_2^3, v_{x_6}\}, \ldots.$$

Setting $K = n + 2m$, one can show that the constructed graph has a vertex cover of size K if and only if the original formula is satisfiable. Computing each edge of the graph requires examining only one clause, and the relevant indices can be computed in $\text{poly}(\log n)$ time. □

Corollary 2.5.2 INDEPENDENT SET *is robustly complete for* NQL.

Proof. Use the same reduction as the previous theorem, but set $K = (3m+2n) - (n+2m) = n+m$. As a graph G has a vertex cover of size ℓ if and only if G has an independent set of size $n - \ell$, the theorem follows. □ F

Theorem 2.5.4 MAX CUT *is robustly complete for* NQL.

Proof. The proof (which is folklore) proceeds in two steps. We first reduce 3-SAT to the problem NAE 3-SAT (Not-All-Equal), where one is given a 3-CNF formula but the task is to provide a satisfying assignment that assigns *false* to at least one literal in every clause. (So, every clause has the property that not all of its literals are equally assigned.) The second step of the proof reduces NAE 3-SAT to MAX CUT. For simplicity we use the *weighted version* of MAX CUT, where each edge has a positive weight specified by $\text{poly}(\log n)$ bits, and the task is to find a cut of maximum weight sum.

Consider a 3-SAT formula F with n variables and $n \cdot \text{poly}(\log n)$ clauses. The reduction from 3-SAT to NAE 3-SAT takes each clause $c_i = (\ell_1 \vee \ell_2 \vee \ell_3)$ of F and adds the clauses

$$(\ell_1 \vee \ell_2 \vee v_{c_i}), (\neg v_{c_i} \vee \ell_3 \vee w)$$

to a formula F', for new variables v_{c_i} and a single universal variable w. Clearly, any clause in F' can easily be determined by reading a particular clause of F, encoded in $O(\log n)$ bits. Moreover, the new instance has an assignment with one true literal and one false literal in every clause if and only if the original formula is satisfiable. If the original F is satisfiable, then F' can be satisfied in a NAE fashion by setting w to false, and v_{c_i} to true iff ℓ_1 and ℓ_2 are both false. If F' is satisfiable in a NAE fashion, then let the value of w is false (without loss of generality– if w is true, then flip the values of all variables). The resulting formula's satisfiability implies the satisfiability of F.

To reduce from NAE 3-SAT to MAX CUT, given a formula F' with variables $x_1, \ldots, x_{n \cdot \text{poly}(\log n)}$, make nodes $v_{x_1}, v_{\neg x_1}, \ldots, v_{x_{n\cdot\text{poly}(\log n)}}, v_{\neg x_{n\cdot\text{poly}(\log n)}}$ in a new graph G. Convert each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ of F' into edges $\{v_{\ell_1}, v_{\ell_2}\}, \{v_{\ell_2}, v_{\ell_3}\}, \{v_{\ell_3}, v_{\ell_1}\}$ of G. These "clause edges" all have weight 1. We also add the "variable edges" $\{v_{x_i}, v_{\neg x_i}\}$ for all $i = 1, \ldots, n \cdot \text{poly}(\log n)$, each having weight n^2. Under a reasonable encoding (where the number of variables can be determined in $\text{poly}(\log n)$ time, etc.), each edge of G can be quickly determined by reading one clause of the original instance.

Finally, we claim that F' is a "yes" instance to NAE 3-SAT if and only if G has a cut of value $2m + n^3$, where $m = n \cdot \text{poly}(\log n)$ is the number of clauses in F'. Any maximum cut of G has all the variable edges $\{v_{x_i}, v_{\neg x_i}\}$ crossing it, as the weight of any variable edge dominates the total weight of all clause edges (for sufficiently large n). Therefore, the variable edges contribute n^3 to the weight of a maximum cut, and a maximum cut in G corresponds to a variable assignment in F'. A clause that has one literal false and one literal true corresponds to two edges crossing the cut, and the all-false or all-true assignment to a clause corresponds to no edges crossing the cut. It follows that k clauses are satisfied (under the NAE 3-SAT requirement) by the assignment corresponding to a maximum cut if and only if the weight of edges crossing that cut is $2k + n^3$. Thus a maximum cut in G corresponds to a satisfying assignment in F'. □

Theorem 2.5.5 MAX 2-SAT *is robustly complete for* NQL.

Proof. We reduce from 3-SAT, using Garey, Johnson, and Stockmeyer's reduction [GJS76]. Without loss of generality, a given 3-CNF formula F has exactly three literals in each clause. Given such a formula on m clauses, for each clause $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ of F we add the set of 2-CNF clauses

$$(\ell_1), (\ell_2), (\ell_3), (y_c), (\neg\ell_1 \vee \neg\ell_2), (\neg\ell_1 \vee \neg\ell_3), (\neg\ell_2 \vee \neg\ell_3), (\ell_1 \vee \neg y_c), (\ell_2 \vee \neg y_c), (\ell_3 \vee \neg y_c)$$

to a new formula F', where y_c is a new variable indexed by c. It is routine to verify that when the clause c is satisfied by a given assignment, then 7 clauses in the above set can be satisfied by the same assignment and the appropriate choice of y_c. But when c is falsified by a given assignment, then only 6 of the above clauses can be satisfied by any choice of y_c. Therefore, determining the maximum fraction of clauses that can be satisfied in F' tells us whether or not F is satisfiable: F is satisfiable if and only if $7m$ clauses of F' can be satisfied simultaneously. Clearly, every clause of F' is determined by reading only one clause of F, and so it can be computed in poly$(\log n)$ time as in the previous constructions. $\square$

2.5.2 Robust Completeness for Problems Outside of NP

Each level of the polynomial time hierarchy also has natural complete problems within it. For instance, in $\Sigma_2\mathsf{P}$, there are complete problems where one tries to guess a minimum-sized structure, and the test for equivalence is a coNP predicate. Consider the FORMULA MINIMIZATION problem from the previous section, where one wants to know if a given formula F is the smallest one with its functionality. When F is written in DNF, and the question is to determine if there is a DNF F' of size K that agrees with F, then FORMULA MINIMIZATION is known to be $\Sigma_2\mathsf{P}$-complete, by work of Umans [Uma01]. Here, our focus is on the collection of problems Σ_k-SAT for integers $k \geq 1$, defined as follows:

- For odd k, we are given a CNF formula F on k sets of variables $X_1, \ldots, X_k$ and are asked if the first order sentence
$$(Q_1 X_1) \cdots (Q_k X_k) F$$
is true, where $Q_1 = \exists$, $Q_k = \exists$, and each Q_i is the quantifier opposite to Q_{i-1}.

- For even k, we are given a DNF formula F on k sets of variables $X_1, \ldots, X_k$ and are asked if the first order sentence
$$(Q_1 X_1) \cdots (Q_k X_k) F$$
is true, where $Q_1 = \exists$, $Q_k = \forall$, and each Q_i is the quantifier opposite to Q_{i-1}.

We need to use different normal forms for the two cases because of the type of the kth quantifier: a sentence of the form $(\forall x)F$ is trivial to check for validity, if F is a CNF. Likewise, it is trivial to check that $(\exists x)F$ is valid, if F is a DNF. (Recall that the TAUTOLOGY problem, which is the "coNP version" of SAT, assumes that the formula is given in DNF.)

Meyer and Stockmeyer [MS72] showed that Σ_k-SAT is complete for $\Sigma_k\mathsf{P}$. Letting $k > 1$, if we define $\Sigma_k\mathsf{QL} := \Sigma_k\mathsf{TIME}[n \cdot \text{poly}(\log n)]$, one can start asking which problems are robustly complete for these new quasilinear time classes. The robust reduction for SAT (Theorem 2.5.1) goes through in a straightforward way to show:

Theorem 2.5.6 Σ_k-SAT *is robustly complete for* $\Sigma_k\mathsf{QL}$.

Dieter van Melkebeek (private communication) has reported that FORMULA MINIMIZATION is also robustly complete for $\Sigma_2\mathsf{QL}$.

Theorem 2.5.6 shows that every level of the polynomial time hierarchy has a natural complete problem, namely that of determining the validity of sentences with a certain number of quantifier blocks. More complete problems of this nature can be constructed from other NP-complete problems. Essentially these problems capture what happens when an NP complete problem is turned into a two-player game, between one player that is trying to find a witness and another player that is trying to eliminate witnesses. For example, Σ_2-SAT can be viewed as the problem of determining the optimal strategy for a one-round game between two players, where the first player tries to set variables that turn the underlying DNF formula into a tautology, and the second player tries to set variables that falsify the DNF.

Chapter 3

Introduction to Time-Space Tradeoffs for NP

Rooted in work of Kannan [Kan83, Kan84] in the early 80's, and initiated by Fortnow [For97] in 1997, an intriguing thread of lower bound research has yielded tangible progress on the famous $\mathsf{L} \ ? = \ \mathsf{NP}$ and $\mathsf{NL} \ ? = \ \mathsf{NP}$ questions. While it is well-known that $\mathsf{L} \neq \mathsf{NP}$ is equivalent to $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTISP}[n^k, \log n]$ for all $k \geq 1$, proving such a large lower bound on nondeterministic linear time (and consequently, SAT) currently appears beyond our reach. Nevertheless, it is of course still very interesting to ask what *can* be proven about the matter. In particular, we wish to find the largest k for which the above separation provably holds.

Naturally, when one starts to consider fixed time bounds, the model of computation becomes a possible issue. For example, Santhanam [San01] showed that SAT cannot be solved in $n^{2-\varepsilon}$ time and $n^{o(1)}$ space on multitape Turing machines, by reducing PALINDROMES to SAT and invoking an old time-space tradeoff of [Cob66]. To illustrate this kind of proof technique, we briefly describe how to extend Santhanam's result to multitape Turing machines with multiple heads per tape. Babai, Nisan, and Szegedy [BNS92] prove that the function $G_k : \{0,1\}^{kn} \to \{0,1\}$ defined by

$$G_k(x_{1,1}, \ldots, x_{1,n}, \ldots, x_{k,1}, \ldots, x_{k,n}) = \sum_{i=1}^{k} \left(\prod_{j=1}^{n} x_{i,j} \right) \mod 2$$

cannot be computed in $\Omega(n^{2-\varepsilon})$ time and $n^{o(1)}$ space on a k-head multitape Turing machine. One can easily reduce the problem of computing G_k on a given input $\{x_{i,j}\}$ to a SAT instance on $x_{i,j}$ with k auxillary variables. Let $\texttt{PARITY}(b_1, \ldots, b_k)$ be the 2^{k-1}-clause CNF that is true if and only if $\sum_i b_i = 1 \mod 2$. Then a SAT instance for computing $G_k(\{x_{i,j}\})$ is

$$\texttt{PARITY}(b_1, \ldots, b_k) \wedge \left(b_1 \iff \bigwedge_{i=1}^{n} x_{1,j} \right) \wedge \cdots \wedge \left(b_k \iff \bigwedge_{i=1}^{n} x_{k,j} \right)$$

$$\equiv \quad \texttt{PARITY}(b_1, \ldots, b_k) \wedge (\neg b_1 \vee x_{1,1}) \wedge \cdots \wedge (\neg b_1 \vee x_{1,n}) \wedge (b_1 \vee \neg x_{1,1} \vee \cdots \neg x_{1,n}) \cdots\cdots .$$

The reduction can be made robust (even for multitape machines) using ideas from the previous chapter.

The time-space lower bounds for multitape machines are interesting in their own right, but it is clear that they reveal more about the restricted nature of tape access than about the inherent difficulty of NP problems, since PALINDROMES and G_k are very easy in other realistic models.

Proposition 3.0.1 *On random access machines,* PALINDROMES *and G_k can be computed by algorithms running in $O(n \cdot \text{poly}(\log n))$ time and $O(\log n)$ space simultaneously.*

A deeper and more difficult question is how one might prove a polynomial time lower bound for random access machines in the space-restricted setting. Random access Turing machines are appealing because they are simple to reason about, and are time-equivalent to many other random-access models within polylogarithmic factors [PR81, GS89]. As we saw in the previous chapter, SAT and other NP-complete problems have strong completeness properties in the random access Turing machine model, so $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[n^k, \text{poly}(\log n)]$ implies that SAT is not in $\mathsf{DTISP}[n^{k-o(1)}, \text{poly}(\log n)]$. Therefore, proving a polynomial time lower bound for $\mathsf{NTIME}[n]$ in the random access model also implies a lower bound for a collection of well-studied NP-complete problems.

3.1 History of Time-Space Tradeoffs for Nondeterminism

We now give a brief history of relevant results leading up to the current time-space tradeoffs for nondeterministic time computation, focusing on those results which hold for random access models.

In 1984, Kannan [Kan84] proved the separation $\mathsf{DTISP}[n, o(n)] \subsetneq \mathsf{NTIME}[n]$. His result was stated for multitape Turing machines, but it works equally well for RAMs. Furthermore, he proved that there is a universal constant k such that for all polynomials $t(n)$, $\mathsf{DTISP}[t(n), o(t(n)^{1/k})] \subsetneq \mathsf{NTIME}[t(n)]$. Kannan's proof used traditional simulation and diagonalization ideas. All later time-space tradeoffs for SAT build upon his original idea, in a certain sense; we shall discuss that sense later in more detail.

In 1997, Fortnow [For97] (following a strategy similar to Kannan) showed that SAT is not in the intersection of NL and $\mathsf{DTIME}[n^{1+o(1)}]$. In particular,

$$\mathsf{NTIME}[n] \not\subseteq \mathsf{NL} \cap \mathsf{DTIME}[n^{1+o(1)}].$$

Applying an efficient reduction from problems in $\mathsf{NTIME}[n]$ to SAT, Fortnow proved that either SAT is not solvable in nondeterministic logspace, or SAT is not solvable in $n^{1+o(1)}$ time.

In 1999, Ajtai [Ajt02] studied the element distinctness problem: given a list of $O(\log n)$-bit strings, determine if all strings are different. He proved the following time-space tradeoff bound for any random access machine with $O(\log n)$-bit registers: for all $k > 1$ there is an $\varepsilon > 0$ such that ELEMENT DISTINCTNESS cannot be solved in time kn with εn bits of memory. (Note that the complement of ELEMENT DISTINCTNESS can be easily solved on a nondeterministic machine that guesses two distinct registers and checks that they are the same.) Ajtai's techniques were combinatorial in nature, instead of being based on diagonalization.

Also in 1999, Lipton and Viglas [LV99] sharpened Fortnow's results, considering the case where both the time and space bounds are small. They proved for all $c < \sqrt{2}$ that

$$\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[n^c, n^{o(1)}].$$

This implies that SAT cannot be solved in $n^{\sqrt{2}-\varepsilon}$ time and sub-polynomial (*i.e.* $n^{o(1)}$) space, for all $\varepsilon > 0$. Shortly after Lipton and Viglas' work appeared, Fortnow and Van Melkebeek [FvM00] improved the time lower bound to $\Omega(n^{\phi-\varepsilon})$ for all $\varepsilon > 0$, where $\phi \approx 1.618$ is the golden ratio. (The journal version [FLvMV05] of this work merges the two results.) We shall present proofs of these two lower bounds shortly.

3.2 Indirect Diagonalization

The lower bounds of Kannan, Fortnow, Lipton-Viglas, and Fortnow-Van Melkbeek were all proved using a common strategy that has been coined "indirect diagonalization." (We believe that Dieter van Melkebeek was the originator of this term, in [vM04].) The idea behind an indirect diagonalization lower bound is to assume the opposite of what one wants to prove, then use the algorithms implied by this assumption to derive a contradiction with a known diagonalization result.

Many lower bound arguments from the past have followed a similar high-level pattern that transcends the time-space tradeoff lower bounds above. We call this pattern the *alternation-trading scheme*. A special case of this pattern was identified in a survey by Van Melkebeek [vM04] and in Viglas' thesis [Vig02].

3.2.1 The Alternation-Trading Scheme for Proving Lower Bounds

Let $\mathcal{D}[t(n)]$ denote a class of problems solved by some deterministic machine model running in time $t(n)$. Many lower bound arguments for nondeterministic time (and in general, alternating time) follow a certain high-level scheme:

1. Assume (for contradiction) that $\mathsf{NTIME}[n] \subseteq \mathcal{D}[n^c]$.
2. Prove that for reasonable time functions $t(n)$, $\mathcal{D}[t(n)]$ can be *sped up* by alternating machines. More precisely, $\mathcal{D}[t(n)] \subseteq \Sigma_\ell\mathsf{TIME}[f(t(n))]$ for some $\ell \geq 1$ and $f(n) = o(n)$.
3. Prove using (1) that alternations in an alternating computation can be "removed", at the cost of a small *slow down* in time. For example, (1) implies $\Sigma_2\mathsf{TIME}[n] \subseteq \mathsf{NTIME}[n^c]$, by converting the co-nondeterministic part of the Σ_2 computation into a deterministic computation.
4. Apply (1) and (2) in some sequence on a given complexity class, in such a way that one can conclude a contradiction to a known time hierarchy theorem.

More details on the alternation-trading scheme will be elucidated in upcoming sections. We believe the first example of a lower bound following the alternation-trading scheme was given by Kannan [Kan83] in 1983, who showed that nondeterministic linear time on multitape machines is not contained in deterministic $n^{1.1}$ time on one-tape TMs. Since then, a number of lower bounds on nondeterminism and alternation in various machine models [PPST83, Kan84, MS87, WL92, For97, LV99, FvM00, Tou01] have followed this alternation-trading scheme, often in an implicit manner. For example, the celebrated result of Paul, Pippenger, Szemeredi, and Trotter [PPST83] that $\mathsf{NTIME}[n] \neq \mathsf{DTIME}[n]$ for multitape machines can be said to follow the alternation-trading scheme:

1. Assume $\mathsf{NTIME}[n] = \mathsf{DTIME}[n]$.
2. Paul-Pippenger-Szemeredi-Trotter prove $\mathsf{DTIME}[t] \subseteq \Sigma_4\mathsf{TIME}[t/\log^* t]$, for $t(n) \geq n \log^* n$.
3. Item (1) implies that $\Pi_k\mathsf{TIME}[n] = \mathsf{coNTIME}[n] = \mathsf{DTIME}[n]$ for all k.
4. Therefore by padding, $\Pi_4\mathsf{TIME}[t] = \mathsf{DTIME}[t] \subseteq \Sigma_4\mathsf{TIME}[t/\log^* t]$, a contradiction with the alternating time hierarchy.

In the next sections and the following chapter, we give a detailed account of the research on time-space lower bounds for nondeterminism on RAMs, beginning with the $\Omega(n^{\sqrt{2}})$ lower bound of Lipton-Viglas and culminating in our $\Omega(n^{1.801})$ lower bound. The chapter after that shows how to formalize the alternation-trading scheme so that further progress on alternation-trading lower bounds can be feasibly automated.

3.3 Prior Time-Space Lower Bounds and Their Tools

In this remainder of this chapter, we describe the Lipton-Viglas and Fortnow-Van Melkebeek time-space lower bounds on satisfiability, starting with the tools from complexity theory that they employ.

A Class Separation. We require some well-known separation results, each of which are provable by straightforward diagonalization. The following result uses the fact that a random-access machine using k quantifiers in time t can be simulated by a machine using k quantifiers in time $O(t)$, found in Chandra and Stockmeyer's original conference paper on alternation [CS76].

Theorem 3.3.1 ("No Complementary Speedup") *For all $k \geq 1$ and time constructible $t(n) \geq n$, $\Sigma_k\mathsf{TIME}[t] \nsubseteq \Pi_k\mathsf{TIME}[o(t)]$.*

We call it the "No Complementary Speedup" Theorem, as it intuitively says that not all bounded-alternation machines can be sped up by a "complementary" machine with the same number of alternations.

A "Slowdown" Lemma. A simple but useful proposition says that alternations can be removed from a computation while slightly increasing the runtime, provided that there is a close time relationship between classes with fewer alternations. We do not know of a reference for this lemma, but its proof is elementary.

Lemma 3.3.1 (Slowdown Lemma) *Let $d > 1$, let k and ℓ be non-negative integers with $k > \ell$, and let $t(n) \geq n$ be time constructible. If $\Sigma_\ell\mathsf{TIME}[n] \subseteq \Pi_\ell\mathsf{TIME}[n^d]$, then*

- $\Sigma_k\mathsf{TIME}[t] \subseteq \Sigma_{k-1}\mathsf{TIME}[t^d]$, *and*
- $\Pi_k\mathsf{TIME}[t] \subseteq \Pi_{k-1}\mathsf{TIME}[t^d]$.

Proof. First, observe that $\Sigma_\ell\mathsf{TIME}[n] \subseteq \Pi_\ell\mathsf{TIME}[n^d]$ implies $\Pi_\ell\mathsf{TIME}[n] \subseteq \Sigma_\ell\mathsf{TIME}[n^d]$. So by padding, any ℓ-quantifier machine M running in time t is equivalent to some ℓ-quantifier machine N that runs in time t^d, but if M begins with an $\exists$ (resp. $\forall$) quantifier, then N begins with a $\forall$ (resp. $\exists$) quantifier.

Let M be a Σ_k machine with $O(t)$ runtime. Without loss of generality, we may assume that the state of M at the start of the $(k-\ell)^{\text{th}}$ alternation is a special state q^*. Define a machine M^* whose input is an input x to M and a tape configuration C of M:

$M^*(x, C)$: Simulate $M(x)$, starting from C and state q^*.

By assumption, M^* has ℓ quantifiers, since M has k quantifiers and q^* starts at the $(k-\ell)^{th}$ alternation (the beginning of the $(k-\ell+1)^{th}$ quantifier). M^* runs in $O(t)$ time and takes inputs of $O(t)$ size.

The hypothesis implies that there is a machine N^* that is equivalent to M^*, runs in $O(t^d)$ time, uses ℓ quantifiers, but begins each computation with the quantifier opposite to that with which M^* begins. We now define a machine N:

$N(x)$: Simulate $M(x)$ until q^* is reached.
Let C be the configuration of $M(x)$ at this point. Simulate $N^*(x, C)$.

It is easy to verify that $L(N) = L(M)$, and that N runs in $O(t^d)$ time. We claim that N uses only $k-1$ quantifiers. This follows from the fact that the last quantifier of $M(x)$ prior to q^* and the first quantifier of N^* are the *same*. Therefore, in N, no alternation occurs at state q^*. But N and M still have the same number of alternations occurring prior to q^* and after q^*, so N^* has one less alternation than M. □

Fortnow and Van Melkebeek's Speedup Simulations. Along with trading alternations for more time (via the slowdown lemma), we also need to trade *time for alternations*: that is, we reduce the runtime of DTISP computations using alternating machines. Fortnow and Van Melkebeek's speedup simulations are crucial to some of our arguments.

Lemma 3.3.2 (Speedup Lemma of Fortnow-Van Melkebeek [FvM00], Theorem 5.1) *For every natural number $k \geq 2$, and time constructible t, space constructible s, and $b(n)$ such that $1 \leq b(n) \leq t(n)$,*

$$\mathsf{DTISP}[t, s] \subseteq \Sigma_k\mathsf{TIME}[k \cdot b \cdot s + t/b^{k-1}].$$

In particular, the first $(k-1)$ quantifier stages run in $O(b \cdot s)$ time each, and the last quantifier stage guesses $O(\log b)$ bits, followed by a deterministic stage that takes input of length $n+2s$ and runs in $O(t/b^{k-1})$ time and $O(s)$ space.

The case $k=2$ was essentially proved by Kannan [Kan84]. The key idea of Lemma 3.3.2 is to mimic the proof in Chandra, Kozen, and Stockmeyer [CKS81] (following Savitch's theorem [Sav70]) that $\mathsf{DTISP}[t,s] \subseteq \mathsf{ATIME}[s \log t]$. In the proof of $\mathsf{DTISP}[t,s] \subseteq \mathsf{ATIME}[s \log t]$, the alternating simulation of a $\mathsf{DTISP}[t,s]$ machine M works by repeatedly guessing configurations "in the middle" of the computation. Let us think of the input to an alternating simulation A as a triple $\langle k, C, C' \rangle$, where k is a positive integer and C and C' are configurations of M. A wishes to output *yes* iff, when M is executed from C for 2^k steps, its configuration becomes C'. (Without loss of generality, the runtime t is a power of two.) To do this, if $k=0$ then A just simulates M from C for one step, and checks if its configuration equals C'. For $k>0$, A existentially guesses a configuration C'', which is supposed to be the configuration of M that occurs $2^{k-1} = 2^k/2$ steps after starting from C; let us call this configuration C''. A universal quantifier then guesses a 0 or a 1. Finally, A calls itself on $\langle C, C'', k-1 \rangle$ if 0 was written, and calls itself on $\langle C'', C', k-1 \rangle$ if 1 was written. It is easy to see that this simulation works; a little analysis shows that its runtime is $O(s \log t) = O(s^2)$.

The above machine A uses many alternations during its execution ($O(\log t)$, as a matter of fact). To obtain a fast simulation of M that uses a constant number of alternations, one can existentially guess many configurations at once, and universally check each one we guessed. This sacrifices the $O(s \log t)$ runtime, but uses vastly fewer alternations.

Proof of Lemma 3.3.2. (Sketch) Fix a deterministic machine M using time $t(n)$ and space $s(n)$. We first show how to simulate M in $kbs + t/b^{k-1}$ time with $2(k-1)$ quantifiers (a $\Sigma_{2(k-1)}\mathsf{TIME}[kbs + t/b^{k-1}]$ machine). Next, we show how the construction can be modified to use only k quantifiers, *i.e.* we make a simulation in $\Sigma_k\mathsf{TIME}[kbs + t/b^{k-1}]$.

We describe a machine N using $2(k-1)$ quantifiers that simulates M below.

$N(x)$: Let C_0 and C_{t+1} be the unique initial and accept configurations of $M(x)$.
Return *Simulate*$(x, C_0, C_{t+1}, k-1)$.

Simulate(x, C_i, C_j, j):

If $j = 0$ then *accept* iff C_i leads to C_j in at most t/b^{k-1} steps on input x.

- *Existentially* guess machine configurations $C_1^j, \ldots, C_b^j$ of $M(x)$.
  If $C_1^j \neq C_i$ then *reject*.
  If $C_b^j \neq C_j$ then *reject*.
- *Universally* choose $i_j \in \{1, \ldots, b-1\}$ and return *Simulate*$(x, C_{i_j}^j, C_{i_j+1}^j, j-1)$.

It is straightforward to verify that the procedure *Simulate* works analogously to the proof of $\mathsf{DTISP}[t,s] \subseteq \mathsf{ATIME}[s \cdot \log t]$, except that instead of guessing just the "midpoint" configuration C'', we are guessing $b-1$ "midpoint" configurations of $M(x)$ before each recursive call. This increases the runtime, but lowers the number of required alternations.

The machine N clearly has $2(k-1)$ quantifiers, guessing $O(b \cdot s)$ bits existentially and $O(\log t)$ bits universally between each recursive call, then (when $k=0$) running for $O(t/b^{k-1})$ deterministic time and $O(s)$ space on a RAM. Thus the procedure takes $O(k \cdot b \cdot s + t/b^{k-1})$ time overall. Notice that the input to the deterministic part is x and two configurations, so it is of length $n+2s$.

How can we reduce the number of quantifiers to k? We exploit the fact that the computation is deterministic, and therefore closed under complement. Rewrite *Simulate* to be a "negation" of the above:

Simulate2(x, C_i, C_j, j):

If $j=0$ then *accept* iff C_i leads to C_j on input x in at most t/b^{k-1} steps.

- *Universally* choose configurations $C_1^j, \ldots, C_b^j$ of $M(x)$.
  If $C_1^j \neq C_i$ then *accept*.
  If $C_b^j = C_j$ then *accept*.
- *Existentially* choose $i_j \in \{1, \ldots, b-1\}$.
  Return $\neg$*Simulate2*$(x, C_{i_j}^j, C_{i_j+1}^j, j-1)$.

Intuitively, *Simulate2* verifies that C_i leads to C_j by considering all sequences of configurations where the first configuration is C_i, but the last one is *not* C_j. *Simulate2* verifies that for any such sequence, there are two adjacent configurations that do *not* lead from one to the other. Since all sequences from C_i to some $C_j' \neq C_j$ fail to work, it must be that C_i leads to C_j. Clearly, *Simulate2* runs in the same time bound and number of alternations as *Simulate*, but the quantifiers start with a $\forall$ instead.

The final algorithm makes the two procedures mutually recursive: rewrite *Simulate* so that it calls *Simulate2*, and vice-versa. Then, the number of quantifiers in $N(x)$ becomes exactly k, where the first $(k-1)$ quantifiers guess $O(b \cdot s)$ bits each, the last quantifier guesses $O(\log b)$ bits, and the final deterministic phase runs in $O(t/b^{k-1})$ time and $O(s)$ space. □

Remark 3.3.1 *Note that* $\mathsf{DTISP}[t, s] \subseteq \Pi_k\mathsf{TIME}[k \cdot b \cdot s + t/b^{k-1}]$ *follows immediately from the closure of* DTISP *under complementation.*

A significant instantiation of Lemma 3.3.2 is the following important corollary.

Corollary 3.3.1 *For all integers* $k \geq 2$, $\mathsf{DTISP}[t, s] \subseteq \Sigma_k\mathsf{TIME}[(ts^{k-1})^{1/k}]$. *In particular,*

$$\mathsf{DTISP}[t, t^{o(1)}] \subseteq \Sigma_k\mathsf{TIME}[t^{1/k+o(1)}].$$

Proof. When $b = (t/s)^{1/k}$, the overall runtime of the Lemma 3.3.2 simulation is minimized, resulting in the corollary. □

3.3.1 SAT is not in $\mathsf{DTISP}[n^{\sqrt{2}-\varepsilon}, n^{o(1)}]$

Our study of lower bound arguments begins with a short description of Lipton and Viglas' argument, who showed that $\mathsf{NTIME}[n]$ is not contained in $\mathsf{DTISP}[n^{\sqrt{2}-\varepsilon}, n^{o(1)}]$, for all $\varepsilon > 0$.

Theorem 3.3.2 $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTISP}[n^{\sqrt{2}-\varepsilon}, n^{o(1)}]$ *for all* $\varepsilon > 0$.

Proof. If $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^{\sqrt{2}-\varepsilon}, n^{o(1)}]$, then by padding and Corollary 3.3.1,

$$\mathsf{NTIME}[n^2] \subseteq \mathsf{DTISP}[n^{2(\sqrt{2}-\varepsilon)}, n^{o(1)}] \subseteq \Sigma_2\mathsf{TIME}[n^{\sqrt{2}-\varepsilon+o(1)}].$$

But by the Slowdown Lemma, we have that

$$\mathsf{NTIME}[n^2] \subseteq \Sigma_2\mathsf{TIME}[n^{\sqrt{2}-\varepsilon+o(1)}] \subseteq \mathsf{NTIME}[n^{(\sqrt{2}-\varepsilon)(\sqrt{2}-\varepsilon)+o(1)}],$$

which contradicts Theorem 3.3.1 ("No Complementary Speedup") when $\varepsilon > 0$. $\square$

In later sections, we show how to recast the above argument in a simple way that lets us prove better lower bounds by invoking the argument for multiple times.

3.3.2 SAT is not in $\mathsf{DTISP}[n^{\phi-\varepsilon}, n^{o(1)}]$

Not long after the $n^{\sqrt{2}}$ lower bound was proved, Fortnow and Van Melkebeek found an improvement of the lower bound to n^{ϕ}, where ϕ is the golden ratio. (For a detailed exposition of the two lower bounds, please see the combined journal version of their papers [FLvMV05].) Here, we give an alternative exposition of their lower bound. To simplify the presentation, we introduce some new notation:

- Define $\mathsf{DTS}[t(n)]$ to be the class of problems solved by random access machines that run in $t(n)^{1+o(1)}$ time and use $t(n)^{o(1)}$ workspace. In the following, $t(n)$ is always a polynomial in n, so $t^{o(1)}$ is always $n^{o(1)}$.
- Let $\mathcal{C}$ be a complexity class and f be a constructible function. Define $(\exists\ f(n))\ \mathcal{C}$ to be the class of problems solved by some nondeterministic machine N that, on input x, writes a $f(n)^{1+o(1)}$ bit string y nondeterministically to a special tape, then feeds the input $\langle x, y\rangle$ to a machine from class $\mathcal{C}$. The class $(\forall\ f(n))\ \mathcal{C}$ is defined similarly (with co-nondeterministic machines).

Notice that the above notation avoids the inclusion of $o(1)$ factors in exponents. As an example of the notation in use, observe that Lemma 3.3.2 implies

$$\mathsf{DTS}[n^d] \subseteq (\exists\ n^x)(\forall \log n)\mathsf{DTS}[n^{d-x}], \quad \mathsf{DTS}[n^d] \subseteq (\forall\ n^x)(\exists\ \log n)\mathsf{DTS}[n^{d-x}],$$

for any $d \geq x \geq 0$.

Lemma 3.3.3 *For all $k \geq 0$, if $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, then*
$\mathsf{DTS}[n^{2+\sum_{i=1}^{k} c^i}] \subseteq \Pi_2\mathsf{TIME}[n^{c^k+o(1)}] \cap \Sigma_2\mathsf{TIME}[n^{c^k+o(1)}]$.

Proof. The base case is obvious, as $\mathsf{DTS}[n^2] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}] \cap \Sigma_2\mathsf{TIME}[n^{c^k+o(1)}]$. The induction hypothesis is that $\mathsf{DTS}[n^{2+\sum_{i=1}^{k} c^i}] \subseteq \Pi_2\mathsf{TIME}[n^{c^k+o(1)}]$. By the Speedup Lemma (Lemma 3.3.2),

$$\mathsf{DTS}[n^{2+\sum_{i=1}^{k+1} c^i}] \subseteq (\forall\ n^{c^{k+1}})(\exists\ \log n)\mathsf{DTS}[n^{2+\sum_{i=1}^{k} c^i}].$$

The Speedup Lemma also states that the input to the $\mathsf{DTS}[n^{2+\sum_{i=1}^{k} c^i}]$ part of the above class is a pair of $n^{o(1)}$ space configurations, along with an input x of length n. Applying the induction hypothesis,

$$(\forall\ n^{c^{k+1}})(\exists\ \log n)\mathsf{DTS}[n^{2+\sum_{i=1}^{k} c^i}] \quad \subseteq \quad (\forall\ n^{c^{k+1}})(\exists\ \log n)\Sigma_2\mathsf{TIME}[n^{c^k+o(1)}].$$

Since $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, it follows that $\Sigma_2\mathsf{TIME}[n] \subseteq \mathsf{NTIME}[n^c]$ by the Slowdown Lemma, and so

$$(\forall\ n^{c^{k+1}})(\exists\ \log n)\Sigma_2\mathsf{TIME}[n^{c^k+o(1)}] \subseteq (\forall\ n^{c^{k+1}})(\exists\ \log n)\mathsf{NTIME}[n^{c^{k+1}+o(1)}],$$

which is contained in $\Pi_2\mathsf{TIME}[n^{c^{k+1}+o(1)}]$. By swapping the roles of $\exists$ and $\forall$ in the above, similar reasoning shows that $\mathsf{DTS}[n^{2+\sum_{i=1}^{k+1} c^i}] \subseteq \Sigma_2\mathsf{TIME}[n^{c^{k+1}+o(1)}]$. $\square$

Theorem 3.3.3 *For all $\varepsilon > 0$, $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^{\phi-\varepsilon}]$, where $\phi = 1.618\ldots$ is the golden ratio.*

Proof. Assume $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ for some $c < \phi$. Consider the class $\Sigma_2\mathsf{TIME}[n^{2+\sum_{i=1}^{k} c^i}]$, for a parameter $k > 1$ to be determined later. By assumption and the Slowdown Lemma,

$$\Sigma_2\mathsf{TIME}[n^{2+\sum_{i=1}^{k} c^i}] \subseteq \mathsf{NTIME}[n^{2c+\sum_{i=1}^{k} c^{i+1}}] \subseteq \mathsf{DTS}[n^{2c^2+\sum_{i=1}^{k} c^{i+2}}].$$

By Lemma 3.3.3 and padding,

$$\mathsf{DTS}[n^{2c^2+\sum_{i=1}^{k} c^{i+2}}] \subseteq \Pi_2\mathsf{TIME}[n^{c^{k+2}+o(1)}].$$

A contradiction to the "No Complementary Speedup" Theorem (Theorem 3.3.1) follows, provided that

$$\begin{aligned} 2 + \sum_{i=1}^{k} c^i > c^{k+2} \iff 2/c^k + \sum_{i=1}^{k} c^{i-k} > c^2 \quad &\iff \quad 2/c^k + \sum_{j=0}^{k-1} \frac{1}{c^j} > c^2 \\ &\iff \quad 2/c^k + \frac{1-\frac{1}{c^k}}{1-\frac{1}{c}} > c^2. \end{aligned}$$

For $k \to \infty$, observe that the above inequality becomes $\frac{1}{1-1/c} > c^2$, *i.e.*,

$$1 > c(c-1).$$

However, $c(c-1) < 1$ holds for any $c < \phi$. In particular, for any $c < \phi$, one can choose a sufficiently large k satisfying $2 + \sum_{i=1}^{k} c^i > c^{k+2}$. $\square$

When one first sees the above proof (in particular, Lemma 3.3.3), it is difficult to find any obvious weakness or sub-optimality in its steps. For this reason, the $\Omega(n^{\phi-\varepsilon})$ time lower bound stood as the best one known for five years. However there are weaknesses– in a sentence, the significant weakness of the golden ratio lower bound is that not all of its steps actually use the presumed SAT algorithm ($\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$) to the greatest capacity, as we shall see in the sequel.

3.4 Chapter Summary

We introduced the *alternation-trading scheme* for proving lower bounds, and presented some algorithmic tools for manipulating small-space computations. Following the alternation-trading scheme, we demonstrated two time lower bounds for solving many natural NP-complete problems on space-bounded random access machines: the $\Omega(n^{\sqrt{2}})$ lower bound of Lipton and Viglas, and the $\Omega(n^{\phi})$ lower bound of Fortnow and Van Melkebeek.

Chapter 4

New Time-Space Tradeoffs for NP Problems

In the previous chapter, we discussed an *alternation-trading* scheme for proving lower bounds by indirect diagonalization, along with some tools used to carry out the scheme. Let $\mathcal{D}[t]$ be some deterministic complexity class parameterized by a resource bound t. The alternation-trading scheme proceeds as follows:

1. Assume $\mathsf{NTIME}[n] \subseteq \mathcal{D}[n^c]$.
2. Prove that for reasonable time bounds $t(n)$, $\mathcal{D}[t(n)]$ can be *sped up* by alternating machines.
3. Prove using (1) that alternations in an alternating computation can be "removed", at the cost of a small *slow down* in time.
4. Apply (1) and (2) in some sequence on a given complexity class, so that one can conclude a contradiction to a known time hierarchy theorem.

In this chapter, we describe four time-space lower bounds for SAT of our own design which also follow the alternation-trading scheme. To give the reader a feeling of how our results came about, our mode of presentation will be that of a chronologically ordered narrative, where each successive lower bound introduces new observations that allow us to improve upon the weaknesses of the previous bound. For those readers who are interested only in the final $\Omega(n^{1.801})$ lower bound and are familiar with the previous chapter, it suffices to read Sections 4.2 and 4.4.

First, we develop an inductive argument that improves the time-space lower bound for solving SAT (and other NP-complete problems) to $\Omega(n^{1.661})$ time for RAMs that use $n^{o(1)}$ space. Our induction derives

$$\Sigma_k\mathsf{TIME}[n] \subseteq \Pi_k\mathsf{TIME}[n^{b_k}]$$

for a decreasing sequence $\{b_k\}$. In the case where a contradiction does not hold for us, we still obtain a relation between Σ_k and Π_k, which is useful for deriving a relation between Σ_{k+1} and Π_{k+1} and higher levels of the polynomial hierarchy. The dichotomy of deriving either (a) a contradiction, or (b) a better inclusion than before, is the leverage that allows us to improve the lower bounds. Thus one may say that this kind of argument helps us *improve upon item (3)* in the alternation-trading scheme.

Theorem 4.0.1 ([Wil05b]) SAT *cannot be solved by random access machines using* $O(n^{1.661})$ *time and* $n^{o(1)}$ *space.*

Secondly, we boost the time lower bound to $n^{1.7327}$, which is a bit larger than $n^{\sqrt{3}}$. To do this, we *improve upon item (2)* in the 4-step scheme. That is, we give an improved speedup of DTS in $\Sigma_2\mathsf{TIME}$. (Recall from the previous chapter that $\mathsf{DTS}[t(n)] := \mathsf{DTISP}[t(n), t(n)^{o(1)}]$.) The key behind our speedup is that it relies heavily on item (1), *i.e.* the assumption that $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$. The prior speedup results of this kind (Lipton-Viglas and Fortnow-Van Melkebeek) did not utilize this assumption— in fact, their containments of DTS in $\Sigma_k\mathsf{TIME}$ hold unconditionally.

Theorem 4.0.2 ([Wil05b]) SAT *cannot be solved by random access machines using* $O(n^{1.7327})$ *time and* $n^{o(1)}$ *space.*

Thirdly, we show how to extend the inductive argument in the $n^{1.7327}$ lower bound to obtain a better speedup of DTS in $\Sigma_k\mathsf{TIME}$, whereby the improvement increases as k increases. This results in a bound of $\Omega(n^{1.78})$. Finally, by combining the ideas of our work with the $\Omega(n^{1.618})$ lower bound argument of Fortnow-Van Melkebeek, we obtain an $\Omega(n^{1.801})$ time lower bound for SAT. Our study of these lower bound arguments culminates in the next chapter, where we present a formal proof system that captures all these lower bound arguments, and design a computer program that can systematically search for new proofs in a feasible way.

At this point, the casual reader may rightly wonder: *why should we care*? What difference does it really make if SAT can't be solved in $n^{1.6}$ or $n^{1.8}$ – the real question is whether or not it can be solved in polynomial time, with *no* space restriction! While this is the question we would all like to tackle, we have little idea of how to even get started with it. Our ultimate goal is certainly not to fiddle with 0.2 factors in an exponent– it is to develop a deeper understanding of indirect diagonalization lower bounds, both their capabilities and their limitations. There *could be* a proof of $\mathsf{L} \neq \mathsf{NP}$ that uses only classical techniques already known to us, and we do not yet know how to rule out this possibility. (However, in the following chapter we do make progress on a scaled-down version of this question.) Complaining that we should be studying the general P vs NP problem instead of understanding the limits of existing techniques is putting the cart before the horse. In the following chapter, our diligence brings us much more than just a minor improvement on a known limitation– we are led to a formalization of the alternation-trading program for proving lower bounds, not just for time-space tradeoffs but for other proofs following the alternation-trading scheme.

4.1 Sat is not in $\mathsf{DTISP}[n^{1.661}, n^{o(1)}]$

Our first lower bound relies on a new inductive strategy that elaborates upon the alternation-trading scheme, taking better advantage of the polynomial hierarchy in obtaining a contradiction. It is very general and can be applied to improve upon several other time lower bounds as well [Wil05b] for various restricted computational models. The main idea is to derive a sequence of inclusions between complexity classes, where the derivation of each new inclusion uses all of the previous inclusions in the sequence in a productive way.

Notice that, if nondeterministic time n can be simulated in deterministic time n^c, then by Lemma 3.3.1 it follows that $\Sigma_k\mathsf{TIME}[t] \subseteq \Sigma_{k-1}\mathsf{TIME}[t^c]$. A key observation behind our results is that, if we further assume nondeterministic time n is in deterministic time n^c and space $n^{o(1)}$ for $c < 2$, this not only implies that $\Sigma_k\mathsf{TIME}[t]$ can be efficiently simulated by a Σ_{k-1} machine, but also that the runtime for this simulation is *faster* than t^c. Moreover, as k increases, the implied simulation gets faster for appropriately small c. This simulation can be used to improve item (3) from the alternation-trading scheme (the "alternation removal"). We start by using items (1) and (2) from the alternation-trading scheme to derive

$$\Sigma_2\mathsf{TIME}[n] \subseteq \Pi_2\mathsf{TIME}[n^{f_2}]$$

for a small constant f_2. Essentially this means that an "OR of ANDs" at the bottom of the configuration tree of an alternating machine can be switched with an "AND of ORs" of polynomial size. If $f_2 < 1$, then the above already contradicts a known time hierarchy theorem. This idea already gives an alternative proof that $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTISP}[n^c, n^{o(1)}]$ for $c < \sqrt{2}$. First, we assume the contrary. Then by the machinery presented in the previous chapter,

$$\begin{aligned} \Sigma_2\mathsf{TIME}[n] \subseteq \mathsf{NTIME}[n^c] &\subseteq \mathsf{DTS}[n^{c^2}] && (4.1)\\ &\subseteq \Pi_2\mathsf{TIME}[n^{\frac{c^2}{2}+o(1)}] \subseteq \Pi_2\mathsf{TIME}[o(n)], && (4.2)\end{aligned}$$

a contradiction. (The first inclusion follows from the Slowdown Lemma, the second follows by assumption, and the third by Fortnow and Van Melkebeek's simulation.)

Our major observation is that, even if $f_2 = c^2/2 \geq 1$, we have not necessarily lost the battle. Under the right conditions, this inclusion of Σ_2 linear time in Π_2 can be invoked to prove an even better relationship between Σ_3 and Π_3, namely

$$\Sigma_3\mathsf{TIME}[n] \subseteq \Pi_3\mathsf{TIME}[n^{f_3}]$$

for some $f_3 < f_2$. To illustrate, consider if we let $c \geq \sqrt{2}$ in the above inclusion of Σ_2 in Π_2. Then the resulting derivation

$$\Sigma_2\mathsf{TIME}[n] \subseteq \Pi_2\mathsf{TIME}[n^{\frac{c^2}{2}+o(1)}]$$

is not quite a contradiction, but it is at the very least a *lemma* that, in conjunction with Lemma 3.3.1, implies $\Sigma_k\mathsf{TIME}[n] \subseteq \Sigma_{k\text{-}1}\mathsf{TIME}[n^{\frac{c^2}{2}+o(1)}]$, for all $k \geq 3$. Provided that $c < 2$, this is stronger than $\Sigma_k\mathsf{TIME}[n] \subseteq \Sigma_{k\text{-}1}\mathsf{TIME}[n^c]$. The lemma can then be used to get an even tighter inclusion for Σ_3 in Π_3, in particular

$$\Sigma_3\mathsf{TIME}[n] \subseteq \Sigma_2\mathsf{TIME}[n^{\frac{c^2}{2}+o(1)}] \subseteq \mathsf{DTS}[n^{\frac{c^4}{2}}] \subseteq \Pi_3\mathsf{TIME}[n^{\frac{c^4}{6}+o(1)}].$$

If $c^4 < 6$, we have a contradiction. Otherwise, the above inclusion between Σ_3 and Π_3 can be used to prove a relation between Σ_4 and Π_4.

In general, an inclusion of $\Sigma_k\mathsf{TIME}[n]$ in $\Pi_k\mathsf{TIME}[n^{f_k}]$ for some f_k's can be proved by using relations between all lower levels of the polynomial hierarchy. On the one hand, as long as each derived exponent $f_k \geq 1$, then we may apply it to remove alternations from computations with more quantifiers. On the other hand, if any derived exponent f_k ever drops below 1, then we know that the hypothesis assumed in step (1) of the alternation-trading scheme must be false, as that contradicts a time hierarchy theorem. Suppose we derive $\Sigma_2\mathsf{TIME}[n] \subseteq \Pi_2\mathsf{TIME}[n^{f_2(c)}]$, $\Sigma_3\mathsf{TIME}[n] \subseteq \Pi_3\mathsf{TIME}[n^{f_3(c)}]$, *etc.* Our construction and choice of c ensure that the sequence $f_2(c)$, $f_3(c)$, $f_4(c)$, $\ldots$ eventually drops below 1. Moreover, the value of c such that the sequence drops below 1 is larger than the lower bound exponents previously obtained. An inductive strategy for improving lower bounds naturally arises: derive increasingly better Π_k simulations of Σ_k using the previous simulations obtained, and take c to be the largest constant that implies $\Pi_k\mathsf{TIME}[n] \subseteq \Sigma_k\mathsf{TIME}[o(n)]$ for some k. In many cases, this particular attack yields better lower bounds than previous approaches, cf. [Wil05b].

4.1.1 Formalization

We now present a more formal exposition of the inductive strategy. The main theorem of this section is the following.

Theorem 4.1.1 *For every integer $k \geq 2$ and real $c \geq 1$ such that $c < f(k)$,*

$$\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^c],$$

where $f(k) := \prod_{j=1}^{k-1}(1 + 1/j)^{1/2^j}$.

Let us first observe some properties of the f function.

Lemma 4.1.1 *$f(k)$ is monotone increasing and converges to a value greater than 1.661.*

Proof of Lemma 4.1.1. As $(1 + 1/j)^{1/2^j} > 1$ for all j, it is evident that $f(k)$ is monotone increasing. Observe that $(1+1/j)^{1/2^j} \leq \exp(\frac{1}{j \cdot 2^j})$, so $f(k) \leq \exp(\sum_{j=1}^{k-1} \frac{1}{j \cdot 2^j})$. As the sum $\sum_{j=1}^{k-1} \frac{1}{j \cdot 2^j}$ converges, $f(k)$ converges also. Computation of $f(12)$ suffices to show $f(k) > 1.661$. $\square$

Corollary 2.5.1, Theorem 4.1.1, and Lemma 4.1.1 immediately imply Theorem 4.0.1, *i.e.* the $n^{1.661}$ time-space lower bound for SAT.

We use the inductive argument described earlier to prove a relation between Σ_k and Π_k for all $k \geq 2$, from which the theorem will follow. Define an expression e by the inductive definition

$$e(2) := \frac{c^2}{2}, e(k) := \frac{c^2}{k}\left(\prod_{i=1}^{k-1} e(i)\right). \tag{4.3}$$

Lemma 4.1.2 *Assume* $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ *holds for some* $c \geq 1$*, and let* $k \geq 2$ *be an integer. Then*

$$\Sigma_k\mathsf{TIME}[n] \subseteq \Pi_k\mathsf{TIME}[n^{e(k)+o(1)}].$$

Proof. By induction on k. The case $k = 2$ is exactly the $n^{\sqrt{2}}$ lower bound of equation (4.1). We revisit it for completeness. Assuming $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$,

$$\Sigma_2\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^{c^2}] \subseteq \Pi_2\mathsf{TIME}[n^{c^2/2+o(1)}] = \Pi_2\mathsf{TIME}[n^{e(2)+o(1)}],$$

where the last inclusion follows from Corollary 3.3.1.

Induction Hypothesis: Assume for all $i \in \{2, \dots, k-1\}$ that $\Sigma_i\mathsf{TIME}[n] \subseteq \Pi_i\mathsf{TIME}[n^{e(i)+o(1)}]$.

We now prove the theorem for general k. The Slowdown Lemma (Lemma 3.3.1) and induction hypothesis imply

$$\Sigma_\ell\mathsf{TIME}[n] \subseteq \Sigma_{\ell-1}\mathsf{TIME}[n^{e(\ell-1)+o(1)}], \text{ for } \ell \in \{2, \dots, k-1\}.$$

If $e(i) < 1$ for some $i \in \{2, \dots, k-1\}$, then we have a contradiction ($\Sigma_i\mathsf{TIME}[n] \subseteq \Pi[i]\mathsf{TIME}[o(n)]$), so $\Sigma_k\mathsf{TIME}[n] \subseteq \Pi_k\mathsf{TIME}[n^{e(k)+o(1)}]$ holds vacuously. If $e(i) \geq 1$ for all i, then by padding

$$\begin{aligned}\Sigma_k\mathsf{TIME}[n] &\subseteq \Sigma_{k-1}\mathsf{TIME}[n^{e(k-1)+o(1)}] \\ &\subseteq \Sigma_{k-2}\mathsf{TIME}[n^{(e(k-1)+o(1))(e(k-2)+o(1))}] \subseteq \cdots \subseteq \Sigma_2\mathsf{TIME}[n^{\prod_{i=2}^{k-1}(e(i)+o(1))}].\end{aligned}$$

But it is also the case that

$$\begin{aligned}\Sigma_2\mathsf{TIME}[n^{\prod_{i=2}^{k-1}(e(i)+o(1))}] &\subseteq \mathsf{NTIME}[n^{c\prod_{i=2}^{k-1}(e(i)+o(1))}] \\ &\subseteq \quad \mathsf{DTS}[n^{c^2\prod_{i=2}^{k-1}(e(i)+o(1))}] \subseteq \Pi_k\mathsf{TIME}[n^{\frac{c^2}{k}\prod_{i=2}^{k-1}(e(i)+o(1))}] \\ &\subseteq \quad \Pi_k\mathsf{TIME}[n^{e(k)+o(1)}],\end{aligned}$$

where the penultimate inclusion follows from Corollary 3.3.1 (Fortnow and Van Melkebeek's simulation), and the last inclusion follows by definition of $e(k)$. $\square$

We are finally ready to prove the $\Omega(n^{1.661})$ lower bound.

Proof of Theorem 4.1.1. Let k' be the smallest integer such that $c < f(k')$; such a k' exists since f is monotonically increasing (Lemma 4.1.1). First, consider when $k' = 2$. If $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$, then Lemma 4.1.2 implies

$$\Sigma_2\mathsf{TIME}[n] \subseteq \Pi_2\mathsf{TIME}[n^{e(2)+o(1)}] = \Pi_2\mathsf{TIME}[n^{c^2/2+o(1)}].$$

Now if $c < 2^{1/2} = (1 + 1/1)^{1/2} = f(2)$, then $e(2) < 1$. Therefore the above inclusion contradicts the "No Complementary Speedup" Theorem (Theorem 3.3.1), and this concludes the base case. Otherwise, observe that $c \geq 2^{1/2}$ implies $e(2) \geq 1$. In fact, the following relationship holds between the expressions e and f.

Claim 1 *For all* i, $e(i) \geq 1 \iff c \geq f(i)$.

Proof of Claim 1. Recall that $f(k) := \prod_{j=1}^{k-1}(1+1/j)^{1/2^j}$, and $e(1) := 1$, $e(k) := \frac{c^2}{k}\left(\prod_{i=1}^{k-1} e(i)\right)$. First, we claim that $e(i) = \frac{c^{2^{i-1}}}{i!\prod_{j=2}^{i-2}(j!)^{2^{i-j-2}}}$ follows from a proof by induction. The denominator can be simplified further to get $e(i) = \frac{c^{2^{i-1}}}{i\cdot\prod_{j=2}^{i-1} j^{2^{i-j-1}}}$.

For all i, let c_i be the unique number in $(1,2)$ such that $e(i) = 1$ when $c = c_i$, *i.e.* $(c_i)^{2^{i-1}} = i \cdot \prod_{j=2}^{i-1} j^{2^{i-j-1}}$. It suffices to show that $c_i = f(i)$. Observe

$$(c_{i-1})^{2^{i-1}} = ((c_{i-1})^{2^{i-2}})^2 = (i-1)^2 \cdot \prod_{j=2}^{i-2} j^{2^{i-j-1}}$$

by definition of c_{i-1}. Hence

$$\left(\frac{c_i}{c_{i-1}}\right)^{2^{i-1}} = \frac{i \cdot \prod_{j=2}^{i-1} j^{2^{i-j-1}}}{(i-1)^2 \prod_{j=2}^{i-2} j^{2^{i-j-1}}} = i/(i-1),$$

so $\frac{c_i}{c_{i-1}} = \left(\frac{i}{i-1}\right)^{1/2^{i-1}}$. Therefore

$$\begin{aligned} c_i &= (c_i/c_{i-1})(c_{i-1}/c_{i-2})\cdots(c_3/c_2)c_2 \\ &= \prod_{j=2}^{i}\left(1+\frac{1}{j-1}\right)^{\frac{1}{2^{j-1}}} = \prod_{j=1}^{i-1}\left(1+\frac{1}{j}\right)^{\frac{1}{2^j}} = f(i). \end{aligned}$$

$\square$

Claim 1 and our choice of k' implies that k' is the smallest integer such that $e(k') < 1$. Therefore for all $i \le k'-1$ we have that $e(i) \ge 1$, so Lemma 4.1.2 applies. Namely,

$$\Sigma_{k'}\mathsf{TIME}[n] \subseteq \Pi_{k'}\mathsf{TIME}[n^{e(k')+o(1)}].$$

However, the above inclusion contradicts the "No Complementary Speedup" Theorem (Theorem 3.3.1), since $e(k') < 1$. This completes the proof of Theorem 4.1.1. $\square$

4.2 Sat is not in $\mathsf{DTISP}[n^{1.732}, n^{o(1)}]$

The above lower bound can be improved upon, by finding an additional way to apply the assumption $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$ within the argument. At a high level, the $n^{1.661}$ lower bound performs the following moves:

- Start with $\Sigma_k\mathsf{TIME}[n]$.
- Inductively apply inclusions to obtain $\Sigma_k\mathsf{TIME}[n] \subseteq \Sigma_2\mathsf{TIME}[n^e]$ for some e.
- Apply $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^e, n^{o(1)}]$ twice to get $\Sigma_2\mathsf{TIME}[n^e] \subseteq \mathsf{DTISP}[n^{ec^2}, n^{o(1)}]$.
- Apply the kth root speedup to obtain $\mathsf{DTISP}[n^{ec^2}, n^{o(1)}] \subseteq \Pi_k\mathsf{TIME}[n^{ec^2/k}, n^{o(1)}]$.

Every step of the argument uses the assumption that $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$, except for the last one. That is, Fortnow and Van Melkebeek's result (Corollary 3.3.1 of the Speedup Lemma) that $\mathsf{DTISP}[t, t^{o(1)}] \subseteq \Pi_k\mathsf{TIME}[t^{1/k+o(1)}]$ holds *unconditionally*, whereas all other inclusions derived in the above actually depended on the assumption that $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$. Our next lower bound carefully exploits this assumption to get

$$\mathsf{DTS}[t] = \mathsf{DTISP}[t, t^{o(1)}] \subseteq \Pi_2\mathsf{TIME}[t^{1/(2+\delta)+o(1)}],$$

for some $\delta > 0$ that depends on the constant c. This new containment allows us to push the lower bound above $n^{\sqrt{3}}$.

Lemma 4.2.1 *Let $c \in (1, 2)$. Define the sequence $d(1) := 2$, $d(k) := 1 + \frac{d(k-1)}{c}$. If $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ then for all $k \geq 1$,*

$$\mathsf{DTS}[n^{d(k)}] \subseteq (\forall n)(\exists\ \log n)\mathsf{DTS}[n] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}].$$

Let us briefly outline how the proof of Lemma 4.2.1 goes. As the statement suggests, it is an inductive argument, but of a different kind than before. First, we derive

$$\mathsf{NTIME}[n^\ell] \subseteq \mathsf{DTS}[n^{\ell c}] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}]$$

for some $\ell > 1$. Then, this containment can be used to obtain $\mathsf{DTS}[n^d] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}]$, for some $d > 2$ that depends on c. In turn, this DTS in Π_2 simulation can be applied to obtain

$$\mathsf{NTIME}[n^{\ell'}] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}]$$

for some $\ell' > \ell$. That is, the two containments of DTS in $\Pi_2\mathsf{TIME}$ and NTIME in $\Pi_2\mathsf{TIME}$ can mutually improve upon each other, and the amount of improvement that can be achieved depends on the constant c.

Proof of Lemma 4.2.1. By induction on k. The $k = 1$ case is trivial since $\mathsf{DTS}[n^2] \subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n]$ holds unconditionally (Corollary 3.3.1 of the Speedup Lemma).

For the inductive step, assume $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ and $\mathsf{DTS}[n^{d(k)}] \subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n]$. Observe that for $c < 2$, $d(k) \geq c$, since by induction we have that $d(k) = 1 + d(k-1)/c \geq 1 + 1 = 2$. By padding and the inductive hypothesis,

$$\mathsf{NTIME}[n^{d(k)/c}] \subseteq \mathsf{DTS}[n^{d(k)}] \subseteq (\forall n)(\exists\ \log n)\mathsf{DTS}[n]. \tag{4.4}$$

Now consider a Π_2 simulation of $\mathsf{DTS}[n^{1+d(k)/c}]$, where only $O(n)$ bits (that is, $n^{1-o(1)}$ configurations of the DTS machine) are guessed in the universal quantifier. (Formally, we are invoking the Speedup Lemma, with $b = n^{1-o(1)}$.) Written in our class notation, the resulting simulation is expressed by the inclusion:

$$\mathsf{DTS}[n^{1+d(k)/c}] \subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n^{d(k)/c}].$$

The $(\exists\ \log n)[\cdots]$ part in the above corresponds to an NTIME computation that takes an input of $O(n)$ bits (the input x, plus the list of configurations) and runs in $n^{d(k)/c+o(1)}$ time. By equation (4.4) above, this nondeterministic computation can be replaced with a Π_2 computation running in $n^{1+o(1)}$ time. Therefore $\mathsf{DTS}[n^{1+d(k)/c}] \subseteq (\forall\ n)(\forall\ n)(\exists\ \log n)\mathsf{DTS}[n] = (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}]$. $\Box$

The lemma immediately implies a better way to simulate DTS in Π_2 when we assume a sub-quadratic algorithm for SAT.

Theorem 4.2.1 (Conditional Speedup) *Let $c \in (1,2)$. If $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ then for all $\varepsilon > 0$,*

$$\mathsf{DTS}\left[n^{\frac{c}{c-1}-\varepsilon}\right] \subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}].$$

Observe that when $\varepsilon > \frac{1}{c-1}$, the theorem is trivial ($\frac{c}{c-1} - \varepsilon < 1$).

Proof. First, note that for any $c < 2$, the sequence $\{d(k)\}_{k\in\mathbb{N}}$ is monotone increasing. The proof is by induction: $d(3) = 1+2/c > 2 = d(2)$, and when $d(k) > d(k-1)$, we have $d(k+1) = 1+d(k)/c > 1 + d(k-1)/c = d(k)$.

Secondly, $\{d(k)\}_{k\in\mathbb{N}}$ converges to a constant, given by $d_\infty = 1 + \frac{d_\infty}{c}$. Solving, we obtain $d_\infty = c/(c-1)$. Hence for any fixed $\varepsilon > 0$, there is a finite K such that $d(K) \geq \frac{c}{c-1} - \varepsilon$. Therefore $\mathsf{DTS}\left[n^{\frac{c}{c-1}-\varepsilon}\right] \subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n] \subseteq \Pi_2\mathsf{TIME}[n^{1+o(1)}]$ by Lemma 4.2.1. $\Box$

The conditional speedup theorem arms us with an additional lower bound tool. Let us combine Theorem 4.2.1 with the inductive argument from Section 4.1. We know that if $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, then $c \geq 1.661 > \phi$. It follows that $c^2 > c/(c-1) - \varepsilon$ for all $\varepsilon > 0$. Now for all $\varepsilon > 0$ and sufficiently small $\varepsilon_2 > 0$, there is an $\varepsilon_1 > 0$ such that

$$\begin{aligned}
\Sigma_2\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^{c^2}] &\subseteq \mathsf{DTS}[\left(n^{c^2\cdot(\frac{c-1}{c}+\varepsilon_1)}\right)^{c/(c-1)-\varepsilon}] \\
&\subseteq \mathsf{DTS}[\left(n^{c(c-1)+\varepsilon_2}\right)^{c/(c-1)-\varepsilon}] \\
&\subseteq \Pi_2\mathsf{TIME}[n^{c\cdot(c-1)+\varepsilon_2+o(1)}],
\end{aligned}$$

where the penultimate inclusion follows by taking $\varepsilon_1 = \varepsilon_2/c^2$, and the last inclusion follows from Theorem 4.2.1. Observe that this new inclusion of Σ_2 linear time in Π_2 time is *superior* to the previously derived $\Sigma_2\mathsf{TIME}[n] \subseteq \Pi_2\mathsf{TIME}[n^{c^2/2+o(1)}]$ inclusion, for all $c < 2$. More precisely, $c(c-1) < c^2/2$ for all $c \in (1,2)$. Notice that $c(c-1) = 1$ precisely when c is the golden ratio ϕ. Thus in a sense, instead of having Lipton-Viglas' $n^{\sqrt{2}}$ lower bound as our base case, the above is an inclusion that resembles Fortnow-Van Melkebeek's n^ϕ lower bound as a base case.

Proceeding inductively as in Section 4.1, we derive for all sufficiently small $\varepsilon_3 > 0$ that

$$\begin{aligned}
\Sigma_3\mathsf{TIME}[n] \subseteq \Sigma_2\mathsf{TIME}[n^{c\cdot(c-1)+\varepsilon_2+o(1)}] &\subseteq \mathsf{DTS}[n^{c^3\cdot(c-1)+\varepsilon_2 c^2}] \\
&\subseteq \Pi_3\mathsf{TIME}[n^{\frac{c^3\cdot(c-1)}{3}+\varepsilon_3+o(1)}],
\end{aligned}$$

by setting $\varepsilon_2 = 3\varepsilon_3/c^2$.

Similarly, for all sufficiently small $\varepsilon_4 > 0$ we obtain

$$\begin{aligned}\Sigma_4\mathsf{TIME}[n] \subseteq \Sigma_3\mathsf{TIME}[n^{\frac{c^3\cdot(c-1)}{3}+\varepsilon_3+o(1)}] &\subseteq \Sigma_2\mathsf{TIME}[n^{\frac{c^4\cdot(c-1)^2}{3}+c(c-1)\varepsilon_3+o(1)}]\\ &\subseteq \mathsf{DTS}[n^{\frac{c^6\cdot(c-1)^2}{3}+c^3(c-1)\varepsilon_3}]\\ &\subseteq \Pi_4\mathsf{TIME}[n^{\frac{c^6\cdot(c-1)^2}{12}+\varepsilon_4+o(1)}],\end{aligned}$$

by setting $\varepsilon_4 = 4\varepsilon_3/(c^3(c-1))$.

We can formally state the new relation between Σ_k and Π_k for general k as follows. Define $g(2) := c(c-1)$, and for $k \geq 3$, define

$$g(k) := \frac{c^{3\cdot 2^{k-3}}(c-1)^{2^{k-3}}}{k\cdot\left(\prod_{i=3}^{k-1} i^{2^{(k-1)-i}}\right)}.$$

Observe that $g(3) = c^3(c-1)/3$ and $g(4) = c^{3\cdot 2}(c-1)^2/(4\cdot 3) = c^6(c-1)^2/12$.

Lemma 4.2.2 *Assume* $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, *and let* $k \geq 2$ *be an integer. Then*

$$\Sigma_k\mathsf{TIME}[n] \subseteq \Pi_k\mathsf{TIME}[n^{g(k)+o(1)}].$$

Proof. By induction on k. □

Finally, we are in position to improve our previous lower bound for SAT (Theorem 4.0.1) to $\Omega(n^{1.7327})$ time on subpolynomial space machines.

Proof of Theorem 4.0.2. By Corollary 2.5.1, it suffices for us to show $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^{1.7327}]$. Assuming $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$, we wish to find the largest c possible such that $g(k) < 1$ for some k. (Note that, as with the function f, the function g is also monotone decreasing.) By Lemma 4.2.2, such a c implies that $\Sigma_k\mathsf{TIME}[n] \subseteq \Pi_k\mathsf{TIME}[o(n)]$, and therefore yields a contradiction with the 'No Complementary Speedup' theorem. Observe that the function $g(k)$ can be simplified to

$$\begin{aligned}g(k) &= \frac{c^{3\cdot 2^{k-3}}(c-1)^{2^{k-3}}}{k\cdot(3^{2^{k-4}}\cdot 4^{2^{k-5}}\cdot 5^{2^{k-6}}\cdots(k-1))}\\ &= \left(\frac{c^3(c-1)}{k^{2^{-k+3}}\cdot(3^{2^{-1}}\cdot 4^{2^{-2}}\cdot 5^{2^{-3}}\cdots(k-1)^{2^{-k+3}})}\right)^{2^{k-3}}.\end{aligned}$$

Now, $g(k) < 1$ if and only if

$$g'(k) := \frac{c^3(c-1)}{k^{2^{-k+3}}\cdot(3^{2^{-1}}\cdot 4^{2^{-2}}\cdot 5^{2^{-3}}\cdots(k-1)^{2^{-k+3}})} < 1,$$

so it suffices to analyze the latter expression.

The denominator of $g'(k)$ numerically converges to $3.81213\cdots$ as $k \to \infty$. Therefore the calculation of c reduces to finding the positive root of $c^3\cdot(c-1) = 3.81213$, or $c \approx 1.7327$. □

There is still some slack that remains in our above lower bound. Notice that we used a conditional speedup for DTS in Π_2, but only applied the old unconditional speedup for DTS in Π_k when $k \geq 3$. Mending this discrepancy, Diehl and Van Melkebeek [DvM06] observed that our Conditional Speedup Theorem 4.2.1 also implies a conditional speedup for DTS in $\Pi_k\mathsf{TIME}$, for all $k \geq 2$.

Theorem 4.2.2 (General Conditional Speedup [DvM06]) *If $c < 2$ satisfies $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, then for all $k \geq 1$ and $e < k + 1/(c-1)$,*

$$\mathsf{DTS}[n^e] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_{k+1}\mathsf{TIME}[n^{1+o(1)}].$$

Proof. (Sketch) The proof of Lemma 3.3.2 (Fortnow and Van Melkebeek's speedup) implies that any machine in $\mathsf{DTS}[n^e]$ can be simulated by a machine N in $\Sigma_k\mathsf{TIME}$, where the k quantifiers of N each guess $n^{1+o(1)}$ bits, followed by a deterministic time predicate R that runs in $n^{e-(k-1)} = O(n^{1+1/(c-1)-\varepsilon})$ time for an $\varepsilon > 0$.

By our conditional speedup (Theorem 4.2.1), the predicate R can be simulated by a machine M_1 in $\Pi_2\mathsf{TIME}[n^{1+o(1)}]$, and also by a machine M_2 in $\Sigma_2\mathsf{TIME}[n^{1+o(1)}]$. If k is odd, R is replaced in N with M_1, otherwise R is replaced with M_2. The resulting N has only one more alternation and runs in $n^{1+o(1)}$ time. $\square$

When one plugs in the above speedup in lieu of the unconditional speedup, the lower bound exponent increases from 1.7327 to 1.759. For more details, cf. [DvM06].

4.3 SAT is not in $\mathsf{DTISP}[n^{1.784}, n^{o(1)}]$

While Diehl and Van Melkebeek's argument makes greater use of the assumption $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]$ than the previous arguments, theirs is still not yet optimal. We show how to prove even better conditional speedups for classes with more alternations, leading to a small improvement in the lower bound.

Theorem 4.3.1 SAT *is not in $\mathsf{DTS}[n^c]$ for all $c < 1.784$.*

We note in passing that it is not necessary to know the proof of Theorem 4.3.1 in order to understand the final $n^{2\cos(\pi/7)}$ lower bound, so the reader interested purely in that result need not read this section.

Our assortment of tools allows us to obtain

$$\begin{aligned}
\Sigma_2\mathsf{TIME}[t] &\subseteq (\exists t)\mathsf{DTS}[t^c] \text{ by Slowdown (Lemma 3.3.1)} \\
&\subseteq \mathsf{DTS}[t^{c^2}] \text{ by Slowdown} \\
&\subseteq (\Sigma_3 \cap \Pi_3)\mathsf{TIME}\left[t^{\frac{c^2}{2+1/(c-1)}+o(1)}\right], \text{ by General Conditional Speedup (Theorem 4.2.2)}
\end{aligned}$$

for time functions t such that $t^{c^2} \geq n^{2+1/(c-1)}$. Just as we built upon an inclusion of NTIME in Π_2 to derive a better speedup of DTS in Π_2, our strategy is to build upon the above inclusion of Σ_2 in Π_3 to inductively obtain a better conditional speedup theorem for DTS on machines that take at least three alternations.

The first inductive step. These conditional speedups take the form of the proof in Lemma 4.2.1. That is, we suppose $\mathsf{DTS}[n^e] \subseteq \Pi_3\mathsf{TIME}[n^{1+o(1)}]$ for all $\varepsilon > 0$, where e is as large as possible, and try to show that $\mathsf{DTS}[n^{e'}]$ is also in $\Pi_3\mathsf{TIME}[n^{1+o(1)}]$, for some $e' > e$. Notice that from Theorem 4.2.2, we know that $e \geq 2 + 1/(c-1)$ holds already.

Consider $\mathsf{DTS}[n^{1+e\cdot(1+1/(c-1))/c^2}]$. We wish to show that this class is in $\Pi_3\mathsf{TIME}[n^{1+o(1)}]$ as well. Assuming that $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, we derive the following sequence:

$$\begin{aligned}
\mathsf{DTS}[n^{1+e\cdot\frac{(1+1/(c-1))}{c^2}}] &\subseteq (\forall\ n)(\exists\ \log n)\mathsf{DTS}[n^{e/c^2\cdot(1+1/(c-1))}] \text{ by Speedup (Lemma 3.3.2)} \\
&\subseteq (\forall\ n)(\exists\ n^{e/c^2})(\forall \log n)\mathsf{DTS}[n^{e/c^2}] \text{ by Cond. Spdup (Thm 4.2.1) \& padding} \\
&\subseteq (\forall\ n)(\exists\ n^{e/c^2})\mathsf{DTS}[n^{e/c}] \text{ by Slowdown (Lemma 3.3.1)} \\
&\subseteq (\forall\ n)\mathsf{DTS}[n^e] \text{ by Lemma 3.3.1} \\
&\subseteq \Pi_3\mathsf{TIME}[n^{1+o(1)}] \text{ by assumption.}
\end{aligned}$$

Notice that in the above, we require that $e/c^2 \geq 1$. This is true, so long as

$$c^2 \leq 2 + 1/(c-1),$$

or $c < 1.8019\ldots$.

Is the above simulation of DTS in $\Pi_3\mathsf{TIME}$ any better than the general conditional speedup of Theorem 4.2.2? For $e = 2 + 1/(c-1)$, one finds that $1 + e \cdot (1 + 1/(c-1))/c^2 > e$ when $c^2 < 2 + 1/(c-1)$. Numerically, this occurs when $c < 2\cos(\pi/7) \approx 1.8019$. (We will prove this later.) In other words, the above simulation of DTS in $\Pi_3\mathsf{TIME}$ is indeed an improvement over Theorem 4.2.2, but only when $c < 1.8019$. Therefore this new speedup simulation is only effective for proving lower bounds up to $n^{2\cos(\pi/7)}$. For now we shall not worry about this limitation, but keep in the back of our mind that we must return to it in the future.

Since we started with the hypothesis $\mathsf{DTS}[n^e] \subseteq \Pi_3\mathsf{TIME}[n^{1+o(1)}]$, and concluded with the inclusion $\mathsf{DTS}[n^{1+e/c^2\cdot(1+1/(c-1))}] \subseteq \Pi_3\mathsf{TIME}[n^{1+o(1)}]$, the implication naturally suggests an inductive argument along the lines of our original conditional speedup (Theorem 4.2.1). Consider the sequence defined by

$$e(1) := 2 + 1/(c-1), \quad e(k+1) := 1 + e(k) \cdot (1 + 1/(c-1))/c^2.$$

As in the $n^{\sqrt{3}}$ lower bound, it is easy to prove that this sequence is increasing and convergent.

Claim 2 *For $c^2 < 2 + 1/(c-1)$, the sequence $\{e_k\}$ is monotone increasing, and converges to e_∞ where $e_\infty = 1 + e_\infty \cdot (1 + 1/(c-1))/c^2$. That is, $e_\infty = \frac{c(c-1)}{c(c-1)-1} = 1 + \frac{1}{c(c-1)-1}$.*

Proof. The base case holds, since

$$\begin{aligned} e(2) > e(1) &\iff 1 + \left(\frac{2+1/(c-1)}{c^2}\right) \cdot (1+1/(c-1)) > 2+1/(c-1) \\ &\iff \left(\frac{2+1/(c-1)}{c^2}\right) \cdot (1+1/(c-1)) > 1+1/(c-1) \\ &\iff 2+1/(c-1) > c^2. \end{aligned}$$

For the induction step,

$$\begin{aligned} e(k+1) > e(k) &\iff 1 + \left(\frac{e(k)}{c^2}\right) \cdot (1+1/(c-1)) > 1 + \left(\frac{e(k-1)}{c^2}\right) \cdot (1+1/(c-1)) \\ &\iff e(k) > e(k-1). \end{aligned}$$

$\square$

The claim immediately implies a new conditional speedup for DTS in $\Sigma_k\mathsf{TIME}$.

Lemma 4.3.1 (Speedup for Σ_3) *For all $\varepsilon > 0$, if there is $c^2 < 2+1/(c-1)$ such that $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, then*

$$\mathsf{DTS}\left[n^{1+\frac{1}{c(c-1)-1}-\varepsilon}\right] \subseteq \Sigma_3\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_3\mathsf{TIME}[n^{1+o(1)}].$$

The requirement on c in the lemma is equivalent to $c < 1.8019\ldots$. This curious constant will arise in the next lower bound, where we show $\Omega(n^{1.8019})$ can actually be achieved.

Corollary 4.3.1 *For all $\varepsilon > 0$, if there is $c^2 < 2+1/(c-1)$ such that $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ then*

$$\mathsf{DTS}\left[n^{k+\frac{1}{c(c-1)-1}-\varepsilon}\right] \subseteq \Sigma_{k+2}\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_{k+2}\mathsf{TIME}[n^{1+o(1)}].$$

Proof. Analogous to the proof of the General Conditional Speedup, *i.e.* Theorem 4.2.2. $\square$

The Speedup for Σ_3 Lemma leads to a better lower bound exponent, when one applies it to the alternation-switching argument. For instance, the previous section shows that

$$\Sigma_2\mathsf{TIME}[n] \subseteq \mathsf{NTIME}[n^c] \subseteq \mathsf{DTS}[n^{c^2}] \subseteq \Pi_2\mathsf{TIME}[n^{c^2/(1+1/(c-1))+o(1)}] = \Pi_2\mathsf{TIME}[n^{c(c-1)+o(1)}],$$

therefore

$$\Sigma_3\mathsf{TIME}[n] \subseteq \Sigma_2\mathsf{TIME}[n^{c(c-1)+o(1)}] \subseteq \mathsf{DTS}[n^{c^3(c-1)}] \subseteq \Pi_3\mathsf{TIME}\left[n^{\frac{c^3(c-1)}{1+1/(c(c-1)-1)}+o(1)}\right].$$

Let d_k be such that $\Sigma_k\mathsf{TIME}[n] \subseteq \Pi_k\mathsf{TIME}[n^{d_k+o(1)}]$. Then

$$\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}\left[n^{c^2\prod_{j=2}^{k-1} d_j}\right] \subseteq \Pi_k\mathsf{TIME}\left[n^{\frac{c^2}{k-2+1/(c(c-1)-1)}\cdot\prod_{j=2}^{k-1} d_j+o(1)}\right].$$

From the above, it follows that $d_2 = c(c-1)$, and $d_k = c^2/(k-2+1/(c(c-1)-1)) \cdot \prod_{j=2}^{k-1} d_j$. By substituting d_{k-1} ($k \geq 4$) for its corresponding expression in d_k, this can be rewritten as:

$$d_k = \frac{c^4 \left(\prod_{j=2}^{k-2} d_j\right)\left(\prod_{j=2}^{k-2} d_j\right)}{(k-2+1/(c(c-1)-1))(k-3+1/(c(c-1)-1))} = (d_{k-1})^2 \cdot \frac{k-2+1/(c(c-1)-1)}{k-3+1/(c(c-1)-1)}.$$

Solving for $d_k > 1$ in the range $c = (\phi, 2)$, where ϕ is the golden ratio, one obtains $c > 1.7802$ when $d_{14} \leq 1$.

The next step. Similarly, it is possible to use what was derived above to prove a better inclusion of Σ_3 in Π_4. Namely,

$$\Sigma_3\mathsf{TIME}[t] \subseteq \Sigma_2\mathsf{TIME}[t^{c(c-1)+o(1)}] \subseteq (\exists t^{c(c-1)})\mathsf{DTS}[t^{c^2(c-1)}] \subseteq \Pi_4\mathsf{TIME}\left[t^{\frac{c^2(c-1)}{2+1/(c(c-1)-1)}+o(1)}\right].$$

The above inclusion produces a better speedup of DTS in $\Pi_4 \cap \Sigma_4$. The induction proceeds similarly as before: suppose $\mathsf{DTS}[n^e] \subseteq \Sigma_4\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_4\mathsf{TIME}[n^{1+o(1)}]$. Then

$$\Sigma_3\mathsf{TIME}[n^{e/(c^3(c-1))}] \subseteq \mathsf{DTS}[n^e] \subseteq \Pi_4\mathsf{TIME}[n^{1+o(1)}].$$

Consider $\mathsf{DTS}[n^{1+e(1+1/(c(c-1)-1))/(c^3(c-1))}]$. This class is contained in

$$\begin{aligned} &(\exists\ n)(\forall \log n)\mathsf{DTS}[n^{e(1+1/(c(c-1)-1))/(c^3(c-1))}] \\ \subseteq\ &(\exists\ n)(\forall \log n)(\forall\ n^{e/(c^3(c-1))})(\exists\ n^{e/(c^3(c-1))})(\forall \log n)\mathsf{DTS}[n^{e/(c^3(c-1))}], \end{aligned}$$

which is contained in $\Pi_4\mathsf{TIME}[n^{1+o(1)}]$ by the above derivation. As before, we get a non-decreasing sequence of exponents, which converges to $e_\infty = 1 + \frac{1}{c^2(c(c-1)-1)-1}$. Thus for all $\varepsilon > 0$,

$$\mathsf{DTS}\left[n^{1+\frac{1}{c^2(c(c-1)-1)-1}-\varepsilon}\right] \subseteq \Pi_4\mathsf{TIME}[n^{1+o(1)}].$$

However, the above speedup is only an improvement over the speedup implied by Corollary 4.3.1 (that is, $\mathsf{DTS}[n^{2+1/(c(c-1)-1)-\varepsilon}] \subseteq \Pi_4\mathsf{TIME}[n^{1+o(1)}]$) when $c < 1.7859$. Carrying out the alternation-switching argument as before, for $k \geq 4$,

$$\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^{c^2 \cdot \prod_{j=2}^{k-1} d_j}] \subseteq \Pi_k\mathsf{TIME}\left[n^{\frac{c^2}{k-3+(1+\frac{1}{c^2(c(c-1)-1)-1})}\cdot\prod_{j=2}^{k-1} d_j + o(1)}\right].$$

Therefore $d_k \leq \frac{c^2}{k-3+(1+\frac{1}{c^2(c(c-1)-1)-1})} \cdot \prod_{j=2}^{k-1} d_j$; that is,

$$d_k \leq d_{k-1}^2 \frac{k-3+(1+1/(c^2(c(c-1)-1)-1))}{k-4+(1+1/(c^2(c(c-1)-1)-1))} = d_{k-1}^2 \cdot \left(1 + \frac{1}{k-3+1/(c^2(c(c-1)-1)-1)}\right).$$

Assuming $d_9 \geq 1$, we find that $c > 1.7829\ldots$, using the above along with our previously derived bounds on d_2 and d_3.

Since the above derived speedup of DTS in Π_4 is only an improvement for $c < 1.7859\ldots$, our lower bound on c cannot be improved by much more with this particular inductive extension. By further argument along the same lines (but for higher levels of the polynomial hierarchy), one can show the lower bound is at least n^c time and $n^{o(1)}$ space, where $c \geq 1.7843$. We omit the details, as the next section shows that one can prove an even better lower bound by taking a different approach.

4.4 SAT is not in $\mathsf{DTISP}[n^{2\cos(\pi/7)-\varepsilon}, n^{o(1)}]$

In the previous section, when we developed the Speedup for Σ_3 Lemma, we ran into an annoying obstacle: our new speedup theorem only works for proving time lower bounds less than $n^{2\cos(\pi/7)}$, or $n^{1.8019}$. In this section we prove that this limitation can be fully reached: an $\Omega(n^{2\cos(\pi/7)-\varepsilon})$ time lower bound is possible.

Theorem 4.4.1 *For all c satisfying $c^3 - c^2 - 2c + 1 < 0$, $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^c]$.*

For the sake of mathematical curiosity, let us first make a few remarks about the constant $2\cos(\pi/7) = 1.8019\ldots$ and how it arises. Whereas the golden ratio is the unique solution in the interval $(1, 2)$ to the equation $c^2 = 1 + 1/(c-1)$, our quantity is the unique solution in $(1, 2)$ to $c^2 = 2+1/(c-1)$. We can give a rough informal explanation for the origins of these equations. The golden ratio lower bound effectively shows how to speed up a deterministic small-space computation by a $(1 + 1/(c-1))$th root, using two quantifiers. Removing these two quantifiers (to obtain a deterministic computation again) multiplies the exponent by a factor of c^2. If $c^2 < 1 + 1/(c-1)$, then the speedup exceeded the slowdown, hence the resulting deterministic computation runs faster than the original– a contradiction. Our $n^{2\cos(\pi/7)}$ lower bound is more intricate, achieving a speedup with two quantifiers by a $(2 + 1/(c-1))$th root, by invoking a $(1 + 1/(c-1))$th root speedup for multiple times.

To give a trigonometric reason for the appearance of $2\cos(\pi/7)$, let us compute the roots of the polynomial $p(c) = c^3 - c^2 - 2c + 1$. With a little foresight we let $c = 2\cos(u)$, and recall that $2\cos(x) = e^{ix} + e^{-ix}$. Plugging into p, we get

$$\begin{aligned} p(c) &= (e^{iu} + e^{-iu})^3 - (e^{iu} + e^{-iu})^2 - 2(e^{iu} + e^{-iu}) + 1 \\ &= (e^{3iu} + e^{-3iu}) - (e^{2iu} + e^{-2iu}) + (e^{iu} + e^{-iu}) - 1, \end{aligned}$$

after simplifying. Multiplying by e^{3iu}, the above sum becomes

$$(e^{6iu} + 1) - (e^{5iu} + e^{iu}) + (e^{4iu} + e^{2iu}) - e^{3iu},$$

which is

$$e^{6iu} - e^{5iu} + e^{4iu} - e^{3iu} + e^{2iu} - e^{iu} + 1 = \frac{e^{7iu} + 1}{e^{iu} + 1}.$$

For $u \neq \pi$, the roots of p are given by $e^{7iu} = -1$, leading to the three equations:

$$e^{7u_1 i} = e^{\pi i},\ e^{7u_2 i} = e^{3\pi i},\ \text{and } e^{7u_3 i} = e^{5\pi i}.$$

So, the roots are $c_1 = 2\cos(\pi/7), c_2 = 2\cos(3\pi/7), c_3 = 2\cos(5\pi/7)$, where only c_1 is a positive real.

The crux of the proof for Theorem 4.4.1 is in the following result, which is a subtle combination of the golden ratio proof strategy of Fortnow *et al.* [FLvMV05] from Chapter 3.3 and our Conditional Speedup Theorem (Theorem 4.2.1).

Theorem 4.4.2 *Suppose $c < 2$ satisfies $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$. Then for all integers $k \geq 1$, and $d < c/(c-1)$,*

$$\mathsf{DTS}[n^{d+\sum_{i=1}^{k}(c^2/d)^i}] \subseteq \Sigma_2\mathsf{TIME}[n^{(c^2/d)^k+o(1)}] \cap \Pi_2\mathsf{TIME}[n^{(c^2/d)^k+o(1)}].$$

We first show how Theorem 4.4.2 implies Theorem 4.4.1.

Proof of Theorem 4.4.1. Assuming $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ and Theorem 4.4.2,

$$\Sigma_2\mathsf{TIME}[n^{d+\sum_{i=1}^{k}(c^2/d)^i}] \subseteq \mathsf{NTIME}[n^{c(d+\sum_{i=1}^{k}(c^2/d)^i)}] \subseteq \mathsf{DTS}[n^{c^2(d+\sum_{i=1}^{k}(c^2/d)^i)}] \subseteq \Pi_2\mathsf{TIME}[n^{c^2(c^2/d)^k+o(1)}].$$

A contradiction with the "No Complementary Speedup" Theorem is reached (and therefore $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^c]$) precisely when

$$d + \sum_{i=1}^{k}(c^2/d)^i > c^2 \cdot (c^2/d)^k,$$

that is, when

$$\begin{aligned} c^2 < \sum_{i=1}^{k}\left(\frac{c^2}{d}\right)^{i-k} + d \cdot \frac{d^k}{c^{2k}} \quad &\iff \quad c^2 < \sum_{j=0}^{k-1}\left(\frac{d}{c^2}\right)^j + d \cdot \left(\frac{d}{c^2}\right)^k \\ &\iff \quad c^2 < \frac{1-\left(\frac{d}{c^2}\right)^k}{1-\left(\frac{d}{c^2}\right)} + d \cdot \left(\frac{d}{c^2}\right)^k \end{aligned} \tag{4.5}$$

Note that $c^2 \geq d$, since Fortnow *et al.* [FLvMV05] proved that c must be at least the golden ratio, therefore $c(c-1) \geq 1$, *i.e.* $c^2 \geq c/(c-1) > d$. Therefore $1 > (d/c^2)^k$ for all $k \geq 1$, and

$$\lim_{k\to\infty}(d/c^2)^k = 0.$$

Hence for any $\varepsilon > 0$, one can set $d = c/(c-1) - \varepsilon$ and find a k such that $((c/(c-1)-\varepsilon)/c^2)^k \leq \varepsilon$, whereby the inequality (4.5) turns into

$$c^2 < \frac{1-\varepsilon}{1-\frac{c/(c-1)-\varepsilon}{c^2}} + (c/(c-1)-\varepsilon)\cdot\varepsilon.$$

Simple algebraic manipulation yields the equivalent condition:

$$c^2 - (c/(c-1)-\varepsilon) < (1-\varepsilon) + (c/(c-1)-\varepsilon)\cdot\varepsilon\cdot\left(1-\frac{c/(c-1)-\varepsilon}{c^2}\right)$$

Multiplying through by $(c-1)$, the condition becomes

$$\begin{aligned} c^2(c-1) - (c-\varepsilon(c-1)) \ &< \ ((c-1)-\varepsilon(c-1)) + (c-\varepsilon(c-1))\cdot\varepsilon\cdot\left(1-\frac{c/(c-1)-\varepsilon}{c^2}\right) \\ \iff c^3 - c^2 - 2c + 1 &< (c-\varepsilon(c-1))\cdot\varepsilon\cdot\left(1-\frac{c/(c-1)-\varepsilon}{c^2}\right) - 2\varepsilon(c-1) \end{aligned} \tag{4.6}$$

Now, as ε approaches 0, the RHS approaches 0. We arrive at the following condition implying a contradiction:

$$c^3 - c^2 - 2c + 1 < 0,$$

which is what we wanted to prove. That is, for any c satisfying $c^3 - c^2 - 2c + 1 < 0$, one can choose an $\varepsilon > 0$ such that c and ε satisfy inequality (4.6). $\square$

Proof of Theorem 4.4.2. By induction on k. Suppose $c < 2$ satisfies $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$. Pick d such that $c \leq d < c/(c-1)$. We only prove the containment for $\mathbf{\Sigma}_2$, as the proof for $\mathbf{\Pi}_2$ is analogous.

For $k = 0$, the task is just to show $\mathsf{DTS}[n^d] \subseteq \mathbf{\Sigma}_2\mathsf{TIME}[n^{1+o(1)}]$, which is precisely the Conditional Speedup Theorem (Theorem 4.2.1). For the inductive step, start with $\mathsf{DTS}[n^{d+\sum_{i=1}^{k}(c^2/d)^i}]$. Applying the Speedup Lemma (Lemma 3.3.2),

$$\mathsf{DTS}[n^{d+\sum_{i=1}^{k}(c^2/d)^i}] \subseteq (\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{DTS}[n^{d+\sum_{i=1}^{k-1}(c^2/d)^i}],$$

where the $\mathsf{DTS}[\cdots]$ part of the $\mathbf{\Sigma}_2$ computation has input of length $n + n^{o(1)}$ (the original input x, and two configurations). By the induction hypothesis,

$$(\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{DTS}[n^{d+\sum_{i=1}^{k-1}(c^2/d)^i}] \subseteq (\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathbf{\Pi}_2\mathsf{TIME}[n^{(c^2/d)^{k-1}+o(1)}].$$

Applying the Slowdown Lemma (Lemma 3.3.1) to the $\mathbf{\Pi}_2$ part (which takes input of length $n+n^{o(1)}$),

$$(\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathbf{\Pi}_2\mathsf{TIME}[n^{(c^2/d)^{k-1}+o(1)}] \subseteq (\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{coNTIME}[n^{c\cdot(c^2/d)^{k-1}+o(1)}].$$

Combining adjacent quantifiers of the same type,

$$(\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{coNTIME}[n^{c\cdot(c^2/d)^{k-1}+o(1)}] \subseteq (\exists\ n^{(c^2/d)^k})\mathsf{coNTIME}[n^{c\cdot(c^2/d)^{k-1}+o(1)}].$$

Now the input to the $\mathsf{coNTIME}$ part is $O(n^{(c^2/d)^k}) \leq O(n^{c\cdot(c^2/d)^{k-1}})$, since $d \geq c$. Therefore the Slowdown Lemma can be applied again to obtain

$$(\exists\ n^{(c^2/d)^k})\mathsf{coNTIME}[n^{c\cdot(c^2/d)^{k-1}+o(1)}] \subseteq (\exists\ n^{(c^2/d)^k})\mathsf{DTS}[n^{c^2\cdot(c^2/d)^{k-1}}].$$

But by the Conditional Speedup Theorem, the DTS part of the above class can be replaced with a $\mathbf{\Sigma}_2$ computation, in particular

$$(\exists\ n^{(c^2/d)^k})\mathsf{DTS}[n^{c^2\cdot(c^2/d)^{k-1}}] \subseteq (\exists\ n^{(c^2/d)^k})(\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{DTS}[n^{(c^2/d)^k}].$$

Finally,

$$(\exists\ n^{(c^2/d)^k})(\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{DTS}[n^{(c^2/d)^k}] \subseteq (\exists\ n^{(c^2/d)^k})(\forall\ \log n)\mathsf{DTS}[n^{(c^2/d)^k}]$$

by combining quantifiers. This completes the proof. $\square$

Note the above argument can be extended to prove a time-space tradeoff for SAT:

Corollary 4.4.1 *For all $c < 2\cos(\pi/7)$ there is a $d \in (0,1)$ such that* SAT *is not in* $\mathsf{DTISP}[n^c, n^d]$.

Proof. (Sketch) In the above proofs, one can replace the $n^{o(1)}$ space bound by n^d for a sufficiently small $d > 0$. The sizes of quantifiers in the alternating simulations increase only by an additive factor of $q_k d$ in the exponents, where q_k is a constant that depends on the (finite) sequence of speedup and slowdowns applied in the argument. $\square$

4.4.1 An Extension to Non-Uniform Algorithms

A straightforward application of an observation by Tourlakis [Tou01] further yields a lower bound on non-uniform algorithms for Sat. A non-uniform algorithm is a *collection of algorithms*, one for each possible input length n. The notation $\mathcal{C}/f(n)$ denotes the class of problems solved by non-uniform algorithms of type $\mathcal{C}$ whose program size for n bit inputs is $f(n)$. More formally, we define that $L \in \mathcal{C}/f(n)$ if there is a function $A : \mathbb{N} \to \Sigma^*$ such that $|A(n)| = f(n)$ for all n, and

$$x \in L \iff M(x, A(|x|)) \text{ accepts},$$

where M is an algorithm that solves a problem in class $\mathcal{C}$. We say that M *takes* $f(n)$ *bits of advice* in such a situation, and A is the *advice function* for M.

Corollary 4.4.2 $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTISP}[n^{1.801}, n^{o(1)}]/n^{o(1)}$.

One implication of the corollary is that Sat cannot be solved by any collection of deterministic $n^{1.801}$ time and $n^{o(1)}$ space non-uniform algorithms, where the algorithm for n bit inputs can be described in $n^{o(1)}$ bits.

Proof. (Sketch) We outline how all of the tools developed carry over to the non-uniform setting.

- First, we argue that a non-uniform extension to the "No Complementary Speedup" Theorem holds; that is,
$$\Sigma_k\mathsf{TIME}[n^\ell] \nsubseteq \Pi_k\mathsf{TIME}[n^{\ell-\varepsilon}]/n \tag{4.7}$$
for all $k \geq 1$, $\ell > 1$ and $\varepsilon > 0$. Tourlakis [Tou01] describes a diagonalization method, attributed to Rackoff, that proves the case $k = 1$, *i.e.* $\mathsf{NTIME}[n^\ell] \nsubseteq \mathsf{coNTIME}[n^{\ell-\varepsilon}]/n$. The diagonalizing $\mathsf{NTIME}[n^\ell]$ machine M has the following behavior: on input x, it nondeterministically simulates $N_{|x|}(x, x)$ and complements its outcome, where N_i is the ith co-nondeterministic algorithm that runs in $n^{\ell-\varepsilon}$ time and takes n bits of advice.

  Suppose N' is a $\mathsf{coNTIME}[n^{\ell-\varepsilon}]$ machine that takes n bits of advice and solves the same problem as M. Let i be such that $L(N') = L(N_i)$, and let A be the advice function for N'. Note that $|A(|x|)| = |x|$, by assumption, so $|A(i)| = i$. Therefore
$$\begin{array}{rll} & M(A(i), A(i)) & \text{accepts (as a nondeterministic algorithm)} \\ \iff & N_{|A(i)|}(A(i), A(i)) & \text{rejects (as a co-nondeterministic algorithm)} \\ \iff & N_i(A(i), A(i)) & \text{rejects.} \end{array}$$
  Therefore $L(M) \neq L(N_i) = L(N')$, a contradiction. Our observation is simply that the above argument works irrespective of the number of quantifiers in the alternating algorithm– it works equally well for $\Sigma_k\mathsf{TIME}[n^\ell]$ algorithms versus $\Pi_k\mathsf{TIME}[n^{\ell-\varepsilon}]/n$ algorithms.

- Secondly, if $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]/n^{o(1)}$, then $\mathsf{NTIME}[n^\ell] \subseteq \mathsf{DTISP}[n^{c\ell}, n^{o(1)}]/n^{o(1)}$, by a padding argument.

- Thirdly, we claim that if $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]/n^{o(1)}$, then it follows that $\Sigma_k\mathsf{TIME}[n^\ell] \subseteq \Sigma_{k-1}\mathsf{TIME}[n^{c\ell}]/n^{o(1)}$, for any $\ell \geq 1$. This is effectively a non-uniform version of the Slowdown Lemma, and it has an analogous proof to that lemma.

- Fourthly, we have $\mathsf{DTISP}[n^k, n^{o(1)}]/n^{o(1)} \subseteq \Sigma_k\mathsf{TIME}[n^{1+o(1)}]/n^{o(1)} \cap \Pi_k\mathsf{TIME}[n^{1+o(1)}]/n^{o(1)}$, which is a non-uniform version of Fortnow and Van Melkebeek's simulation with a practically identical proof. (The simulating alternating machine simply uses the same advice function that the deterministic machine used.)

With the four above ingredients, one can derive non-uniform analogues for all of our lower bounds. For instance, when $\mathsf{NTIME}[n] \subseteq \mathsf{DTISP}[n^c, n^{o(1)}]/n^{o(1)}$ the third and fourth points above can be used to infer for any $k \geq 1$ that

$$\Sigma_2\mathsf{TIME}[n^k] \subseteq \mathsf{NTIME}[n^{ck}]/n^{o(1)} \subseteq \mathsf{DTS}[n^{kc^2}]/n^{o(1)} \subseteq \Pi_2\mathsf{TIME}[n^{kc^2/2}]/n^{o(1)},$$

so for $c < \sqrt{2}$ this is a contradiction to (4.7). □

4.4.2 A Generalization to Lower Bounds for Quantified Boolean Formulas

As first observed by Fortnow and Van Melkebeek, the alternation-trading scheme for proving lower bounds against nondeterminism extends naturally to lower bounds against alternating computations. Since alternating polynomial time is equal to polynomial space ($\mathsf{AP} = \mathsf{PSPACE}$ [CKS81]), we know that $\mathsf{ATIME}[n] \not\subseteq \mathsf{DTS}[n^k, n^{o(1)}]$ for *every* $k \geq 1$, otherwise $\mathsf{SPACE}[n] \subseteq \mathsf{ATIME}[n^2] \subseteq \mathsf{SPACE}[n^{o(1)}]$ by Savitch's theorem [Sav70], contradicting the space hierarchy theorem. So we already have a polynomial time lower bound for the general quantified Boolean formula problem (QBF) in our time and space bounded setting.

The more interesting question is: how large can the lower bounds be when we restrict ourselves to solving quantified Boolean formulas where the number of quantifier blocks is a fixed constant? Recall that Σ_k-SAT is the problem of solving a QBF with k quantifier blocks (*i.e.* deciding the truth of Σ_k sentences in first-order Boolean logic). Building on Fortnow and Van Melkebeek [FvM00] who showed that Σ_k-SAT requires $\Omega(n^{k-\varepsilon})$ time on $n^{o(1)}$-space machines, we prove time lower bounds for Σ_k-SAT of the form $\Omega(n^{k+1-\varepsilon_k})$ on the same model, where $\varepsilon_k \in (0,1)$ is a small constant that approaches 0 as k approaches infinity. Our results follow a similar pattern to the $n^{2\cos(\pi/7)}$ lower bound for SAT: we first argue that Σ_k-SAT is "robustly complete" in the appropriate sense, then prove that $\Sigma_k\mathsf{TIME}[n] \not\subseteq \mathsf{DTS}[n^c]$ for appropriate constants $c > 1$, by interleaving a conditional speedup theorem with a "Fortnow-Van Melkebeek-like" argument. Let us recall the result of Fortnow *et al.*, mentioned in Chapter 2:

Theorem 4.4.3 (Fortnow-Lipton-Van Melkebeek-Viglas [FLvMV05]) *For all $k \geq 1$, Σ_k-SAT is robustly complete for $\Sigma_k\mathsf{QL}$.*

Our generalized statement of time lower bounds for quantified Boolean formulas is:

Theorem 4.4.4 *For all $k \geq 1$, Σ_k-SAT requires $\Omega(n^c)$ time on $n^{o(1)}$ space RAMs, where $c^3/k - c^2 - 2c + k < 0$.*

The remainder of this section proves Theorem 4.4.4. As our argument follows the style of other proofs, we keep the proof exposition at a more informal level. The main tool we need is a generalization of our Conditional Speedup Theorem (Theorem 4.2.1) that works for assumptions of the form "$\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^c]$".

Theorem 4.4.5 (Conditional Speedup for the Polynomial Hierarchy) *If the containment $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^c]$ holds for some $c > k$, then for all d satisfying $c \le d < \frac{c}{c-k}$,*

$$\mathsf{DTS}[n^d] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_{k+1}\mathsf{TIME}[n^{1+o(1)}].$$

Note the above generalizes the original Conditional Speedup Theorem (Theorem 4.2.1), as it shows $\Sigma_1\mathsf{TIME}[n] = \mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ implies

$$\mathsf{DTS}[n^{c/(c-1)-\varepsilon}] \subseteq \Sigma_2\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_2\mathsf{TIME}[n^{1+o(1)}].$$

Proof. Similar to the proof of the Conditional Speedup Theorem. We show that if $\mathsf{DTS}[n^d] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_{k+1}\mathsf{TIME}[n^{1+o(1)}]$ then $\mathsf{DTS}[n^{1+dk/c}] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_{k+1}\mathsf{TIME}[n^{1+o(1)}]$ as well. This process converges when $d = 1 + dk/c$, or $d = c/(c-k)$.

We start by invoking the Speedup Lemma (Lemma 3.3.2) to show that

$$\mathsf{DTS}[n^{1+dk/c}] \subseteq (\exists\ n)(\forall\ \log n)\mathsf{DTS}[n^{dk/c}, n^{o(1)}].$$

Applying the speedup theorem for k more times, we obtain

$$\mathsf{DTS}[n^{1+dk/c}] \subseteq (\exists\ n)(\forall\ \log n)\underbrace{(\forall\ n^{d/c})\cdots(Q\ n^{d/c})}_{k-1}(\neg Q\ \log n)\mathsf{DTS}[n^{d/c}]$$

for some $Q \in \{\exists, \forall\}$, where $\neg Q$ is opposite to Q. Since $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^c]$,

$$(\exists\ n)\underbrace{(\forall\ n^{d/c})\cdots(Q\ n^{d/c})(\neg Q\ \log n)}_{k}\mathsf{DTS}[n^{d/c}] \subseteq (\exists\ n)\mathsf{DTS}[n^d].$$

Finally, since $\mathsf{DTS}[n^d] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_{k+1}\mathsf{TIME}[n^{1+o(1)}]$,

$$(\exists\ n)\mathsf{DTS}[n^d] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{1+o(1)}].$$

An analogous argument shows that $\mathsf{DTS}[n^{1+dk/c}] \subseteq \Pi_{k+1}\mathsf{TIME}[n^{1+o(1)}]$. $\square$

Theorem 4.4.6 *If $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^c]$, then for all $\ell \ge 1$ and d satisfying $c \le d < c/(c-k)$,*

$$\mathsf{DTS}[n^{d+\sum_{i=1}^{\ell}\left(\frac{c^2}{dk}\right)^i}] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{\left(\frac{c^2}{dk}\right)^\ell+o(1)}] \cap \Pi_{k+1}\mathsf{TIME}[n^{\left(\frac{c^2}{dk}\right)^\ell+o(1)}].$$

Proof. By induction on ℓ. The case $\ell = 0$ is immediate, by the previous theorem. For the inductive step, suppose $\mathsf{DTS}[n^{d+\sum_{i=1}^{\ell}\left(\frac{c^2}{dk}\right)^i}] \subseteq \Sigma_{k+1}\mathsf{TIME}[n^{\left(\frac{c^2}{dk}\right)^\ell+o(1)}]$. First, the Speedup Lemma implies

$$\mathsf{DTS}[n^{d+\sum_{i=1}^{\ell+1}\left(\frac{c^2}{dk}\right)^i}] \subseteq (\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})(\forall\ \log n)\mathsf{DTS}[n^{d+\sum_{i=1}^{\ell}\left(\frac{c^2}{dk}\right)^i}],$$

where the input to the DTS part has length $n + 2n^{o(1)}$. By the induction hypothesis, the above is contained in

$$(\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})(\forall\ \log n)\Pi_{k+1}\mathsf{TIME}[n^{\left(\frac{c^2}{dk}\right)^{\ell}+o(1)}].$$

Applying $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^c]$ to the Σ_k part of the $\Pi_{k+1}\mathsf{TIME}$ class, the above lies in

$$(\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})(\forall\ \log n)(\forall\ n^{\left(\frac{c^2}{dk}\right)^{\ell}})\mathsf{DTS}[n^{c\left(\frac{c^2}{dk}\right)^{\ell}}].$$

If $c < k$, we already have a contradiction, because $\Sigma_k\mathsf{TIME}[n^k] \subseteq \mathsf{DTS}[n^{kc}] \subseteq \Pi_k\mathsf{TIME}[n^c]$ by the kth root speedup of DTS in Σ_k (Corollary 3.3.1 of the Speedup Lemma).

If $c \geq k$, the kth root speedup of DTS in Σ_k can be applied to show that the above class is contained in

$$(\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})(\forall\ \log n)(\forall\ n^{\left(\frac{c^2}{dk}\right)^{\ell}})\Pi_k\mathsf{TIME}[n^{\frac{c}{k}\cdot\left(\frac{c^2}{dk}\right)^{\ell}+o(1)}] = (\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})\Pi_k\mathsf{TIME}[n^{\frac{c}{k}\cdot\left(\frac{c^2}{dk}\right)^{\ell}+o(1)}].$$

Note $(\frac{c^2}{dk})^{\ell+1} \leq \frac{c}{k}\cdot(\frac{c^2}{dk})^{\ell}$, because $d \geq c$. Applying the assumption $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTS}[n^c]$ again results in the class

$$(\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})\mathsf{DTS}[n^{\frac{c^2}{k}\cdot\left(\frac{c^2}{dk}\right)^{\ell}}].$$

Finally, since $d(\frac{c^2}{dk})^{\ell+1} = \frac{c^2}{k}\cdot(\frac{c^2}{dk})^{\ell}$, the Conditional Speedup Theorem for Σ_k (Theorem 4.4.5) applies, and the above class is in

$$(\exists\ n^{\left(\frac{c^2}{dk}\right)^{\ell+1}})\Sigma_{k+1}\mathsf{TIME}[n^{\frac{c^2}{dk}\cdot\left(\frac{c^2}{dk}\right)^{\ell}+o(1)}] = \Sigma_{k+1}\mathsf{TIME}[n^{\left(\frac{c^2}{dk}\right)^{\ell+1}+o(1)}].$$

An analogous argument proves the containment for $\Pi_{k+1}\mathsf{TIME}[n^{\left(\frac{c^2}{dk}\right)^{\ell+1}+o(1)}]$. □

Set $K_\ell = d + \sum_{i=1}^{\ell}\left(\frac{c^2}{dk}\right)^i$, for $\ell \geq 1$. We claim (the proof is straightforward) that

$$\left(\frac{c^2}{dk}\right)^{\ell} \leq K_\ell\left(1 - \frac{dk}{c^2} - \varepsilon_\ell\right),$$

for a small constant $\varepsilon_\ell > 0$ satisfying $\lim_{\ell\to\infty}\varepsilon_\ell = 0$. We deduce the chain:

$$\begin{aligned}\Sigma_{k+1}\mathsf{TIME}[n^{K_\ell}] &\subseteq (\exists\ n^{K_\ell})\mathsf{DTS}[n^{cK_\ell}]\\ &\subseteq (\exists\ n^{K_\ell})\Sigma_k\mathsf{TIME}[n^{(c/k)K_\ell}] \subseteq \mathsf{DTS}[n^{(c^2/k)K_\ell}] \subseteq \Pi_{k+1}\mathsf{TIME}[n^{(c^2/k)K_\ell(1-\frac{dk}{c^2}-\varepsilon_\ell)}].\end{aligned}$$

For sufficiently large K_ℓ, a contradiction with the "No Complementary Speedup" Theorem is reached when $\frac{c^2}{k}(1 - \frac{dk}{c^2}) < 1$. Recalling that $d < c/(c-k)$, the condition simplifies to $p_k(c) = c^3/k - c^2 - 2c + k < 0$.

Notice when $k = 1$ we obtain precisely the lower bound exponent for SAT. We numerically determined the lower bound exponent for larger values of k; the results are in Table 4.1. As the evidence suggests, at least one root of the polynomial p_k gradually approaches $k+1$ as k increases unboundedly; therefore the lower bound exponent for Σ_k-SAT approaches $k+1$.

Proposition 4.4.1 $\lim_{k\to\infty} p_k(k+1) = 0$. *In particular, for all* k, $p_k(k+1-1/k) < 0$ *and* $p_k(k+1) > 0$.

Problem	Time Lower Bound Exponent
SAT	$n^{1.801}$
Σ_2-SAT	$n^{2.903}$
Σ_3-SAT	$n^{3.942}$
Σ_4-SAT	$n^{3.962}$
Σ_{10}-SAT	$n^{100.991}$
Σ_{100}-SAT	$n^{100.999902}$

Table 4.1: **Time lower bounds for Σ_k-SAT on Small Space RAMs.**

Proof. Algebraic manipulation gives

$$p_k(k+1) = 1/k > 0$$

and

$$p_k(k+1-1/k) = 3/k^3 - 1 - 1/k - 1/k^2 - 1/k^4 < -1/k^4 < 0,$$

for all $k \geq 1$. $\square$

4.4.3 On Further Improvements to SAT Lower Bounds

How far can the alternating-trading scheme be pushed? Can we achieve a *quadratic time* lower bound for SAT on random access machines that use $n^{o(1)}$ space? As demonstrated above, there are many counterintuitive ways to "speed up" a deterministic time computation when one assumes the existence of a fast SAT algorithm. In the next chapter we address the apparent abundance of possible arguments, and show how one can systematically search for new lower bounds via computer. For now, it is interesting to note that one component of the lower bound arguments is actually *optimal* already, at least in some cases. Recall the speedup simulation of Lemma 3.3.2, which states that for all time bounds $t(n) \geq n$, $\mathsf{DTS}[t(n)] \subseteq \Sigma_k\mathsf{TIME}[t(n)^{1/k}]$. The following shows that this theorem is optimal, in the sense that deterministic linear time cannot be sped up any faster with k alternations.

Theorem 4.4.7 *For all $\varepsilon > 0$,*

$$\mathsf{DTS}[n] \not\subseteq \Sigma_k\mathsf{TIME}[n^{1/k-\varepsilon}].$$

Recall the Conditional Speedup Theorem says that if $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ for $c < 2$, then for all $\varepsilon > 0$ we have $\mathsf{DTS}[n^{c/(c-1)-\varepsilon}] \subseteq \Sigma_2\mathsf{TIME}[n^{1+o(1)}] \cap \Pi_2\mathsf{TIME}[n^{1+o(1)}]$. However, if $c < 2$ then there is an $\varepsilon > 0$ such that $c/(c-1) - \varepsilon > 2$. Thus, if SAT had a subquadratic algorithm, then we *could* speed up superquadratic DTS algorithms to Σ_2 linear time. We consider this to be an unlikely possibility, given Theorem 4.4.7. However, it is beyond our capabilities at the present time to prove that such a speedup is not possible.

Proof of Theorem 4.4.7. Consider the decision problem PARITY, of computing whether the parity of n bits is odd. Clearly PARITY is in $\mathsf{DTS}[n]$. We show using a standard construction that if PARITY is in $\Sigma_k\mathsf{TIME}[n^{1/k-\varepsilon}]$, then the problem can be solved with depth $k+1$ circuits of $2^{O(n^{1/k-\varepsilon})}$ size, contradicting Håstad's [Has86] celebrated circuit lower bound for PARITY.

Consider a $\Sigma_k\mathsf{TIME}[n^{1/k-\varepsilon}]$ machine M. Without loss of generality, suppose M guesses $n^{1/k-\varepsilon}$ bits in each alternation. Let M' be the "deterministic part" of M, *i.e.* a deterministic random access machine that runs in $O(n^{1/k-\varepsilon})$ time on k auxiliary $n^{1/k-\varepsilon}$-bit strings in addition to the input.

For each integer n, if k is odd (repectively, even), let D_n be a $2^{O(kn^{1/k-\varepsilon})}$-size DNF (respectively, CNF) formula that is equivalent to M' on all n bit inputs with k auxiliary inputs of $n^{1/k-\varepsilon}$. Make a tree-like circuit C_n of depth k, with an OR at the output gate (and alternating AND/ORs at each depth), and precisely $2^{n^{1/k-\varepsilon}}$ fan-in at every gate that is not a leaf. (Note that if k is odd, the bottom level of gates are ORs; otherwise, the bottom level are ANDs.) Label each of the $2^{n^{1/k-\varepsilon}}$ wires fanning into a gate with a unique $n^{1/k-\varepsilon}$ bit string. Label each leaf of the tree with the k labels on the wires that are traversed on the path from the output gate to that leaf. Now replace each leaf with a copy of D_n, where the k auxiliary inputs to a copy of D_n are the label of that leaf. Finally, feed the n bit input for the circuit C_n, to the input of each copy of D_n.

We claim that the circuit C_n simulates M exactly on all n-bit inputs. Moreover, the depth of C_n is $k+1$, and its size is $2^{O(kn^{1/k-\varepsilon})}$. Thus, PARITY has depth-$(k+1)$ circuits of $2^{O(kn^{1/k-\varepsilon})}$ size, a contradiction. $\square$

The above proof also shows that if one can show an exponential lower bound for depth-three circuits that solve problems in $\mathsf{NTIME}[n]$, then our desired quadratic time lower bound would be immediate.

Corollary 4.4.3 *Suppose for every $\varepsilon > 0$ there is a problem $L_\varepsilon \in \mathsf{NTIME}[n]$ that cannot be recognized by depth-three circuits of size $2^{O(n^{1-\varepsilon})}$). Then $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^{2-\varepsilon}]$ for all $\varepsilon > 0$.*

Proof. Prove the contrapositive: if $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^{2-\varepsilon}]$, then using the proof of Theorem 4.4.7 one can construct *uniform* depth-three circuits of size $2^{O(n^{1-\varepsilon})}$ for every problem in $\mathsf{NTIME}[n]$. $\square$

Unfortunately, it is only known how to prove exponential lower bounds for depth-three circuits in restricted cases, *e.g.* when the fan-in of gates at the lowest level is bounded by a small function of n such as $o(\log\log n)$ or $o(\log n)$ (cf. [IPZ01, CIP06]). In order for these lower bounds to be useful for us, this fan-in would need to be at least $n^{1-\varepsilon}$.

In the next chapter, we take a unifying approach to all lower bound proofs that follow the alternation-trading scheme, and formalize the task of proving better lower bounds. This formalization leads us into the realm of automated theorem proving, in which we find hard evidence that the $\Omega(n^{2\cos(\pi/7)})$ time lower bound is actually the best possible that can be achieved with the current tools.

4.5 Chapter Summary

We proved a series of increasing time lower bounds for solving many natural NP-complete problems on space-bounded random access machines, starting with an $\Omega(n^{1.6616})$ lower bound and culminating in an $\Omega(n^{2\cos(\pi/7)-\varepsilon})$ lower bound. The method extends naturally to lower bounds for the same problems on non-uniform RAMs with program size $n^{o(1)}$. Larger lower bounds are possible for complete problems in higher levels of the polynomial hierarchy, such as the $\mathbf{\Sigma}_k$-SAT problem for $k \geq 2$.

Chapter 5

Automated Search For Time Lower Bounds

Recall that a large class of time lower bounds follow an alternation-trading scheme:

1. Assume *e.g.* that SAT can be solved in $n^{1.8}$ time and $n^{o(1)}$ space.
2. Define a notion of *speedup*, or "trading time for alternations", in which a deterministic time t class is shown to be contained in an alternating time $o(t)$ class, and the function in the $o(\cdot)$ depends on the model and the number of alternations used.
3. Define a notion of *slowdown*, or "trading alternations for time", in which some alternations from an alternating time t class can be "removed", at a low time cost. This item uses the assumption that one is trying to contradict, *e.g.* that SAT is in $n^{1.8}$ time and $n^{o(1)}$ space.
4. Use some combination of (1) and (2) to show that $C[t]$ is contained in $C[t^{1-\varepsilon}]$, for some $\varepsilon > 0$ and some complexity class $C[t]$ parameterized by a time bound t.

While the overall strategy behind these lower bounds is certainly principled, the proofs themselves have an "ad-hoc" feel to them. Speedups and slowdowns are applied in carefully contrived ways, and one gets a sense that the space of all possible proofs would be rather difficult (if not impossible) to systematically explore. In this chapter, we show how to turn the informal alternation-trading scheme into a sound formal proof system, so that the search for new lower bounds becomes a feasible problem that computers themselves can help us attack. Informally speaking, the "hard work" in the proofs can be replaced by a computer search that solves a series of linear programming problems, each solvable in polynomial time [Kha79, Kar84]. Our formal proof system captures every previous lower bound proof of this kind, including our own.

The key to our approach is that we separate the *discrete choices* in an alternation-trading proof from the continuous, *real-valued choices*. Discrete choices consist of choosing whether to apply a speedup or slowdown at a given step, and which complexity class $\mathcal{C}$ to try to use in the contradiction. We show how to restrict the number of possible discrete choices that need to be made by writing proofs in a certain *normal form*. For example, one consequence of the normal form is that it suffices to consider just $\mathcal{C} = \mathsf{DTS}[t]$. The real-valued choices come from the speedup step: one must guess how to split up the runtime of a DTS class using quantifiers, and there are infinitely many choices one could make for this split. However, once all the discrete choices are made, it turns out that with some care we can formulate the remaining real-valued problem as an instance of linear programming, which can then be efficiently solved.

Unfortunately, we cannot efficiently search the space of *all* possible proofs. The number of discrete choices increases exponentially with the number of lines in the proof, although the exponential increase is not horrendous (it is $O(2^n/n^{3/2})$ for n-line proofs, with small hidden constants). With our normal form simplifications and linear programming reduction, we can search a rather substantial portion of the proof space. For example, it is a feasible task to search over *all* proofs of up to 24 lines for the best possible lower bounds. From those results, we can restrict the discrete search space much further and work over a space of much longer proofs using a heuristic search.

Having done so, we report a rather surprising conclusion: that the $\Omega(n^{2\cos(\pi/7)})$ lower bound from the previous chapter appears to be *optimal* among alternation-trading proofs. We have found no proof that is better than $n^{1.8019}$, and nothing to suggest that there might be a better overall strategy than that taken by the $n^{1.8019}$ bound. As the number of lines increases, a clear pattern emerges: any short proof that mimics our $\Omega(n^{2\cos(\pi/7)})$ proof achieves a very good lower bound with respect to others of similar length, and as the proof length increases, more and more of these 1.801-like proofs achieve roughly the same bound, numerically. That is, 1.801-like proofs appear to dominate the proofs that are not 1.801-like, but no 1.801-like proof completely dominates over the others that are 1.801-like; we witness what seems to be a "convergence" of all the best proofs to a single uniform type: the $\Omega(n^{2\cos(\pi/7)})$ proof.

5.1 Formalization of time-space lower bounds

Let us first formalize the style of proofs that have been under consideration. Every lower bound proof applied a sequence of "speedup theorems" and "slowdowns" in some order. (There is also the "extra" step of combining like quantifiers, *e.g.* $(\exists\ n)(\exists\ n)\mathsf{DTS}[n] \subseteq (\exists\ n)\mathsf{DTS}[n]$, but that can be integrated into the speedup step.) To do this properly, we need to introduce new notation. We write an alternating complexity class in the form

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}]$$

to mean that the input to the ith quantifier block is of length $O(n^{b_i})$, and the input to the DTS computation has length $O(n^{b_{k+1}})$. (Note the first quantifier always has input of length n.) We also assume without loss of generality that adjacent quantifiers are of opposite types, *e.g.* if Q_1 is $\exists$ then Q_2 is $\forall$. In our formalization it shall be crucial to keep track of the input lengths to quantifiers, since the golden ratio and $2\cos(\pi/7)$ time lower bounds rely on the nice properties of input sizes granted by the Speedup Lemma (Lemma 3.3.2).

Definition 5.1.1 *Let $c > 1$. An* alternation-trading proof *for c is a list of complexity classes of the form*

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}],$$

where k is a non-negative integer, and $a_i > 0$, $b_i \geq 1$ for all i. (When $k = 0$, the class is deterministic.) The items of the list are called lines of the proof*. Each line is obtained from the previous line by applying either a* speedup rule *or a* slowdown rule*. More precisely, if the ith line is*

$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}],$$

then the $(i+1)$st line has one of three possible forms:

1. *(Speedup Rule)*
$$(Q_k\ n^{x})^{\max\{x,1\}}(Q_{k+1}\ n)^{1}\mathsf{DTS}[n^{a_{k+1}-x}],$$
*for some $x \in (0, a_{k+1})$, provided that $k = 0$ (*i.e.* the ith line is a deterministic class).*

2. *(Speedup Rule)*
$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{\max\{a_k,x\}})^{\max\{x,b_{k+1}\}}(Q_{k+1}\ n)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}-x}],$$
for some $x \in (0, a_{k+1})$, provided that $k > 0$.

3. *(Slowdown Rule)*
$$(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_{k-1}}(Q_{k-1}\ n^{a_{k-1}})^{b_k}\mathsf{DTS}[n^{c\cdot\max\{a_{k+1},a_k,b_k\}}].$$

We say that an alternation-trading proof shows $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c] \implies A_1 \subseteq A_2$ *if its first line is A_1 and its last line is A_2.*

The first two rules are just a syntactic formulation of the Speedup Lemma, where the DTS part of the sped-up computation only reads two guessed configurations– so the input it reads is different from the input read by the innermost quantifier. In particular, the second rule follows since

$$\begin{aligned}&(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}]\\ \subseteq\ &(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{a_k})^{b_{k+1}}(Q_k\ n^{x})^{\max\{b_{k+1},x\}}(Q_{k+1}\ n)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}]\\ \subseteq\ &(Q_1\ n^{a_1})^{b_2}(Q_2\ n^{a_2})\cdots^{b_k}(Q_k\ n^{\max\{a_k,x\}})^{\max\{b_{k+1},x\}}(Q_{k+1}\ n)^{b_{k+1}}\mathsf{DTS}[n^{a_{k+1}}].\end{aligned}$$

Note that for $k = 0$, the first Speedup Rule adds two quantifiers to the line, and for $k > 0$, the second Speedup Rule adds one new quantifier to the line. It is vital to observe that each application of a Speedup Rule introduces a new real-valued parameter x that must be determined.

The third rule is a syntactic formulation of the Slowdown Lemma, where we assume that $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$, and use it to remove a quantifier.

The justification for the definition of alternation-trading proof comes directly from the proofs of the Speedup Lemma and the Slowdown Lemma. Due to these, alternation-trading proofs are *sound*, in that an alternation-trading proof of $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c] \implies A_1 \subseteq A_2$ implies the *truth* of that implication.

5.1.1 A normal form for alternation-trading proofs

All lower bound proofs following the alternation-trading scheme effectively use the assumption $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ to derive a contradiction to a time hierarchy theorem, namely the "No Complementary Speedup" Theorem that $\Sigma_k\mathsf{TIME}[t(n)] \nsubseteq \Pi_k\mathsf{TIME}[t(n)^{1-\varepsilon}]$ for some k, $t(n) \geq n$, and $\varepsilon > 0$. It turns out to be sufficient to derive $\Sigma_k\mathsf{TIME}[t(n)] \subseteq \Pi_k\mathsf{TIME}[t(n)]$ and obtain an absurdity from that. To consider all possible ways to derive a time hierarchy contradiction (at least, between alternating, nondeterministic, and deterministic classes), we consider the most general setting of deriving a contradiction from *complementary alternating classes*.

Definition 5.1.2 *We say that A_1 and A_2 are* complementary alternating classes *if A_1 is the complement of A_2.*

Every known time lower bound following the alternation-trading scheme proves that $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ implies $A_1 \subseteq A_2$, for some complementary alternating classes A_1 and A_2.

We now introduce a restriction to the space of alternation-trading proofs we consider, calling it a "normal form." Our intent is to show that any lower bound provable with complementary alternating classes can also be proved with a normal form proof, so it suffices for us to explore normal form proofs.

Definition 5.1.3 *Let $c \geq 1$. An alternation-trading proof for c is in* normal form *if*

- *The first and last lines are $\mathsf{DTS}[n^a]$ and $\mathsf{DTS}[n^{a'}]$ respectively, for some $a \geq a'$.*
- *No other lines are DTS classes.*

First we show that a normal form proof for c implies that $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$.

Lemma 5.1.1 *Let $c \geq 1$. If there is an alternation-trading proof for c in normal form having at least two lines, then $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$.*

Proof. Let P be an alternation-trading proof for c in normal form. We consider two cases.

- Suppose $a > a'$. In this case, $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ implies $\mathsf{DTS}[n^a] \subseteq \mathsf{DTS}[n^{a-\delta}]$ for some $\delta > 0$. By translation, $\mathsf{DTS}[n^a] \subseteq \mathsf{DTS}[n^{a-\delta}]$ implies

$$\mathsf{DTS}[n^{a^2/(a-\delta}] \subseteq \mathsf{DTS}[n^a] \subseteq \mathsf{DTS}[n^{a-\delta}],$$

and $\mathsf{DTS}[n^{a\cdot(a/(a-\delta))^i}] \subseteq \mathsf{DTS}[n^{a-\delta}]$ for all $i \geq 0$. Since $\delta > 0$, this implies $\mathsf{DTS}[n^L] \subseteq \mathsf{DTS}[n^{a-\delta}]$ for all $L \geq a - \delta$. Therefore, if $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ then for all $L \geq a$,

$$\mathsf{NTIME}[n^L] \subseteq \mathsf{DTS}[n^{Lc}] \subseteq \mathsf{DTS}[n^{a-\delta}] \subseteq \mathsf{coNTIME}[n^{a-\delta}],$$

a contradiction to the "No Complementary Speedup" theorem.

- Suppose $a = a'$. Let A be a line in P with a positive number of alternations. (Such a line must exist since P has at least two lines.) The proof P shows that $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c]$ implies $\mathsf{DTS}[n^a] \subseteq A \subseteq \mathsf{DTS}[n^{a'}]$, so $A = \mathsf{DTS}[n^a]$.

  Since $\mathsf{DTS}[n^a]$ is closed under complement,

$$A = A', \tag{5.1}$$

  where A' is the complement of A. Without loss of generality, assume $A = (\exists n^\delta)B$ and $A' = (\forall n^\delta)B'$ for some $\delta > 0$ and complementary classes B and B'. It is easy to see that

$$A' = (\forall n^\delta)A' \text{ and } A = (\exists n^\delta)A. \tag{5.2}$$

  Now consider the class $\mathsf{DTS}[n^{\delta\lceil\frac{k}{\delta}\rceil}] \supseteq \mathsf{DTS}[n^k]$, for arbitrary $k \geq 1$. By the Speedup Lemma (Lemma 3.3.2) and the fact that $\mathsf{DTS}[n^\varepsilon] \subseteq A'$ for some $\varepsilon > 0$,

$$\mathsf{DTS}[n^k] \subseteq \mathsf{DTS}[n^{\delta\lceil\frac{k}{\delta}\rceil}] \subseteq \underbrace{(\exists\ n^\delta)(\forall\ n^\delta)\cdots(\exists\ n^\delta)(\forall\ n^\delta)}_{\lceil k/\delta\rceil} A'.$$

  Applying equations (5.1) and (5.2), we have

$$\begin{gathered}
(\exists\ n^\delta)(\forall\ n^\delta)\cdots(\exists\ n^\delta)(\forall\ n^\delta)A' \\
= (\exists\ n^\delta)(\forall\ n^\delta)\cdots(\exists\ n^\delta)A' \\
= (\exists\ n^\delta)(\forall\ n^\delta)\cdots(\exists\ n^\delta)A \\
= (\exists\ n^\delta)(\forall\ n^\delta)\cdots A \\
= \cdots = (\exists\ n^\delta)(\forall\ n^\delta)A' = (\exists\ n^\delta)A' = (\exists\ n^\delta)A = A.
\end{gathered}$$

  Therefore $\mathsf{DTS}[n^k] \subseteq A$, for *every* $k \geq 1$. Hence $\mathsf{NP} \subseteq \mathsf{DTS}[n^{O(1)}, n^{o(1)}] \subseteq A$. But by applying a slowdown step for a finite number of times to A, there is an alternation-trading proof that $A \subseteq \mathsf{DTS}[n^K]$ for a constant K. It follows that $\mathsf{NP} \subseteq A \subseteq \mathsf{DTS}[n^K] \subseteq \mathsf{coNTIME}[n^K]$, contradicting the "No Complementary Speedup Theorem." So $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$ in this case as well.

$\square$

We now prove that any alternation-trading proof showing $\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c] \implies A_1 \subseteq A_2$ for complementary alternating classes A_1 and A_2 can be converted into an analogous normal form proof. As mentioned earlier, every time-space lower bound known in this area falls in this category. Therefore, to find good lower bounds that build on the existing tools, it suffices for us to search for good normal form proofs.

Theorem 5.1.1 *Let A_1 and A_2 be complementary. If there is an alternation-trading proof P for c that shows ($\mathsf{NTIME}[n] \subseteq \mathsf{DTS}[n^c] \implies A_1 \subseteq A_2$), then there is a normal form proof for c, of length at most that of P.*

Proof. Consider an alternation-trading proof P for c, written as

$$P = A_1, C_1, \ldots, C_k, A_2.$$

Define the *dual proof P'* by

$$P' = A_2, \neg C_1, \ldots, \neg C_k, A_1,$$

where the notation $\neg C$ denotes the unique complementary alternating class for C, *i.e.* every '$\forall$' in C is replaced with '$\exists$', and vice-versa. Note that P' is an alternation-trading proof if and only if P is one.

Since the quantifiers of the first and last line of P are different, there must be a line $C_i = \mathsf{DTS}[n^a]$ for some a.

- Suppose there is only one deterministic class in P; call it C_i. Then

  $$P'' = C_i, C_{i+1}, \ldots C_k, A_2, \neg C_1, \ldots, \neg C_i$$

  is also an alternation-trading proof, obtained by piecing together the appropriate lines from P and P'. However, $C_i = \neg C_i$, since $\mathsf{DTS}[n^a]$ is closed under complement. Hence P'' is in normal form: its first and last lines are DTS classes, and no intermediate class is a DTS class.

- Suppose there are $k \geq 2$ different DTS classes in P. Write P as:

  $$P = A_1, \ldots, \mathsf{DTS}[n^{a_1}], \ldots, \mathsf{DTS}[n^{a_2}], \ldots, \ldots, \mathsf{DTS}[n^{a_k}], \ldots, A_2.$$

  There are two cases:

  - If there is an $i \in [k]$ satisfying $a_i \geq a_{i+1}$, we are done: simply take P'' to be the sequence of lines from $\mathsf{DTS}[n^{a_i}]$ and $\mathsf{DTS}[n^{a_{i+1}}]$ to be the normal form proof.
  - If $a_i < a_{i+1}$ for every i, then set $P'' = \mathsf{DTS}[n^{a_k}], \ldots, A_2, \ldots, \mathsf{DTS}[n^{a_1}]$, where the classes in the first "..." in P'' are taken directly from P, and the classes in the second "..." in P'' are gotten by taking the lines $A_2, \ldots, \mathsf{DTS}[n^{a_1}]$ in P'. P'' is in normal form since $a_k > a_1$.

□

N.B. From here on, we assume that all alternation-trading proofs under discussion are in normal form.

5.1.2 Proof annotations

Different lower bound proofs result in quite different sequences of speedups and slowdowns. To illustrate this, we look at the notion of a *proof annotation*, which makes the differences between the lower bounds cleanly visible.

Definition 5.1.4 *A* proof annotation *for an alternation-trading proof of ℓ lines is the unique $(\ell-1)$-bit vector A that has $A[i] = 1$ (respectively, $A[i] = 0$) if the ith line is obtained by a speedup (respectively, a slowdown), for all $i = 1, \ldots, \ell - 1$.*

By definition, an $(\ell-1)$-bit proof annotation corresponds to a proof strategy for an ℓ-line proof. For a normal form alternation-trading proof with ℓ lines, its proof annotation A must have $A[1]=1$, and $A[\ell-1]=0$. In fact, $A[\ell-2]=0$ as well – otherwise the last line would have at least one quantifier. Thus every proof annotation begins with a 1 and ends with two 0's.

How many possible proof annotations are there? Thanks to the use of normal form, the number of possible annotations is closely related to the number of well-balanced strings over parentheses. For a string $x = x_1 \cdots x_{\ell-1}$ with $x_i \in \Sigma$, we define $x[i..j] := x_i x_{i+1} \cdots x_j$ for $i \leq j$.

Definition 5.1.5 *Let $n > 0$ be an integer and $x \in \{\texttt{(},\ \texttt{)}\}^n$. x is* well-balanced *if*

- *for all $i = 1, \ldots, n-1$, the number of* (*'s in $x[1..i]$ is greater than the number of*) *'s, and*
- *the number of* (*'s in x equals the number of*) *'s in x.*

The well-balanced strings are also called the *Dyck words*. Intuitively, the definition of well-balanced just means that each left parenthesis "matches" with a right parenthesis. Recall that the kth Catalan number is $C(k) = \frac{1}{k+1}\binom{2k}{k}$. A well-known fact in combinatorics states that the number of well-balanced strings of a given length can be counted with the Catalan numbers.

Fact 1 *The number of well-balanced strings of length $2k$ is $C(k)$.*

Proposition 5.1.1 *Let $\ell > 3$ be even. The number of possible annotations for proofs of ℓ lines is $C(\ell/2-1)$.*

Proof. Consider an $(\ell-1)$-bit vector $A = [1, \ldots, 0]$. The first 1 introduces two quantifiers in line 1 of the corresponding proof. All subsequent 1's introduce only one quantifier. All 0's remove one quantifier. So in order for A to count as a proof annotation, it must be that the number of 1's in any prefix of A is greater than (or equal to) the number of 0's, up to the last line, in which the number of 0's becomes the number of 1's plus one. (Recall that in normal form, only the first and last lines of a proof are DTS classes.)

Given these observations, it is not hard to see that every proof annotation can be expressed as a well-balanced string of the form $(x)y$, where x and y are well-balanced strings such that $|x| + |y| = \ell - 4$, corresponding to the bit vector $[1, x', 0, y', 0]$, where x' and y' are the strings gotten by replacing '(' with '1' and ')' with '0' in x and y, respectively. But the number of such well-balanced strings is $C(\ell/2-1)$, for even $\ell > 3$: *every* well-balanced string of length $\ell-2$ can be written in the form $(x)y$, where $|x| + |y| = \ell - 4$, and by the fact, the number of such strings is precisely $C(\ell/2-1)$. $\square$

Corollary 5.1.1 *Let $\ell > 3$ be even. The number of possible annotations for proofs of ℓ lines is $\Theta(2^\ell/\ell^{3/2})$.*

Proof. By the previous proposition, the number is

$$C((\ell/2)-1) = \frac{2}{\ell}\binom{\ell-2}{\ell/2-1} = \Theta\left(\frac{1}{\ell}\cdot\frac{2^\ell}{\ell^{1/2}}\right),$$

where the last equality follows from a standard estimate. □

That is, the number of n-bit proof annotations is only a poly(n)-fraction of the total number of n-bit strings.

5.1.3 Proof annotations for the lower bounds we have seen

By unraveling their arguments, it is straightforward (and enlightening) to discern what proof annotations are used by the lower bounds from the previous two chapters. Note that a proof annotation in itself does not determine the proof entirely– each application of a speedup rule introduces a new real-valued parameter x that must be set to some value. The problem of determining optimal values for these parameters shall be tackled in the next section.

The $n^{\sqrt{2}}$ lower bound (Chapter 3, Section 3.3.1) This proof is gotten by using the only possible three-bit proof annotation: $[1, 0, 0]$.

The other lower bounds are all inductive arguments, corresponding to an infinite sequence of proof annotations of increasing length.

The n^ϕ lower bound of Fortnow-Van Melkebeek (Chapter 3, Section 3.3.2). Their argument starts from DTS, performs a sequence of speedups, then a sequence of slowdowns to reach DTS again. So the corresponding proof annotations are:

$$[1,0,0],\ [1,1,0,0,0],\ [1,1,1,0,0,0,0],\ [1,\ldots,1,0,\ldots,0,0],\ \ldots$$

That is, many speedups are applied, followed by many slowdowns.

The n^ϕ lower bound via the Conditional Speedup Theorem (Chapter 4, Section 4.2). An alternative argument for the same exponent invokes our conditional speedup theorem to put $\mathsf{DTS}[n^{c/(c-1)-\varepsilon}]$ in $\Sigma_2\mathsf{TIME}[n^{1+o(1)}]$, then perform two slowdowns on the Σ_2 class. This argument corresponds to the sequence:

$$[1,0,0],[1,0,1,0,0],[1,0,1,0,1,0,0],\ [1,0,1,0,\ldots,1,0,0],\ldots$$

That is, the proof performs speedups and slowdowns in alternation.

Our $n^{1.6616}$ lower bound (Chapter 4, Section 4.1). Recall this lower bound proves that $\Sigma_2\mathsf{TIME}[n] \subseteq \Pi_2\mathsf{TIME}[n^{c^2/2}]$, then $\Sigma_3\mathsf{TIME}[n] \subseteq \Pi_3\mathsf{TIME}[n^{c^4/6}]$, *etc.* That is, the first annotation is $A_1 = [0, 0, 1]$, the second annotation is $A_2 = [0, 0, 1, 0, 0, 1, 1]$, and the kth annotation is given by

$$A_k = A_{k-1} \bullet\ \cdots\ \bullet A_1 \bullet [0,0,\underbrace{1,\ldots,1}_{k}],$$

where '$\bullet$' represents concatenation. Putting these annotations in normal form (Theorem 5.1.1), the sequence becomes

$$[1,0,0],\ [1,1,\underbrace{0,0,1}_{A_1},0,0],\ [1,1,1,\underbrace{0,0,1,0,0,1,1}_{A_2},\underbrace{0,0,1}_{A_1},0,0],\ \ldots,\ [\underbrace{1,\ldots,1}_{k}]\bullet A_{k-1}\bullet\cdots\bullet A_1\bullet[0,0],\ldots$$

Our $n^{\sqrt{3}}$ lower bound (Chapter 4, Section 4.2). The annotation for the $n^{\sqrt{3}}$ lower bound is obtained by combining the Conditional Speedup annotation and the $n^{1.6616}$ annotation. Now $A_1 = [0,0,1,0,1,\ldots,0,1]$, but A_k is defined just as before. The resulting sequence (in normal form) is

$$[1,0,\ldots,1,0,0],[1,1,\underbrace{0,0,1,\ldots,0,1}_{A_1},0,0],[1,1,1,\underbrace{0,0,1,\ldots,0,1,0,0,1,1}_{A_2},\underbrace{0,0,1,\ldots,0,1}_{A_1},0,0],\ldots$$

Our $n^{1.78}$ lower bound (Chapter 4, Section 4.3). The first annotation here is the same as the $n^{\sqrt{3}}$ lower bound, but the annotation that simulates DTS in Π_3 changes, from merely $[1,1]$ to the annotations

$$S_1 = [1,\underbrace{1,0,1,0,\ldots,1,0,1}_{\text{cond. speedup}},0,0,1,1], \text{ and } S_\ell = [1,\underbrace{1,0,1,0,\ldots,1,0,1}_{\text{cond. speedup}},0,0]\bullet S_{\ell-1}.$$

Now A_k changes to become

$$A_k = A_{k-1}\bullet\ \cdots\ \bullet A_1\bullet[0,0,\underbrace{1,\ldots,1}_{k-2}]\bullet S_\ell,$$

for some sufficiently large ℓ. Substituting S_ℓ into the normal form sequence for $n^{\sqrt{3}}$, the sequence is

$$\begin{aligned}
&[1,0,\ldots,1,0,0],\\
S_\ell\bullet{}&[\underbrace{0,0,1,\ldots,0,1}_{A_1},0,0],\\
[1]\bullet S_\ell\bullet{}&\underbrace{(A_1\bullet[0,0]\bullet S_\ell)}_{A_2}\bullet A_1\bullet[0,0],\\
[1,1]\bullet S_\ell\bullet{}&A_3\bullet A_2\bullet A_1\bullet[0,0],\ldots
\end{aligned}$$

Our $n^{2\cos(\pi/7)}$ lower bound (Chapter 4, Section 4.4). Recall that this lower bound combines the n^{ϕ} argument of Fortnow *et al.* with the n^{ϕ} argument via our Conditional Speedup Theorem. The resulting annotation reflects this directly. Let $A_{1.618} = [1,0,1,0,\ldots,1,0,0]$, where the $\ldots$ contain some finite number of repetitions of $1,0$. This is precisely the sequence from the Conditional Speedup Theorem lower bound. The sequence of proof annotations for our $n^{2\cos(\pi/7)}$ bound is given by:

$$[1, 0, 1, 0, \ldots, 1, 0, 0],$$
$$[1, \underbrace{1, 0, 1, 0, \ldots, 1, 0, 0}_{A_{1.618}}, \underbrace{1, 0, 1, 0, \ldots, 1, 0, 0}_{A_{1.618}}],$$
$$[1, 1, \underbrace{1, 0, 1, 0, \ldots, 1, 0, 0}_{A_{1.618}}, \underbrace{1, 0, 1, 0, \ldots, 1, 0, 0}_{A_{1.618}}, \underbrace{1, 0, 1, 0, \ldots, 1, 0, 0}_{A_{1.618}}], \ \ldots$$

That is, the proof performs many speedups, then a sequence of slowdown-speedup alternations, then two consecutive slowdowns, repeating this until the original sequence of speedups has been cancelled out.

5.2 Translating lower bounding into linear programming

We have defined alternation-trading proofs, given a nice normal form for them, and have shown how the strategies of prior lower bounds can be expressed succinctly in terms of proof annotations. A potential plan for finding new lower bounds is to try searching over all proof annotations to see which is best. However, in order for this plan to be sensible, we need a way to efficiently determine the best proof that can be obtained with a given annotation. We now describe how to take any proof annotation A and constant $c > 1$, and formulate an instance of linear programming that has a feasible solution if and only if there is an alternation-trading proof of $\mathsf{NTIME}[n] \nsubseteq \mathsf{DTS}[n^c]$ that follows the steps of annotation A.

Let A be a proof annotation of ℓ bits and let $c > 1$. Let m be the maximum number of quantifiers in any line of A; note m can be computed in linear time by reading the bits of A. The corresponding linear programming instance has variables $a_{i,j}$, $b_{i,j}$, and x_i, for all $i = 0, \ldots, \ell - 1$ and $j = 1, \ldots, m$.[1] Intuitively,

$a_{i,j}$ is the exponent for the runtime of the jth quantifier block in the class on the ith line,

and

$b_{i,j}$ is the exponent for the input to the jth quantifier block of the class on the ith line.

The variable x_i is only defined if the ith line was obtained by a speedup step, and it represents the (real-valued) choice of an exponent in a quantifier block. For example:

- If the kth line of a proof is $\mathsf{DTS}[n^a]$, the corresponding constraints for that line are

$$a_{k,0} = a, \quad b_{k,1} = 1, \quad (\forall k > 0)\ a_{k,i} = b_{k,i} = 0.$$

- If the kth line of a proof is $(\exists\ n^b)^b \mathsf{DTS}[n^a]$, then

$$a_{k,0} = a, \quad b_{k,1} = b, \quad a_{k,1} = b, \quad b_{k,1} = 1, \quad (\forall k > 1)\ a_{k,i} = b_{k,i} = 0.$$

We now describe the constraints of the linear programming instance.

[1] We start the numbering of lines at 0, so that at the ith line it follows that i rules have been applied.

Initial constraints. For the 0th and $(\ell-1)$th lines, we have the constraints

$$a_{0,1} \geq 1,\ b_{0,1} = 1,\ \ (\forall k > 1)\ a_{0,k} = b_{0,k} = 0, \text{ and}$$

$$a_{\ell,1} \geq 1,\ b_{\ell,1} = 1,\ \ (\forall k > 1)\ a_{\ell,k} = b_{\ell,k} = 0,$$

representing the classes $\mathsf{DTS}[n^{a_{0,1}}]$ and $\mathsf{DTS}[n^{a_{\ell-1,0}}]$, respectively. The constraint $a_{0,1} \geq a_{\ell-1,1}$ is also included, to ensure that the proof is in normal form. The first line of a proof is always an application of the first Speedup Rule, being

$$(Q_1 n^x)^{\max\{x,1\}}(Q_2\ n)^1 \mathsf{DTS}[n^{a-x}].$$

The corresponding LP constraints for the first line are:

$$\begin{gathered} a_{1,1} \geq a_{0,1} - x_1,\ b_{1,1} = 1, \\ a_{1,2} = 1,\ b_{1,2} \geq x_1,\ b_{1,2} \geq 1, \\ b_{1,3} = 1,\ a_{1,3} = x_3 \\ (\forall\ k:\ 4 \leq k \leq m)\ a_{1,k} = b_{1,k} = 0. \end{gathered}$$

Speedup Rule constraints. For the ith line where $i > 1$ and $A[i] = 1$, the constraints are

$$\begin{gathered} a_{i,1} \geq 1,\ a_{i,1} \geq a_{i-1,1} - x_i,\ b_{i,1} = b_{i-1,1} \\ a_{i,2} = 1,\ b_{i,2} \geq x_i,\ b_{i,2} \geq b_{i-1,1}, \\ a_{i,3} \geq a_{i-1,2},\ a_{i,3} \geq x_i,\ b_{i,3} \geq b_{i-1,2} \\ (\forall\ k:\ 4 \leq k \leq m)\ a_{i,k} = a_{i-1,k-1}, b_{1,k} = b_{i-1,k-1}. \end{gathered}$$

These constraints express that

$$\cdots\ {}^{b_2}(Q_2\ n^{a_2})^{b_1} \mathsf{DTS}[n^{a_1}]$$

in the ith line is replaced with

$$\cdots\ {}^{b_2}(Q_2\ n^{\max\{a_2,x\}})^{\max\{x,b_2\}}(Q_1\ n)^{b_1} \mathsf{DTS}[n^{\max\{a_1-x,1\}}]$$

in the $(i+1)$st line, where Q_1 is opposite to Q_2.

Slowdown Rule constraints. For the ith line where $A[i] = 0$, the constraints are

$$\begin{gathered} a_{i,1} \geq c \cdot a_{i-1,1},\ a_{i,1} \geq c \cdot b_{i-1,2},\ a_{i,1} \geq c \cdot a_{i-1,2},\ b_{i,1} = b_{i-1,2} \\ (\forall\ k:\ 2 \leq k \leq m-1)\ a_{i,k} = a_{i-1,k+1},\ b_{i,k} = b_{i-1,k+1} \\ a_{i,m} = b_{i,m} = 0. \end{gathered}$$

These express the replacement of

$$\cdots\ {}^{b_2}(Q_1 n^{a_2})^{b_1} \mathsf{DTS}[n^{a_1}]$$

in the ith line with

$$\cdots\ {}^{b_2}\mathsf{DTS}[n^{c \cdot \max\{a_1,a_2,b_2\}}]$$

in the $(i+1)$st line.

This concludes the description of the linear program.

5.3 Experimental Results

Armed with the above formulation, we wrote proof search routines in Maple 10, exploiting the fast Optimization package of Maple. Thanks to built-in procedures, our code is quite short – under a few hundred lines. For large proof annotations (exceeding 100 lines), we used the freeware `lp_solve` package to solve the corresponding linear program.[2] Our routines include the following:

- `list2LP`: Takes a proof annotation as input and outputs a set of constraints corresponding to the relevant LP instance. This routine performs the translation given in the previous section.

- `proofproduce`: Takes a solution to a LP instance (given as a set of assignments to variables) and prints a line-by-line human-readable proof.

- `binarysearch`: Takes a proof annotation and range (c_1, c_2) as input, and prints a human-readable proof as well as the largest $c \in (c_1, c_2)$ (within nine digits of precision) such that an n^c lower bound could be proved for the given annotation.

- `randomadmiss`: Takes an integer k and outputs a random k-bit proof annotation, drawn uniformly at random over all such annotations. (Random bits are obtained using the Blum-Blum-Shub generator [BBS86].) To perform the sampling, we adapted a simple method for producing random well-balanced strings, given by Arnold and Sleep [AS80].

- `writeLP`: Takes an LP instance and filename and writes the LP to the `*.lp` file format, used by `lp_solve`.

After writing the routines, our first objective was to verify the lower bounds that we knew to hold, such as the $n^{1.8019}$ result. The goal here was to ensure that our choice of parameters in the 1.8019 proof were tight. We tested a 424-line LP corresponding to an annotation following the 1.8019 bound, and found that it was feasible for $c = 1.8017$ but infeasible for 1.8018; moreover, its choice of parameters mimicked ours. In contrast, we found that the choices of parameters in our $n^{1.6616}$, $n^{1.732}$, and $n^{1.784}$ lower bound proofs were *not* optimal for those proof annotations; however, the optimal choices did not yield an improvement over $n^{1.8019}$.

Our second objective was to search as much of the space of proof annotations as we could, looking for interesting patterns. For all even-numbered k from 2 to 26, we conducted an exhaustive search over all valid proof annotations with k lines. The best proof annotations for each k are given in the below table. For $k > 26$ we have not exhaustively searched the space of all proofs, but we have searched by random uniform sampling ($> 40,000$ samples) over all proof annotations – these rows in the following table are marked with an asterisk ($*$). For those rows with multiple annotations, we checked the annotations to two more decimal places to further verify that the obtained lower bounds are the same.

[2]The `lp_solve` package is an open source simplex-based linear programming solver. It is maintained by a community on Yahoo Groups: `http://groups.yahoo.com/group/lp_solve`.

#Lines	Best Proof Annotation(s)	L.B.	Δ
4	$[1,0,0]$	1.4142	0
6	$[1,0,1,0,0]$	1.5213	0.1071
	$[1,1,0,0,0]$		
8	$[1,1,0,0,1,0,0]$	1.6004	0.0791
10	$[1,1,0,0,1,0,1,0,0]$	1.633315	0.032915
	$[1,1,0,1,0,0,1,0,0]$		
	$[1,1,1,0,0,0,1,0,0]$		
12	$[1,1,1,0,0,1,0,0,1,0,0]$	1.6635	0.0302
14	$[1,1,1,0,0,1,0,0,1,0,1,0,0]$	1.6871	0.0236
16	$[1,1,1,0,0,1,0,1,0,0,1,0,1,0,0]$	1.699676	0.012576
	$[1,1,1,0,1,0,0,1,0,0,1,0,1,0,0]$		
	$[1,1,1,1,0,0,0,1,0,0,1,0,1,0,0]$		
18	$[1,1,1,1,0,0,1,0,0,1,0,0,1,0,1,0,0]$	1.7121	0.0125
20	$[1,1,1,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,0]$	1.7232	0.0111
22	$[1,1,1,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0]$	1.7322	0.0090
24	$[1,1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0]$	1.737851	0.005651
	$[1,1,1,1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0]$		
	$[1,1,1,1,1,0,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0]$		
26	$[1,1,1,1,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0]$	1.7437	0.005849
28*	$[1,1,1,1,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0]$	1.7491	0.0054
30*	$[1,1,1,1,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,0,0]$	1.7537	0.0046
32*	$[1,1,1,1,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,0]$	1.7577	0.0040
34*	$[1,1,1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,0]$	1.760632	0.002932
	$[1,1,1,1,1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,0]$		
	$[1,1,1,1,1,1,0,0,0,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,0]$		

The Δ of a row is the difference between the lower bound exponent of that row and the exponent of the previous row.

Before we analyze the above table, let us first note that the proofs produced by the above annotations bear strong similarities to those in our 1.801 lower bound. To give a small example, the best 14-line proof (establishing an $\Omega(n^{1.6871})$ time lower bound) is output as:

```
0, "DTS[n^5.275587925]"
1, "(E n^1.853485593)(A n^1.)DTS[n^3.422102331]"
2, "(E n^1.853485593)(A n^1.422102331)(E n^1.)DTS[n^2.000000001]"
3, "(E n^1.853485593)(A n^1.422102331)(E n^1.000000001)(A n^1.000000000)DTS[n^1.]"
4, "(E n^1.853485593)(A n^1.422102331)(E n^1.000000001)DTS[n^1.687100000]"
5, "(E n^1.853485593)(A n^1.422102331)DTS[n^2.846306408]"
6, "(E n^1.853485593)(A n^1.423153204)(E n^1.000000000)DTS[n^1.423153204]"
7, "(E n^1.853485593)(A n^1.423153204)DTS[n^2.401001771]"
8, "(E n^1.853485593)DTS[n^4.050730087]"
9, "(E n^1.853485593)(A n^1.000000000)DTS[n^2.197244494]"
10, "(E n^1.853485593)DTS[n^3.706971186]"
11, "(E n^1.853485593)(A n^1.000000000)DTS[n^1.853485593]"
12, "(E n^1.853485593)DTS[n^3.127015544]"
13, "DTS[n^5.275587925]"
```

(Note that the LP actually returns numbers to 18 decimal places– the proof-printing routine truncates them to make the presentation legible.) The above 14-linear applies three speedups in a row, then slowly removes the quantifiers by performing a restricted version of our conditional speedup theorem (alternating speedups and slowdowns).

Turning back to the results of the table, we see a strong correlation between later rows of the table and earlier ones. For example, there is a tie for best annotation at 10, 16, 24, and 34 lines, among three annotations that differ only in three of their bits. To develop a greater understanding of what is happening, let us introduce some abbreviations in the annotation. Where an annotation contains the string $(10)^k\ 0$, we put the symbol $\mathbf{k}$, for $k \geq 1$. Where an annotation contains the string 11000, we just put $\mathbf{0}$. The following table emerges:

#Lines	Best Proof Annotation(s)	L.B.	Δ
4	**1**	1.4142	0
6	**2**	1.5213	0.1071
	0		
8	1 **2**	1.6004	0.0791
10	1 **1 2**	1.633315	0.032915
	1 **2 1**		
	1 **0 1**		
12	1 1 **1 1 1**	1.6635	0.0302
14	1 1 **1 1 2**	1.6871	0.0236
16	1 1 **1 2 2**	1.699676	0.012576
	1 1 **2 1 2**		
	1 1 **0 1 2**		
18	1 1 1 **1 1 1 2**	1.7121	0.0125
20	1 1 1 **1 1 2 2**	1.7232	0.0111
22	1 1 1 **1 1 2 3**	1.7322	0.0090
24	1 1 1 **1 2 2 3**	1.737851	0.005651
	1 1 1 **2 1 2 3**		
	1 1 1 **0 1 2 3**		
26	1 1 1 1 **1 1 1 2 3**	1.7437	0.005849
28*	1 1 1 1 **1 1 2 2 3**	1.7491	0.0054
30*	1 1 1 1 **1 1 2 3 3**	1.7537	0.0046
32*	1 1 1 1 **1 1 2 3 4**	1.7577	0.0040
34*	1 1 1 1 **1 2 2 3 4**	1.760632	0.002932
	1 1 1 1 **2 1 2 3 4**		
	1 1 1 1 **0 1 2 3 4**		

The pattern is evident: for an optimal annotation that ends with a non-zero **k**, a longer optimal annotation can be obtained by adding either a **k** or **k+1** to the end, and a 1 at the beginning. (There are of course some restrictions– there are no more than three consecutive **1**'s, no more than two consecutive **2**'s, *etc.*) Unfortunately, we do not yet know how to *prove* that all of the best proofs must have this behavior, but it would be rather extraordinary if this pattern deviated at some later point.

We have solved the corresponding LP for many proof annotations, including those annotations used by our previous time lower bounds, with no success beyond the $2\cos(\pi/7)$ bound. The above table suggests that it should suffice for us to examine those proof annotations of the form $1 \cdots 1$ **0 1 2 3 4**$\cdots$; however, these annotations also do not lead to a better bound. To illustrate, for the 424-line proof annotation denoted by the sequence

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 **0 1 2 3 4** $\cdots$ **17 18 19**,

experiments with `lp_solve` revealed that the optimal exponent is only in the interval $[1.80175, 1.8018)$.

These empirical results point strongly to the following conjecture:

Conjecture 5.3.1 *There is no alternation-trading proof that* $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTS}[n^c]$*, for any* $c > 2\cos(\pi/7) \approx 1.8019$.

One should not be discouraged by the possible truth of the conjecture, although at present we have little idea of how to prove it. For one, it was already believed that alternation-trading proofs of time lower bounds for SAT had limitations– in particular, a quadratic time lower bound appears to be the best one could do. Secondly, the above conjecture forces us to rethink our whole approach. In order to significantly exceed the current bound, it appears that we cannot expect to rearrange the existing ingredients– we must find ways to mix in additional complexity-theoretic components. If this is true, it is very valuable knowledge. Perhaps we could use randomness as well as alternation to perform interesting "speedups" of DTS, where the addition of randomness allows us to obtain new types of proof-by-contradiction. Perhaps we can find a better way to "slowdown", removing alternations more efficiently than the simple Slowdown Lemma. There are plenty of tools out there; the challenge is to find those that can better sharpen our understanding of time lower bounds.

5.4 Other Applications

The approach of using proof annotations combined with linear programming can be applied to other polynomial-strength time lower bounds that are provable by the alternation-trading scheme– any argument that uses its own version of "speedup" and "slowdown" theorems is a potential application. We are still in the early stages of deducing the consequences that our work has for these other lower bound problems, but we do not see any serious impediment to formalizing them in a similar way. For an example of some new results obtained with our theorem prover, we state some new time-space tradeoff lower bounds for solving SAT, obtained by adapting the LP with different speedup rules. The following table gives time-space pairs for which our theorem prover has shown that no SAT algorithm can satisfy both time and space requirements simultaneously. (The proof annotation used is a sufficiently large one from the 1.801 lower bound– experimentation suggested that this annotation was the best possible, independently of the space bound.)

Time	Space
$n^{1.06}$	$n^{.9}$
$n^{1.17}$	$n^{.75}$
$n^{1.24}$	$n^{.666}$
$n^{1.36}$	$n^{.5}$
$n^{1.51}$	$n^{.333}$
$n^{1.58}$	$n^{.25}$
$n^{1.7}$	$n^{.1}$
$n^{1.75}$	$n^{.05}$

Based on the results of this table, we conjecture that the time-space product for any algorithm solving SAT is at least $\Omega(n^{2\cos(\pi/7)})$, and the product is minimized when the space is as small as possible.

5.5 Chapter Summary

We introduced a new formalism for proving time-space lower bounds in the alternation-trading framework. In this formalism, once a desired time lower bound exponent c and a sequence of proof rules are both fixed, the problem of finding a proof that uses the rules to obtain a time lower bound $\Omega(n^c)$ can be posed as a linear programming instance, solvable in polynomial time. This result, combined with a normal form theorem for alternation-trading proofs, makes it feasible to search through the space of all possible alternation-trading proofs for new time lower bounds. Implementing a small-scale theorem prover, we discovered overwhelming empirical evidence that the $\Omega(n^{2cos(\pi/7)})$ time lower bound of the previous chapter is actually *optimal* for the current framework. In principle, the linear-programming-based approach is general enough that it can be applied to a variety of settings for which we do not yet know the best lower bounds attainable.

Chapter 6

Accelerated Algorithms For a Class of NP-Hard Problems

6.1 Prologue: The Next Two Chapters

In the previous chapters, we presented a series of concrete limitations for solving hard problems under time and space constraints. We found that SAT, VERTEX COVER, MAX CUT, and other problems require at least $n^{1.8}$ time when $n^{o(1)}$ space is used. Still, we cannot yet rule out the possibility that SAT can be solved in $O(n)$ time, when $O(n)$ space is allowed![1] Indeed, the extent to which NP-hard problems are hard to solve remains largely undetermined. We do not know an $\Omega(n \cdot \text{poly}(\log n))$ time lower bound for SAT on RAMs, yet the best algorithm we know for SAT can only be shown to take $O(2^{n-n/\log n})$ time(!). That is to say, there is an enormous chasm between our knowledge of what computational resources are required to solve NP-hard problems and what resources are sufficient. In this chapter and the sequel, we examine the other side of complexity theory's coin, studying the possibility for efficient algorithms for some NP-hard problems. For some problems, it intuitively appears that the best one can do is examine every candidate solution, but this intuition has been shown to fail in many scenarios. The fledgling development of *exponential time* algorithms in recent years has indicated that for many hard problems, something substantially faster than brute-force search can be done, even in the worst case. While these algorithms are still exponential and are therefore impractical for large instances, some do work reasonably well in practice, and the existence of better algorithms gives hope that further progress is possible.

Over the last decade and a half, a tremendous amount of literature has been devoted to finding algorithms for hard problems that are exponentially faster than brute-force search. Most results present algorithms that run in $O^*(2^{\delta t})$ time for some $\delta < 1$, whereas $O^*(2^t)$ is the runtime of a naïve

[1] For some restricted computational models, this possibility *can* be ruled out. In the off-line one-tape Turing machine model, Van Melkebeek and Raz [vMR05] showed an $\Omega(n^{1+\delta})$-style time lower bound for SAT. Moreover, they use an alternation-trading argument, so our lower bound framework from the previous chapter can be applied to improve their lower bound.

brute-force search.[2] While such algorithms are typically called “exact” or “improved exponential” algorithms, these names are often considered confusing to non-practitioners. For the purposes of concrete and unambiguous terminology, we shall call them *accelerated algorithms*. A surprising number of difficult problems have been shown to exhibit accelerated algorithms. To cite a representative list of papers at this point would inevitably (and unintentionally) omit significant results; however, we find the engaging surveys by Woeginger [Woe03, Woe04] to be fairly comprehensive.

While the overall work on accelerated algorithms has remarkably flourished, researchers have still been unable to find any accelerated algorithms for some fundamental problems, after prolonged effort. Some regularly cited examples in the literature are MAX-2-SAT, MAX CUT, and SAT (for general CNF formulae). We study the first two problems and some of their relatives in the following sections, giving novel accelerated algorithms for these problems. In the chapter that follows, we discuss the prospects for an accelerated algorithm for SAT, showing how better algorithms for some polynomial time solvable problems entail better SAT algorithms.

6.1.1 Notation

Here we quickly introduce some notation used in this half of the thesis. Define $[n] := \{1, \ldots, n\}$ and $[n]^+ := [n] \cup \{0\}$. The names $x_1, \ldots, x_n$ are reserved for variables over a finite domain D. Partial assignments a to variables of V are given by a sequence of variable assignments $x_{i_1} := v_1, x_{i_2} := v_2, \ldots, x_{i_k} := v_k$, where $i_j \in [n]$, $v_j \in D$, and $k \geq 1$. For a given Boolean formula F and subset of variables S, let $F[S = a]$ denote the formula obtained by substituting the partial assignment a for the variables of S. For a clause c, define $\mathsf{vars}(c) \subseteq \{x_1, \ldots, x_n\}$ to be the set of variables that appear in c. Throughout, ω refers to the smallest real number such that for all $\varepsilon > 0$, matrix multiplication over a ring can be performed in $O(n^{\omega+\varepsilon})$ ring operations.

6.2 Introduction

In this chapter, we present a novel accelerated algorithm for exactly solving (in fact, counting solutions to) constraint satisfaction optimization problems with at most two variables per constraint, along with some generalizations. The algorithm can count the number of optima in MAX 2-SAT and MAX CUT instances in $O(m^3 2^{\omega n/3})$ time, where $\omega < 2.376$ is the matrix product exponent over a ring. Our construction shows that improvement in the runtime exponent of either *k-clique* solution (even when $k = 3$) or *matrix multiplication over* GF(2) would consequently improve the runtime exponent for solving these optimization problems.

There has been notable theoretical interest in discovering accelerated algorithms for MAX-2-SAT and structurally similar problems. Unlike problems such as VERTEX COVER and k-SAT, where analysis of branch-and-bound techniques (with or without randomness) has sufficed for improving the naïve time bounds (*e.g.* [Rob86, PPSZ05, DGHKKPRS02]), the MAX-SAT problem has been surprisingly difficult to attack. Prior work has only found algorithms for special cases, such as sparse instances [NR00, CK04, Wil03, DW06] or approximate solutions [Hir03, DGHK01, LLZ02]. Substantial work has gone into finding exact algorithms for MAX 2-SAT [BR99, MR99, Hir00,

[2] Recall that the O^* notation suppresses polynomial factors.

GN00, KK06] and Max Cut [KF02, SS03, KMRR05, SS06], but our general algorithm scheme is the only one known to solve the two problems in $2^{\delta n}$ time (for some universal $\delta < 1$) on all possible instances.

6.2.1 Outline of our approach: Split and List

Most exact algorithms for NP-hard problems in the literature involve either a case analysis of a branch-and-bound strategy (*e.g.* [GHNR03]), repeated random choice of assignments (*e.g.* [PPSZ05]), or local search (*e.g.* [Sch99]). Our design departs from these approaches, and applies a form of dynamic programming akin to earlier algorithms from the 70's [HS74, SS81]. We call this dynamic programming strategy the *split-and-list* approach. The basic idea is to *split* up the problem instance into parts, enumerate (or *list*) all possible solutions for the individual parts separately, then give an interesting way to combine the parts so that a global solution can be efficiently found. Typically the number of parts is chosen to be a small constant so that a polynomial time algorithm can combine them. The downside to choosing a small number of parts is that each part must be relatively large with respect to the input size, thus the list of possible solutions to each part will be (exponentially) large.

Example. The general paradigm of split-and-list was first used to solve the Subset Sum problem in $O^*(2^{n/2})$ time [HS74]. In this problem, one is given a set S of n integers and a target integer T, and is asked if there is a subset of S whereby $\sum_{x \in S} x = T$. A split-and-list algorithm for the problem is obtained by first splitting the list of integers into two parts of $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ integers respectively, and listing all possible subsets of the two parts (having $2^{\lceil n/2 \rceil}$ and $2^{\lfloor n/2 \rfloor}$ subsets, respectively). For one of the lists (call it L), associate each subset s in L with a key $k_s = \sum_{x \in S} x$. Sort L by its keys. Now for every subset s' in the other list, compute the sum of integers in s' as k' and binary search in L for a subset s with key value $k_s = T - k'$. If such an s is ever found, return $s \cup s'$; otherwise, return that there is no solution. It is easy to see that this algorithm takes only $O^*(2^{n/2})$ time (where the O^* hides the cost of addition, and an $O(n)$ factor resulting from the sorting of a $2^{n/2}$ length list). This split-and-list algorithm exponentially reduces Subset Sum to a problem that is in $O(n \log n)$ time: given a list of numbers and a target, determine if there is a pair of numbers that sum to the target.

Our approach to solving Max 2-Sat and its relatives bares some similarities to the above canonical example. We split the set of n variables into k partitions (for $k \geq 3$) of (roughly) equal size, and list the $2^{n/k}$ variable assignments for each partition. From these $k2^{n/k}$ assignments, we build a graph with weights on its nodes and edges, arguing that an optimum weight k-clique in the graph corresponds to an optimum solution to the original instance. The weights are eliminated using a polynomial reduction, and a fast k-clique algorithm on undirected graphs yields the accelerated algorithm.

6.3 Fast k-Clique Detecting and Counting

We first review an algorithm by Nesetril and Poljak [NP85] for detecting if a graph has a k-clique in less than n^k steps.

Theorem 6.3.1 *([NP85]) Let $r \in \mathbb{Z}^+$. Then $3r$-clique on undirected graphs is solvable in $O(n^{\omega r})$ time.*

Proof. First consider the case $k = 3$. Given $G = (V, E)$ with $n = |V|$, let $A(G)$ be its adjacency matrix. Recall that $\mathrm{tr}(M)$, the trace of a matrix M, is the sum of the diagonal entries. The quantity $\mathrm{tr}(A(G)^3)$ is computable in two matrix multiplications, and it is easy to see that $\mathrm{tr}(A(G)^3)$ is non-zero if and only if there is a triangle in G. (This observation was first made by Itah and Rodeh [IR78].) For $3r$-cliques when $r > 1$, build a graph $G_r = (V_r, E_r)$ where V_r is the collection of all r-cliques in G, and $E_r = \{\ \{c_1, c_2\}\ :\ c_1, c_2 \in V_r,\ c_1 \cup c_2 \text{ is a } 2r\text{-clique in } G\}$. Observe that each triangle in G_r corresponds to a unique $3r$-clique in G. Therefore $\mathrm{tr}(A(G_r)^3) \neq 0$ if and only if there is a $3r$-clique in G, which is determined in $O(n^{\omega r})$ time. To find an explicit $3r$-clique given that one exists, take an i such that $A(G_r)^3[i, i] \neq 0$, and try all $2r$-sets of vertices to extend the r-clique denoted by i to a $3r$-clique. This takes $O(n^{2r})$ time. □

In fact, the above approach may be used to *count* the number of k-cliques as well. Let $C_k(G)$ be the set of k-cliques in G, and G_r be as defined in the previous proof.

Proposition 6.3.1 *For all $r \geq 1$, $\mathrm{tr}(A(G_r)^3) = \binom{3r}{r}\binom{2r}{r} \cdot |C_{3r}(G)|$.*

Proof. Consider the case $r = 1$. In $\mathrm{tr}(A(G)^3)$, each triangle $\{v_i, v_j, v_k\}$ is counted once for each vertex v (say, v_i) in the triangle, times the two paths traversing the triangle starting from that v (for v_i, they are $v_i \to v_j \to v_k \to v_i$ and $v_i \to v_k \to v_j \to v_i$). Similar reasoning shows that each $3r$-clique is counted $\binom{3r}{r}\binom{2r}{r}$ times in $\mathrm{tr}(A(G_r)^3)$, as $\binom{3r}{r}\binom{2r}{r}$ is the number of ways to partition a $3r$-set into an ordered sequence of three r-sets. □

Observe that the above can be easily generalized to directed graphs, when we define a k-clique in a directed graph to be a k-set of nodes $\{v_1, \ldots, v_k\}$ where the edge (v_i, v_j) appears for all $i \neq j$.

6.4 General Algorithm for a Class of Optimization Problems

In this section, we show how a very general problem (that we call the Weighted 2-Constraint Satisfaction Problem) has an accelerated algorithm, by a reduction to the k-clique counting algorithm of the previous section. In later sections, we present an extension of this result to constraints that are representable by degree-two polynomials.

Problem: Weighted 2-CSP

Input: Integers $K_v, K_e \in [n^\ell]^+$ for a fixed integer $\ell > 0$ independent of the input, a finite domain D, and functions

$$w_i : D \to [n^\ell]^+,\ \ \forall i = 1, \ldots, n, \qquad w_{(i,j)} : D \times D \to [n^\ell]^+,\ \ \forall i, j = 1, \ldots, n,\ i \neq j.$$

Output: A variable assignment $a = (a_1, \dots, a_n) \in D^n$ satisfying

$$\sum_{i=1}^{n} w_i(a_i) = K_v, \quad \sum_{i,j} w_{(i,j)}(a_i, a_j) = K_e.$$

We define $\sum_{i=1}^{n} w_i(a_i)$ to be the *variable weight of assignment* a, and $\sum_{i,j} w_{(i,j)}(a_i, a_j)$ to be the *pair weight of assignment* a. The name "weighted 2-CSP" comes from the fact that each weight function is a constraint on at most two variables, and the objective is to satisfy certain equations involving those constraints. We also consider the counting version of the weighted problem.

Problem: Count 2-CSP

Input: Same as above.

Output: The number A of variable assignments that satisfy the above output conditions.

Weighted 2-CSP is easily seen to be NP-complete. Below are a few important examples of NP-complete problems that can be effectively solved using an oracle for Weighted 2-CSP, and polynomial time overhead. All of the below problems have received substantial attention in the computer science literature.

- Vertex Cover/Independent Set: Set $D = \{0, 1\}$. Given $G = (V, E)$, assume without loss of generality that $V := [n]$. Set $w_i(x_i) := 1$ for all i, $K_e := |E|$, and

$$w_{(i,j)}(x_i, x_j) := \begin{cases} 1 & \text{if } (i,j) \in E \text{ and } (x_i \vee x_j) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

  Now binary search for the minimum K_v such that there is an assignment with variable weight K_v and pair weight K_e; this corresponds to finding the minimum size of a vertex cover for G. Independent Set can be formulated similarly.

- Max 2-Sat: Set $D := \{0, 1\}$. Given a 2-CNF formula F on variables $x_1, \dots, x_n$, set $w_i(x_i) := 0$ and $K_v := 0$. For all $i < j$ define

$$w_{(i,j)}(x_i, x_j) = \sum_{c \in F \,:\, \mathsf{vars}(c) \subseteq \{x_i, x_j\}} c(x_i, x_j);$$

  that is, $w_{(i,j)}(x_i, x_j)$ is the sum of clauses c in F such that $\mathsf{vars}(c) \subseteq \{x_i, x_j\}$, and c is satisfied by the given assignment to x_i and x_j. Note $w_{(i,j)}(x_i, x_j) \in [4n^2]^+$, since the total number of distinct clauses is at most $4n^2$. Binary search for the maximum $K_e \in [4n^2]$ whereby there is a variable assignment with pair weight K_e; this corresponds to finding the maximum number of satisfiable clauses in F.

- Max Cut: Set $D := \{0, 1\}$. Given a directed graph $G = (V, E)$, assume without loss of generality that $V = [n]$. Set $w_i(x_i) := 0$ for all i, and

$$w_{(i,j)}(x_i, x_j) := \begin{cases} x_i \texttt{ XOR } x_j & \text{if } (i,j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, set $K_v := 0$. To find a maximum cut, binary search for the largest value of $K_e \in [n(n-1)]$ such that there is a variable assignment with that pair weight– this is a weighted 2-CSP instance. Note the above reduction can be trivially modified to accommodate undirected graphs as well.

- Min Bisection: In this problem, we are given a directed graph (V, E) where $n = |V|$ is even, and we wish to find a $S \subseteq V$ such that $|S| = n/2$ and the cut value of S is minimized. To reduce this problem to a weighted 2-CSP, the setup is the same as for Max Cut, except one sets $w_i(a_i) := a_i$ for all i and $K_v := n/2$. To find a minimum bisection, binary search for the smallest $K_e \in [n(n-1)]$ so that there is a variable assignment with variable weight exactly K_v and pair weight exactly K_e.

- Sparsest Cut: Here we are given a directed graph (V, E) and wish to find a cut $S \subseteq V$ that minimizes the ratio $\frac{\delta(S)}{|S||V-S|}$, where $\delta(S)$ is the cut value of S. To reduce the problem to a weighted 2-CSP, solve the weighted 2-CSP instance for Min Bisection over all $O(n^3)$ ways to assign $K_v \in [n]$ and $K_e \in [n(n-1)]$. From these instances, build a $n^2 \times n$ Boolean table T that has $T[i,j] = 1$ iff there is a cut with i nodes on one side, having cut value j. Using T, compute the minimum ratio $\frac{j}{i \cdot (n-i)}$ over those (i, j) satisfying $T[i, j] = 1$; this is the sparsest cut, by definition.

It is evident that Weighted 2-CSP is capable of succinctly expressing many interesting problems. While Vertex Cover and Independent Set were known to admit accelerated algorithms (the fastest known, to our knowledge, is Robson's $O(1.1889^n)$ algorithm [Rob86]), the other four problems were not known to have accelerated algorithms. In the next section, we give an accelerated algorithm for Count 2-CSP, yielding accelerated algorithms for the above problems and more.

6.4.1 Main Algorithm

For $k \in \mathbb{Z}^+$, define $\kappa(k)$ to be the smallest real number such that the number of k-cliques in a given n-node graph can be computed in $O(n^{\kappa(k)})$ time. One may think of $\kappa(k)$ as the "k-clique exponent", similar to the matrix multiplication exponent ω. Note that Theorem 6.3.1 states that $\kappa(3r) \leq \omega r$, for all constants r.

Theorem 6.4.1 *Let $k(n) \geq 3$ be a monotone non-decreasing function such that $k(n)$ can be computed in $n^{O(1)}$ time. Then* Count 2-CSP *instances with weights in $[n^\ell]^+$ are solvable in*

$$n^{O(\ell \cdot k(n)^2)} \cdot (k(n) d^{\frac{n}{k(n)}})^{\kappa(k(n))} \text{ time,}$$

where n is the number of variables, and d is the domain size.

Corollary 6.4.1 *The number of optima for a given* Max 2-Sat, Max Cut, Min Bisection, *or* Sparsest Cut *instance can be determined in $O^*(1.732^n)$ time, and an optimal assignment can be found in $O^*(1.732^n)$ time.*

Proof of Corollary 6.4.1. Set $d = 2$ and $k = 3$, and use the aforementioned reductions from WEIGHTED 2-CSP to these problems. Explicit assignments/cuts can be found using self-reducibility, increasing the runtime by only an $O(n)$ factor. □

Unfortunately, the best known k-clique algorithms do not improve the runtime of Corollary 6.4.1 when $k > 3$. For example, the best known 9-clique algorithm uses Theorem 6.3.1, and runs in $O(n^{3\omega})$ time– so the improvement in the exponent over the obvious $O(n^9)$ algorithm is a factor of $\omega/3$, just as in the 3-clique case. Finding a 9-clique algorithm that runs in $O(n^{3\omega-\delta})$ for some $\delta > 0$ would immediately give an improvement to Corollary 6.4.1.

Proof of Theorem 6.4.1. We reduce the problem to counting k-cliques in a large graph. Assume without loss of generality that n is divisible by k. Let C be a given instance of WEIGHTED 2-CSP.

Step 1: Build a graph (Split and List).

Arbitrarily split the n variables of C into sets $P_1, P_2, \ldots, P_k$ with n/k variables each. For each P_i, make a list L_i of all $d^{n/k}$ possible assignments to the variables of P_i. Construe the elements of $L_1, \ldots, L_k$ as nodes of a k-partite complete directed graph $G = (V, E)$, having $d^{n/k}$ nodes per partition and arcs between every pair of nodes from different parts (in both directions). For a vertex $v \in L_\ell$ for some $\ell \in [k]$, let a^v denote the partial assignment to which v refers, and let a_i^v be the assignment to variable x_i in L_ℓ.

Step 2: Weight nodes and edges accordingly.

Define a weight function $W : V \cup E \to \mathbb{Z}$ as follows:

- For $\ell \in [k]$ and $v \in L_\ell$ ($\ell \in [k]$), $W(v) := \sum_{x_i \in P_\ell} w_i(a_i^v) + \sum_{x_i, x_j \in P_\ell} w_i(a_i^v, a_j^v)$.
- For $\ell, \ell' \in [k]$, $\ell \neq \ell'$, and $u \in L_\ell, v \in L_{\ell'}$, $W(u, v) := \sum_{x_i \in P_\ell, x_j \in P_{\ell'}} w_{(i,j)}(a_i^u, a_j^v)$.[3]

Let $C_k = \{v_1, \ldots, v_k\}$ be a k-clique in G. Define the *edge-weight of* C_k to be $W_e(C_k) := \sum_{i,j \in [k], i \neq j} W(v_i, v_j)$, and the *node-weight of* C_k to be $W_v(C_k) = \sum_{i=1}^{k} W(v_i)$.

Claim 3 *The number of k-cliques with node-weight K_v and edge-weight K_e in G is equal to* COUNT 2-CSP*(C),* i.e.*, the number of variable assignments for C with variable weight K_v and pair weight K_e.*

To prove this claim, let a be an assignment to the variables of C, and suppose that a has pair and variable weight K_e and K_v, respectively. Clearly, there exist unique vertices $v_i \in L_i$ for $i = 1, \ldots, k$ such that $a = a^{v_1} a^{v_2} \cdots a^{v_k}$, *i.e.* the assignment a corresponds to a particular clique

[3]As an example, consider the case where we merely want to solve MAX CUT. The $v \in L_i$ denote all $2^{n/k}$ possible cuts with a distinct "left" and "right" side on the subgraph of n/k vertices P_i. Every $W(v)$ is the number of edges crossing the "sub-cut" defined by the node v, and $W(u, v)$ is the number of edges crossing from one side of the cut defined by u to the opposite side of the cut defined by v.

$C_a = \{v_1, \ldots, v_k\}$ in G. Since every variable appears in exactly one part, we have

$$\begin{aligned} W_v(C_k) &= \sum_{i=1}^{k} W(v_i) \\ &= \sum_{i=1}^{k} \sum_{x_{i'} \in P_i} w_{i'}(a_{i'}^{v_i}) \\ &= \sum_{i'=1}^{n} w_{i'}(a_{i'}), \end{aligned}$$

which is precisely the variable weight of a. Similarly,

$$\begin{aligned} W_e(C_a) &= \sum_{i,j \in [k],\ i \neq j} W(v_i, v_j) \\ &= \sum_{i,j \in [k],\ i \neq j} \sum_{x_{i'} \in P_i, x_{j'} \in P_j} w_{(i',j')}(a_{i'}^{v_i}, a_{j'}^{v_j}) \\ &= \sum_{i',j' \in [n],\ i' \neq j'} w_{(i',j')}(a_{i'}, a_{j'}), \end{aligned}$$

which is just the pair weight of a. Therefore $W_v(C_a) = K_v$, and $W_e(C_a) = K_e$. Since there is a one-to-one correspondence between k-cliques in G and assignments to C, and because k-cliques with node and edge weight K_v and K_e correspond to assignments with variable and pair weight K_v and K_e, the claim is proved.

Step 3: Reduce the weighted graph to a collection of unweighted graphs.

A problem remains in our above construction. We would like to count k-cliques in the above graph with particular node and edge weights, but the clique-counting algorithm of Theorem 6.3.1 only works on unweighted graphs. We can remove this difficulty and tack on a multiplicative factor that is polynomial in n^ℓ, but exponential in k. Let T_v be the set of k-tuples $(j_1, \ldots, j_k)$ where each $j_i \in [n^\ell]^+$ and $\sum_{i=1}^k j_i = K_v$. Let T_e be the set of $k(k-1)$-tuples of the form

$$(i_{1,2}, i_{2,1}, i_{1,3}, i_{3,1} \ldots, i_{1,k-1}, i_{k-1,1}, i_{2,3}, i_{3,2}, \ldots, i_{k-1,k}, i_{k,k-1})$$

where each $i_{j,l} \in [n^\ell]^+$, and $\sum_{a,b\,:\,a \neq b} i_{a,b} = K_e$. For each choice of tuple $(j_1, \ldots, j_k) \in T_v$ and $(\{i_{a,b}\}) \in T_e$, construct a unweighted directed graph $G_{(\{i_{a,b}\}, j_1, \ldots, j_k)}$ that contains a subset of the nodes and edges of G, namely:

- For $v \in L_{a_1}$, put node v in $G_{(\{i_{a,b}\}, j_1, \ldots, j_k)}$ if and only if $W(v) = j_{a_1}$.
- Out of those $u \in L_{a_1}$ and $v \in L_{a_2}$ that were put into $G_{(\{i_{a,b}\}, j_1, \ldots, j_k)}$ (with $a_1 \neq a_2$), put edge (u, v) in $G_{(\{i_{a,b}\}, j_1, \ldots, j_k)}$ if and only if $W(u, v) = i_{a_1, a_2}$.

Intuitively, the above rules "screen" the edges and nodes of the graph G, and only include them in $G_{(\{i_{a,b}\}, j_1, \ldots, j_k)}$ if they have a prescribed weight.

Now, each k-clique counted in $G_{(\{i_{a,b}\}, j_1, \ldots, j_k)}$ is a k-clique of edge weight K_e and node weight K_v in G, by construction. Moreover, for each pair of tuples $(\{i_{a,b}\})$ and $(j_1, \ldots, j_k)$, the graph

$G_{(\{i_{a,b}\},j_1,\ldots,j_k)}$ represents a distinct collection of such cliques in G. Hence the total number of k-cliques counted over all graphs $G_{(\{i_{a,b}\},j_1,\ldots,j_k)}$ is the number k-cliques in G with edge weight K_e and node weight K_v.

Recall that G has $kd^{n/k}$ nodes. By iterating over all tuples $(j_1,\ldots,j_k) \in T_v$ and $(\{i_{a,b}\}) \in T_e$, constructing the relevant $G_{(\{i_{a,b}\},j_1,\ldots,j_k)}$, and counting its k-cliques in $O((kd^{n/k})^{\kappa(k)})$ time, Count 2-CSP can be computed in $O(|T_e| \cdot |T_v| \cdot (kd^{n/k})^{\kappa(k)})$ time. The cardinalities of T_v and T_e are bounded from above by $(K_v)^{k-1} \leq (n^\ell+1)^{k-1}$ and $(K_e)^{k^2-k} \leq (n^\ell+1)^{k^2-k}$, respectively. One can easily enumerate each tuple in T_e and T_v in $O(\text{poly}(n))$ amortized time. Therefore the runtime of the procedure for Count 2-CSP is at most $n^{O(\ell k(n)^2)} \cdot (k(n)d^{\frac{n}{k(n)}})^{\kappa(k(n))}$. □

Sampling Solutions. The above algorithm can be used to *randomly sample* solutions to a given weighted 2-CSP problem.

Corollary 6.4.2 *After $O^*(2^{\omega n/3})$ preprocessing, one can generate a uniform random sample from the set of solutions to a* Weighted 2-CSP *instance having variable weight exactly K_v and pair weight exactly K_e, in $O^*(2^{2n/3})$ time.*

For example, in $O^*(2^{\omega n/3})$ time, one can produce a maximum cut, chosen uniformly at random over all maximum cuts.

Proof. The preprocessing phase consists of just running the algorithm from the previous theorem with $k = 3$, saving the results of each matrix multiplication performed by the triangle counting subroutine. Let N be the total number of solutions to the weighted 2-CSP instance. For a graph $G_{(\{i_{a,b}\},j_1,j_2,j_3)}$ as defined in Theorem 6.4.1, let $N_{(\{i_{a,b}\},j_1,j_2,j_3)}$ be the total number of k-cliques in it. We have that $\sum N_{(\{i_{a,b}\},j_1,j_2,j_3)} = N$.

To randomly sample a solution, select graph $G_{(\{i_{a,b}\},j_1,j_2,j_3)}$ from the collection of unweighted graphs with probability $\frac{N_{(\{i_{a,b}\},j_1,j_2,j_3)}}{N}$, and then pick a triangle from the selected graph uniformly at random, and return the variable assignment corresponding to it. To pick a triangle, let A be the adjacency matrix of the selected graph. For each edge (i,j), determine the number of triangles that contain the pair i,j by reading $A^2[i,j]$ (which was computed in the preprocessing phase). Pick the edge (i,j) with probability $\frac{A^2[i,j]}{\sum_{(i,j) \text{ is an edge}} A^2[i,j]}$. Out of those vertices adjacent to both i and j, pick one uniformly at random. The resulting edge and vertex pair is a uniform random triangle. □

6.5 Weighted Polynomial Constraint Problems on Boolean Variables

The algorithm as stated in the previous section only works when the weight functions under consideration are functions of at most two variables. In the following, we describe an extension of the algorithm that works for a class of weight functions on possibly more than two Boolean variables, namely polynomials of degree two.

Definition 6.5.1 *A polynomial $p(x_1, \ldots, x_n)$ is* multilinear *if p is a sum of terms of the form $cx_1^{d_1} \cdots x_n^{d_n}$, where $c \in \mathbb{R}$ and $d_i \in \{0, 1\}$ for all i.*

The following is well-known folklore.

Theorem 6.5.1 *Every Boolean function f has a unique representation as a multilinear polynomial of the form $f : \{0,1\}^n \to \{0,1\}$.*

Proof. Write the function f in disjunctive normal form, ensuring that no two conjunctions in the disjunction can be true simultaneously. Then, replace the ORs with *addition*, ANDs with *multiplication*, and every $\neg x_i$ with the expression $(1 - x_i)$. The resulting polynomial is multilinear and it is easy to check that $f(a) = 1$ when $f(a)$ is true, and $f(a) = 0$ when $f(a)$ is false. The polynomial f is unique, because if f' is a polynomial that agrees with f on all points in $\{0,1\}^n$, then $f \equiv f'$. (The proof is by induction on the number of variables– in particular, any nonzero polynomial cannot be zero on all points in $\{0,1\}^n$.) □

With the above characterization, it makes sense to define the degree of a Boolean function.

Definition 6.5.2 *Let f be a Boolean function. Then deg(f) is the degree of the unique multilinear polynomial that represents f from $\{0,1\}^n$ to $\{0,1\}$.*

We define the problem Weighted Degree-Two CSP as follows. The setup of Weighted Degree-Two CSP is about the same as Weighted 2-CSP, except that instead of just two-variable weight functions, we allow for degree-two polynomials.

Problem: Weighted Degree-Two CSP

Input: A set of degree-two polynomials $\mathcal{S} = \{p_1(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n)\}$ such that $m = \text{poly}(n)$, $p_i : \{0,1\}^n \to \{0,1\}$, a weight function $w_i : \{0,1\} \to [\text{poly}(n)]^+$ for $i = 1, \ldots, n$, and integers $K_v, K_p \in [\text{poly}(n)]^+$.

Output: A variable assignment $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ satisfying $\sum_i w_i(a_i) = K_v$ and $\sum_j p_j(a) = K_p$.

This problem is a generalization of Weighted 2-CSP over Boolean-valued weight functions, since every two-variable Boolean function can be represented by a degree two polynomial, via Theorem 6.5.1. The main result of this section is:

Theorem 6.5.2 Weighted Degree-Two CSP *is solvable in $O^*(2^{\omega n/3})$ time.*

Ostensibly, it is not clear if Theorem 6.5.2 says anything new– are there any Boolean functions on three or more variables that have degree two? The answer is yes. Three examples of this phenomenon are the not-all-equals predicate on three variables, the selector predicate on three variables, and the "sort" predicate on four variables, each defined as follows:

- The Boolean function $\texttt{NAE}(x_1, x_2, x_3) = 1$ if and only if there are $i \neq j$ such that $x_i \neq x_j$. The polynomial representation for this function is $\texttt{NAE}(x_1, x_2, x_3) = x_1+x_2+x_3-x_1x_2-x_1x_3-x_2x_3$. Note this polynomial is Boolean-valued when x_1, x_2, and x_3 are drawn from $\{0, 1\}$.
- $\texttt{SEL}(x_1, x_2, x_3) = 1$ if and only if the predicate "if x_1 then x_2 else x_3" is true. We have $\texttt{SEL}(x_1, x_2, x_3) = x_1x_2 + x_3 - x_1x_3$.
- $\texttt{SORT}(x_1, x_2, x_3, x_4) = 1$ if and only if the bit string $x_1x_2x_3x_4$ is sorted, in either ascending or descending order. That is, $\texttt{SORT}$ is true iff it is given one of the eight strings 0000, 0001, 0011, 0111, 1111, 1110 1100, 1000. Interestingly, it can be verified that $\texttt{SORT}(x_1, x_2, x_3, x_4) = 1 - x_2 - x_3 + x_1x_2 - x_1x_4 + x_2x_3 + x_3x_4$.

Thus, a corollary of Theorem 6.5.2 is that optimization problems such as Max NAE-3-Sat and Max SEL-3-Sat (defined with respect to the above functions) have accelerated algorithms. It turns out that a degree two polynomial representing a Boolean function depends on at most four variables. In particular, the following can be shown.

Theorem 6.5.3 (Nisan-Szegedy [NS94]) *Let p be a degree d polynomial representing a Boolean function. Then p depends on at most $d2^{d-1}$ of its variables.*

It follows that Weighted Degree-Two CSP does not include Boolean constraint satisfaction problems that depend on more than four variables.

Proof of Theorem 6.5.2. We shall just describe how to modify the proof of Theorem 6.4.1 to accommodate polynomials instead of 2-constraints. The construction of the graph remains basically the same as before, but the weighting scheme needs some changes.

Write each polynomial p_i in a given Weighted Degree-Two CSP instance as:

$$p_i(x_1, \ldots, x_n) = c_{i,0} + \sum_{j=1}^{n} c_{i,j} \cdot x_i + \sum_{j_1<j_2} c_{i,j_1,j_2} \cdot x_{j_1}x_{j_2},$$

where each $c_{i,j}, c_{i,j,k} \in \mathbb{R}$.

We use the same definition of nodes and edges from Theorem 6.4.1, except now the edges are undirected. Weights are added to the edges and nodes of the constructed graph G according to these rules:

- On a node v from list L_ℓ, put two kinds of weight: $W_1(v) = \sum_{x_i \in P_\ell} w(a_i^v)$ and $W_2(v) = \sum_{i=1}^{m} \left(\sum_{x_{j_1}, x_{j_2} \in P_\ell} c_{i,j_1,j_2} \cdot a_{j_1}^v \cdot a_{j_2}^v + \sum_{x_j \in P_\ell} c_{i,j} \cdot a_j^v \right)$. Effectively, $W_2(v)$ is the contribution of the partial assignment a^v to the sum of all degree-one and degree-two terms with variables from P_ℓ, and $W_1(v)$ is the contribution of a^v to the variable weight.
- On an edge $\{u, v\}$ where u is from L_{ℓ_1} and v is from L_{ℓ_2}, put the weight $W(u, v) = \sum_{i=1}^{m} \sum_{x_{j_1} \in P_{\ell_1}, x_{j_2} \in P_{\ell_2}} c_{i,j_1,j_2} \cdot a_{j_1}^u \cdot a_{j_2}^v$. Thus the edge weight $\{u, v\}$ is the contribution of the partial assignments a^u and a^v to the sum of all degree-two terms with one variable from P_{ℓ_1} and one variable from P_{ℓ_2}.

Now for each polynomial p_j, all of its terms other than the degree-zero coefficients are counted by either an edge or node weight. The following claim is straightforward; its proof resembles that of Claim 3.

Claim 4 *There is a one-to-one correspondence between k-cliques $\{v_1, \ldots, v_k\}$ in G satisfying*

$$\sum_{i=1}^{k} W_1(v_i) = K_v, \quad \sum_{j=1}^{m} a_{j,0} + \sum_{\{v_i,v_j\}} W(v_i, v_j) + \sum_{i=1}^{k} W_2(v_i) = K_p,$$

and variable assignments $a \in \{0,1\}^n$ with

$$\sum_{i=1}^{n} w_i(a_i) = K_v, \quad \sum_{j=1}^{m} p_j(a) = K_p.$$

As in Theorem 6.4.1, we cannot apply the fast k-clique counting algorithm directly to the weighted graph G; it is necessary to eliminate the weights in some manner. The node and edge weights can be eliminated by applying additional observations.

Claim 5 *Let p be a degree two polynomial representing a Boolean function on n variables. Then the degree zero coefficient of p is in $\{0,1\}$, the degree one coefficients are in $\{-1,0,1\}$, and the degree two coefficients are in $\{-2,-1,0,1,2\}$.*

Proof. By induction on n. Clearly the degree zero coefficient must be 0 or 1, since $p(0,\ldots,0)$ equals it. When $n = 0$, the claim is trivial. Assume the claim holds for degree two polynomials on $n-1$ variables. Given a degree-two polynomial p representing a Boolean function, write it as

$$p(x_1, \ldots, x_{n-1}, x_n) = x_n q(x_1, \ldots, x_{n-1}) + (1 - x_n) r(x_1, \ldots, x_{n-1}),$$

where $r = p(x_1, \ldots, x_{n-1}, 0)$ and $q = p(x_1, \ldots, x_{n-1}, 0)$ are degree two polynomials. Write q and r as

$$q = c_0^q + \sum_{i=1}^{n} c_i^q x_i + \sum_{i \neq j} c_{ij}^q x_i x_j$$

and

$$r = c_0^r + \sum_{i=1}^{n} c_i^r x_i + \sum_{i \neq j} c_{ij}^r x_i x_j.$$

Since $r = p(x_1, \ldots, x_{n-1}, 0)$ and $q = p(x_1, \ldots, x_{n-1}, 0)$ are degree two Boolean functions on $n-1$ variables, the claim holds for q and r by induction. Hence $c_0^q, c_0^r \in \{0,1\}$, $c_i^q, c_i^r \in \{-1,0,1\}$ for all

i, and $c^q_{ij}, c^r_{ij} \in \{-2, -1, 0, 1, 2\}$ for all i and j. We can express p as:

$$\begin{aligned} p &= x_n q + (1 - x_n) r \\ &= \sum_{i \neq j} (c^q_{ij} - c^r_{ij}) x_i x_j x_n \\ &+ \sum_{i=1}^{n-1} (c^q_i - c^r_i) x_i x_n + \sum_{i \neq j} c^r_{ij} x_i x_j \\ &+ (c^q_0 - c^r_0) x_n + \sum_{i=1}^{n-1} c^r_i x_i \\ &+ c^r_0. \end{aligned}$$

Note $c^q_{ij} = c^r_{ij}$ for all i, j, since p is of degree two. Since $c^q_i, c^r_i \in \{-1, 0, 1\}$ for all i, it follows that $c^q_i - c^r_i \in \{-2, -1, 0, 1, 2\}$. Since $c^q_0, c^r_0 \in \{0, 1\}$, it follows that $c^q_0 - c^r_0 \in \{-1, 0, 1\}$. This completes the proof. $\square$

Claim 6 *The number of possible weights for any node or edge is at most* $4m(n^2 + 1) + 1$.

Proof. By the previous claim, the coefficient for any monomial in a polynomial p_i is in the range $\{-2, -1, 0, 1, 2\}$. The weight of any node or edge is just the sum of some subset of coefficients from some subset of the polynomials: every variable is either 0 or 1, so every monomial evaluates to either its coefficient or 0. Since there are $\binom{n}{2} + n + 1 \leq n^2 + 1$ monomials and m polynomials, the value of any such sum is an integer in the range $[-2m(n^2 + 1), 2m(n^2 + 1)]$, so the weight of any node or edge is an integer in that range. The claim follows since there are $4m(n^2 + 1) + 1$ integers in the range. $\square$

Therefore, the number of all possible weight combinations for the nodes and edges of a k-clique is at most $(4m(n^2 + 1) + 1)^{k^2} \leq \text{poly}(n)$, when k is fixed. For each such combination, we can set up an unweighted k-clique instance as in the proof of Theorem 6.4.1, performing k-clique detection for each unweighted instance in $O^*(2^{\omega n/3})$ time. It follows that WEIGHTED DEGREE-TWO CSP can be solved in $O^*(2^{\omega n/3})$. $\square$

The above algorithm can also be extended to problems where the possible assignments vary over non-Boolean (*e.g.* ternary) domains. We chose to restrict ourselves to the Boolean case due to its naturalness and simplicity.

6.6 A Potential Application to Breaking a Class of Cryptosystems

As we have seen, our framework for solving weighted constraint problems is quite general. In this section, we extend it even further, assuming that a certain variant of the k-CLIQUE problem can be solved more efficiently than brute force search. More precisely, we show how an improved algorithm for a problem called EDGE-WEIGHT k-CLIQUE can be used to solve MULTIVARIATE QUADRATIC EQUATIONS (abbreviated as MQS) with an accelerated algorithm. In the MQS problem, one is given a set of m equations over n variables that take values from a finite field F, where each equation is of the form

$$p(x_1, \ldots, x_n) = 0$$

for a degree-two polynomial p. The task is to find an assignment $(x_1, \ldots, x_n) \in F^n$ that satisfies every equation. MQS is somewhat different from WEIGHTED DEGREE-TWO CSP, and possibly more difficult, because the cardinality of the range of some $p(x_1, \ldots, x_n)$ in an MQS instance is not necessarily bounded by a polynomial in n.

Several very important cryptosystems have been designed under the assumption that MQS is intractable, even in the average case. In particular, the security of the Advanced Encryption Standard (AES) or Rijndael cipher (used by the U.S. Government to encrypt highly sensitive data) [Lan04] and the Hidden Field Equations (HFE) public key cryptosystem depend on the intractability of MQS [Pat95]. The basic idea behind these systems is to encrypt a string $b_1 \cdots b_n$ of n bits by picking n random quadratic polynomials $p_1(x_1, \ldots, x_n), \ldots, p_n(x_1, \ldots, x_n)$ and send the quadratic equations $p_1(x_1, \ldots, x_n) = p_1(b_1, \ldots, b_n)$ as the encrypted message. In this proposal, encryption is simple, but decryption is presumably very difficult– we also have to hide some kind of trapdoor among the equations, otherwise even the intended recipient cannot decrypt the message efficiently. The cryptosystems designed around MQS employ various methods to "embed" a trapdoor into the system of equations. An accelerated algorithm for MQS gives a way to attack these cryptosystems in a way that is much faster than brute-force search.

To our knowledge, there are no known accelerated algorithms for MQS, though some practical algorithms hold some promise that MQS can be attacked [KS99, CKPS00]. We show that an accelerated algorithm for MQS *does* exist, based on the following plausible conjecture. We define EDGE-WEIGHT k-CLIQUE to be the problem where one is given an edge-weighted undirected graph with weights drawn from a finite field F of $2^{\Theta(b)}$ elements, and is asked if there is a k-clique whose total sum of edge weights (there are $\binom{k}{2}$ of them) is exactly zero over F. Our conjecture is that this problem can be solved faster than brute-force search.

Conjecture 6.6.1 *There is a $\delta \in (0, 1)$ and some $k \geq 3$ such that* EDGE-WEIGHT k-CLIQUE *is in $O(\text{poly}(b) \cdot n^{\delta k})$ time over a field F of $2^{\Theta(b)}$ elements.*

Observe the trivial algorithm can be implemented to run in $O(b \cdot n^k)$ time. A compelling property of the above conjecture is that it is known to hold in some interesting special cases.

- If the weights are restricted to the nodes instead of the edges, then some of our recent work shows that the conjecture indeed holds [VW06]. That is, the NODE-WEIGHT k-CLIQUE problem where the weights are integers in $[-2^b, 2^b]$ can be solved in $O(b \cdot n^{\frac{3+\omega}{2}})$ time, and our method can be adapted to work for finite fields as well.
- If the graph structure is removed from the problem, and we are merely looking for k numbers that sum to zero, each one from three different lists of size n, then the conjecture holds– this is the well-known k-SUM problem from computational geometry [GO95] which can be easily solved in $O(b \cdot n^{\lceil k/2 \rceil})$ time.

If the conjecture is true, then we can use the weighted k-clique algorithm to solve MQS.

Theorem 6.6.1 *Conjecture 6.6.1 implies that* MQS *has a randomized accelerated algorithm, for any finite field F.*

We establish Theorem 6.6.1 in the following paragraphs, showing how to solve MQS instances with m equations (MQS) and n variables over field F in $\mathrm{poly}(m, |F|) \cdot |F|^{n\omega/3}$ time, a significant improvement over brute-force search which would require $\Theta(\mathrm{poly}(m, |F|) \cdot |F|^n)$. The idea is to randomly reduce MQS to the problem of determining whether a sum of degree-two polynomials has a zero solution, then reduce that problem to edge-weighted k-clique. We use a randomization trick that has been employed in other contexts, such as string matching [Kal02] and probabilistically checkable proofs [Sud92].

Definition 6.6.1 *Let K be an extension field of F, and let $r = (r_1, \ldots, r_m) \in K^m$. Define*

$$P_r(x_1, \ldots, x_n) := \sum_{i=1}^{m} r_i \cdot p_i(x_1, \ldots, x_n).$$

Claim 7 *Let K be a (finite) extension field of F, and let $a = (a_1, \ldots, a_n) \in F^n$. Then over the uniform distribution for elements $r \in K^m$:*

- $(\forall i \in [m])[p_i(a_1, \ldots, a_n) = 0] \implies \Pr_{r \in K^m}[P_r(a_1, ..., a_n) = 0] = 1.$
- $(\exists i \in [m])[p_i(a_1, \ldots, a_n) \neq 0] \implies \Pr_{r \in K^m}[P_r(a_1, ..., a_n) \neq 0] = 1 - 1/|K|.$

Proof. First, if $p_i(a_1, \ldots, a_n) = 0$ for all i, then clearly $P_r(a_1, \ldots, a_n) = 0$ for any choice of $r \in K^m$. To prove the contrapositive, we need to show that if there is an $i \in [m]$ such that $p_i(a) \neq 0$, then $\sum_i r_i p_i(a) \neq 0$ with high probability.

For a fixed $i \in [m]$, define $s_i(a) = \sum_{j \,:\, j \neq i} r_j p_j(a)$. We wish to determine the probability that $P_r(a) = r_i p_i(a) + s_i(a) = 0$. For a fixed assignment a, if $p_i(a) \neq 0$, then there is a unique $r_i' \in K$ such that $r_i' = -s_i(a)/p_i(a)$. Thus $P_r(a) = s_i(a) + r_i p_i(a) = 0$ with probability $1/|K|$, when the r_i are chosen uniformly at random. □

Theorem 6.6.2 *Let K be a (finite) extension field of F satisfying $|K| \geq |F|^n \mathrm{poly}(n)$. Then for all variable assignments $a = (a_1, \ldots, a_n) \in F^n$,*

- $(\forall i \in [m])[p_i(a_1, \ldots, a_n) = 0] \implies \Pr_{r \in K^m}[P_r(a_1, ..., a_n) = 0] = 1,$ *and*
- $(\exists i \in [m])[p_i(a_1, \ldots, a_n) \neq 0] \implies \Pr_{r \in K^m}[P_r(a_1, ..., a_n) \neq 0] \leq 1 - 1/(\mathrm{poly}(n)).$

Proof. The first bullet is clearly true. By taking a union bound over all $|F|^n$ possible assignments and invoking the previous claim,

$$\Pr[\text{There exists an assignment } a \text{ s.t. } (\exists i \in [m])[p_i(a) \neq 0] \text{ and } P_r(a) = 0] \leq \frac{|F|^n}{|K|} \leq 1/\mathrm{poly}(n).$$

The theorem follows. □

Therefore, we have randomly reduced the problem of solving an instance of MQS to the problem of finding an assignment $a \in F^n$ satisfying $P_r(a) = \sum_{i=1}^{m} r_i p_i(a) = 0$ over the extension field K. Notice that this problem is similar to an instance of WEIGHTED DEGREE-TWO CSP with $K_p = 0$, $K_v = 0$, the degree-two polynomials $q_i(x) = r_i p_i(x)$, and no node weights– except that the ranges of the polynomials q_i are *not* of poly(n) size. Indeed, the r_i coefficients are from K, which has cardinality at least $|F|^n$poly(n). However, by a construction similar to the proof of Theorem 6.5.2, the problem of satisfying $P_r(a) = 0$ can be reduced to EDGE-WEIGHTED k-CLIQUE, and an $O(\text{poly}(b)n^{\delta k})$ algorithm for EDGE-WEIGHTED k-CLIQUE translates to an $O(\text{poly}(n)d^{\delta n})$ algorithm for MQS. Briefly, the reduction works by

- splitting the set of variables into k parts and listing the $|F|^{n/k}$ partial assignments for each part,
- building a complete k partite graph where the nodes are the partial assignments, and
- putting weights on edges $\{u, v\}$ corresponding to the portion of P_r to which the partial assignments a^u and a^v contribute.

Then one searches for a k-clique with edges that sum to 0 when evaluated over the field K, which we take to have $|F|^n$poly(n) elements.

6.7 Chapter Summary

In this chapter, we gave accelerated algorithms for solving and counting optimal solutions for a large class of NP-complete problems and counting problems, via reductions that exploit fast matrix multiplication. We also showed that a better algorithm for solving a certain edge-weighted k-clique problem would imply a breakthrough in solving multivariate quadratic equations.

An interesting open problem from our work is how one might extend our algorithm scheme to work for k-CSPs, when $k \geq 3$. The most straightforward generalization to k-CSPs results in a reduction to the problem of finding a hyperclique on k nodes in a weighted hypergraph with edges having cardinality up to k. However, there no known accelerated algorithms for finding a k-clique in a hypergraph with k-edges, where $k > 2$ [Yus06]. It is conjectured that matrix multiplication can be done in $O(n^{2+o(1)})$ time, and in our investigation of 3-CSPs, it appears a $2^{3n/4}$ bound might be possible (although at present we do not know how to construct such an algorithm). On the basis of this evidence, we conclude this chapter with an algorithmic conjecture:

Conjecture 6.7.1 *For all $k \geq 2$,* MAX-k-SAT *is in $O^*(2^{n(1-\frac{1}{k+1})})$ time.*

Chapter 7

On Accelerated Algorithms for Satisfiability

In the previous chapter, we discussed a general accelerated algorithm scheme that could be applied to a variety of difficult problems. However, we still do not know of accelerated algorithms for certain key problems that are widely studied in computer science. The most famous of these is the "original" NP-complete problem: the satisfiability problem for Boolean formulas in conjunctive normal form, *i.e.* SAT. Satisfiability is so ubiquitous that an entire conference on theory and applications of the problem is held annually[1]. Recently, a sequence of papers has given algorithms for SAT with $O^*(2^{n-o(n)})$ runtime [Pud98, Sch03, DHW04, DW05a, DW05b, DHW05]. The current best, by Dantsin, Hirsch, and Wolpert [DHW05], is a deterministic algorithm that runs in

$$O^*\left(2^{n\left(1-\frac{1}{\ln(m/n)+O(\ln\ln m)}\right)}\right) \text{ time,}$$

where n is the number of variables, and m is the number of clauses. When the instances are required to have a linear number of clauses (that is, $m \le cn$ for a fixed constant c), Arvind and Schuler [AS03] proved that SAT can be solved in $2^{\delta_c n}$ time for some $\delta_c < 1$; however, δ_c converges to 1 as c goes to infinity. Building on this work, Calabro, Impagliazzo, Paturi [CIP06] have very recently proven an intriguing duality between the solvability of SAT instances with small clause "width" and those with small clause "density":

- there is a constant $\delta_1 < 1$ such that for all $k \ge 3$, k-SAT is in $2^{\delta_1 n}$ time, *if and only if*
- there is a constant $\delta_2 < 1$ such that for all $c \ge 1$, SAT is in $2^{\delta_2 n}$ time on formulas where $m \le cn$.

It does not appear that the current approaches for SAT algorithms will lead directly to an accelerated, $O^*(2^{\delta n})$ time bound, when the instances are allowed to have poly(n) clauses. The

[1] The 'International Conference on Theory and Applications of Satisfiability Testing'. See `http://www.satisfiability.org/`

most related reference is the aforementioned work [CIP06]; a corollary of it is that in order to solve SAT instances with $f(n)$ clauses in $2^{\delta n}$ time, it suffices to find such an accelerated algorithm for $(c \log f(n))$-SAT instances with $f(n)$ clauses for all constants $c \geq 1$. Thus it is known that we may restrict ourselves to looking for a $(c \log n)$-SAT accelerated algorithm.

On the other hand, while many researchers are skeptical that an accelerated algorithm for SAT exists, we have not found much evidence for this skepticism. Results of Impagliazzo and Paturi [IP01] imply that if a $O^*(2^{\delta n})$ algorithm for SAT exists, then a $O^*(2^{\delta(1-\frac{1}{e \cdot k})})$ algorithm for k-SAT exists. But this in itself is not truly evidence; for example, if $\delta = .99$, the implied k-SAT algorithm is only an improvement over the known $O^*((2 - \frac{2}{k+1})^n)$ algorithm when $k > 107$. The lesson here is that a modest improvement over 2^n for SAT would not appear to be earth-shattering.

We have spent a great deal of time attempting to either find an accelerated algorithm for SAT, or give interesting evidence against its possibility. In particular, we seek an algorithm that runs in $\text{poly}(m) \cdot 2^{\delta n}$ time for some $\delta < 1$. We present three algorithmic hypotheses, all of which seem to be reasonable, given the current state of knowledge. Two of the hypotheses exploit the fact that certain important special cases of SAT are solvable in polynomial time. The hypotheses are:

1. There exist $k \geq 3$ and $\varepsilon \in (0, k)$ such that k-DOMINATING SET is in $O(n^{k-\varepsilon})$ time, where n is the number of nodes. (In k-DOMINATING SET, one is given a graph and is asked if there is a set S of k nodes such that every node is either in S or in the neighborhood of S. An $O(n^{k+o(1)})$ algorithm is known, for $k > 7$.)

2. For some $\varepsilon > 0$, 2-SAT+2CLAUSES is in $O((m+n)^{2-\varepsilon})$ time, where m and n are the number of clauses and variables, respectively. (In 2-SAT+2CLAUSES, one is given a 2-CNF formula with two additional clauses of arbitrary length, and is asked if the formula is satisfiable. An $O(mn + n^2)$ algorithm is known.)

3. There is a $k \geq 2$ such that HORN-SAT+kCLAUSES is in $O((n+m)^{k-\varepsilon})$ time. (In the HORN-SAT+kCLAUSES problem, one is given a Horn formula with two additional clauses of arbitrary length, and is asked if the formula is satisfiable. A *Horn formula* has the property that every clause contains at most one positive literal. A trivial $O(n^k(m+n))$ time algorithm is known.)

We prove that if any of the hypotheses are true, then SAT has an accelerated algorithm of the form $\text{poly}(m) \cdot 2^{\delta n}$. Therefore, one should either believe in the existence of an accelerated SAT algorithm, or disbelieve all the hypotheses. While our reductions are all rather simple, they are nevertheless compelling results that connect the solvability of SAT with polynomial time solvable problems. As in the previous chapter, our main approach is the *split-and-list* paradigm: we split up a problem instance into a constant number of subproblems, list the possible solutions for each subproblem, and build some connections between subproblems (in a logical or graphical structure). This maneuver exponentially increases the problem size, but the task of combining subproblems to get a global solution is drastically easier than the original problem. We argue that if a sufficiently good algorithm exists for any of the above three problems, we can use that algorithm to efficiently combine subproblem solutions and solve SAT much faster.

7.1 Good k-Dominating Set Algorithms Imply Accelerated Sat Algorithms

Our first hypothesis concerns the time complexity of k-Dominating Set. In Parameterized Complexity, k-Dominating Set is one of the canonical W[2]-complete problems [DF99]. Given an undirected graph on n nodes and m edges, the task is to find a k-set S of nodes whereby every node of the graph is either in S, or is incident to a node in S. For a long time, the best algorithm known for solving k-Dominating Set was the obvious $O(n^{k+1})$ algorithm that tries all k-sets. Fast matrix multiplication can slightly improve this time bound. The following was recently observed by Eisenbrand and Grandoni [EG04], and independently by the author.

Consider the special case of 2-dominating set. Take the Boolean adjacency matrix A of the graph G, complement it (flip 1's to 0's, and 0's to 1's) and multiply the resulting matrix with its transpose, *i.e.* compute $B = \overline{A} \cdot \overline{A}^T$.

Proposition 7.1.1 *G has a 2-dominating set $\iff$ For some i and j, $B[i,j] = 0$.*

Proof. Let $M[i,:]$ denote the ith row of M and $M[:,j]$ denote the jth column of M. Let $V = [n]$ be the vertices of G. Then

$$\begin{aligned}
\{i,j\} \text{ is a 2-dominating set} \quad &\iff \quad (A \vee I)[i,:] \vee (A \vee I)[j,:] = \mathbf{1}, \text{the all-1's vector} \\
&\iff \quad \langle \overline{(A \vee I)}[i,:], \overline{(A \vee I)}[j,:] \rangle = 0 \\
&\iff \quad \langle \overline{(A \vee I)}[i,:], \overline{(A \vee I)}^T[:,j] \rangle = 0 \\
&\iff \quad B[i,j] = 0.
\end{aligned}$$

□

Therefore, 2-Dominating Set can be solved in $O(n^\omega)$ time, where $\omega < 2.376$ is the matrix multiplication exponent [CW90]. To generalize the algorithm to k-Dominating Set, let $v_1, \ldots, v_n$ be a list of the vertices, and $S_1, \ldots, S_{\binom{n}{k/2}}$ be a list of all $k/2$-sets of the vertices. Define an $\binom{n}{k/2} \times n$ Boolean matrix A_k, where

$$A_k[i,j] = 0 \iff v_j \text{ is dominated by } S_i.$$

Then, the product $B_k = A_k \times A_k^T$ is an $\binom{n}{k/2} \times \binom{n}{k/2}$ matrix, where $B_k[i,j] = 0$ iff $S_i \cup S_j$ is a dominating set.

Proposition 7.1.2 *For $k \geq 7$, k-Dominating Set can be solved in $n^{k+o(1)}$ time.*

Proof. Coppersmith [Cop97] gave an algorithm for multiplying a $n \times n^{.294}$ matrix with a $n^{.294} \times n$ matrix in $n^{2+o(1)}$ ring operations. The product $B_k = A_k \times A_k^T$ is essentially a product of an $N \times N^{2/k}$ matrix with a $N^{2/k} \times N$ matrix, for $N = \binom{n}{k/2}$. But $2/k \leq 0.294$ when $k \geq 7$, so Coppersmith's algorithm can be applied. □

The above method is almost $\Omega(n)$ faster than the trivial algorithm, but still essentially requires examining every possible k-set of vertices. A prominent open problem in parameterized algorithmics is whether or not a time bound even a bit better than n^k is possible for k-Dominating Set. This open problem is our first hypothesis.

Hypothesis 7.1.1 *There is a $k \geq 3$ and $\varepsilon \in (0, k)$ such that k-*Dominating Set *is in $O(n^{k-\varepsilon})$ time.*

We know of no results or conjectures in Parameterized Complexity directly suggesting that Hypothesis 7.1.1 may be false. Surprising algorithms have been found for hard parameterized problems in the past. For example, the W[1]-complete problem k-Clique has an $O(n^{.793k})$ algorithm [FK97, IR78, NP85]. However, if one believes that an improvement for k-Dominating Set is possible, then one must believe that an accelerated algorithm for Sat exists.

Theorem 7.1.1 *Hypothesis 7.1.1 implies that* Sat *has an accelerated algorithm of the form* $\text{poly}(m) \cdot 2^{\delta n}$.

A weaker connection between k-Dominating Set and Sat has been recently established in the literature. We give its formal statement:

Theorem 7.1.2 (Chen *et al.* [CHKX04], Theorem 5.3) *Unless* $\mathsf{FPT} = \mathsf{W[1]}$, *$k$-*Dominating Set *is not in $f(k)n^{o(k)}$ time for any function f.*

It is not necessary to know the meaning of the classes FPT and W[1]; to define them would take us too far afield. What is important to know is the following implication: if $\mathsf{FPT} = \mathsf{W[1]}$ holds, then k-Sat can be solved in $2^{o(n)}$ time. Therefore, Theorem 7.1.2 shows that if k-Dominating Set is solvable in $n^{o(k)}$ time, then k-Sat has a $2^{o(n)}$ time algorithm. However, this result does not say anything *a priori* about the complexity of Sat. The aforementioned result of Calabro, Impagliazzo, and Paturi implies that, if k-Sat has a $2^{o(n)}$ algorithm for all k, then Sat has a $2^{o(n)}$ algorithm when $m \leq cn$, for all constants c.[2] Yet, as far as we know, it is consistent with current knowledge that k-Sat has a $2^{o(n)}$ algorithm for all k, yet Sat does not have an accelerated algorithm on instances with a *polynomial* (or larger) number of clauses.

Theorem 7.1.1 is a special case of the following lemma, which generalizes an NP-hardness reduction from Sat to Dominating Set.

Lemma 7.1.1 *Suppose there is an integer $k \geq 3$ and function f, such that k-*Dominating Set *is solvable in $O(n^{f(k)})$ time. Then* Sat *is in $O\left((m + k2^{\frac{n}{k}})^{f(k)}\right)$ time.*

Proof of Lemma 7.1.1. Fix $k \geq 3$. Let F be a CNF formula with n variables; we build a corresponding graph G_F. Without loss of generality, assume k divides n. Partition the set of its variables into k parts of n/k size each. For each part, make a list of all $2^{n/k}$ partial assignments to variables in that part. Each partial assignment shall correspond to a node in G_F.

[2] More precisely, the implication is that for all $\varepsilon > 0$, there is an algorithm A_ε that solves Sat in $2^{\varepsilon n}$ time.

Make each of the k parts a clique, so there are k disjoint $2^{n/k}$-cliques with $O(2^{2n/k})$ edges. Now add m more nodes, one for each clause, and place an edge from a partial assignment node to a clause node iff the partial assignment satisfies the clause. Finally, for each partial assignment clique, add a dummy node that has edges to all nodes in that clique, but no edges to clause nodes or any other clique.

Consider a k-dominating set S in G_F. Note that no clause node is in S, otherwise some dummy node would not be dominated. Suppose S has two (or more) partial assignment nodes from the same clique. Then there is some clique for which S chose no node; but then S does not dominate its dummy node. Therefore, the collection of partial assignments corresponding to the nodes of S is some satisfying variable assignment, since all clause nodes are dominated.

The total number of nodes is $k2^{n/k} + m + k$, so the lemma follows. $\square$

Proof of Theorem 7.1.1. Let $f(k) = k - \varepsilon$ in Lemma 7.1.1. $\square$

The above result can be rephrased in terms of the k Set Cover problem. Here, one is given a collection $\mathcal{C}$ of n sets over a universe of size m, and the task is to find a $\mathcal{S} \subseteq \mathcal{C}$ so that $|\mathcal{S}| = k$ and every element of the universe is contained in some set of $\mathcal{S}$. By associating the set $S_v = N(v) \cup \{v\}$ with each vertex of a graph $G = (\{v_1, \ldots, v_n\}, E)$, and setting the universe to be $\{v_1, \ldots, v_n\}$, a k set cover for the collection $\{S_{v_1}, \ldots, S_{v_n}\}$ is a k-dominating set for G. Thus an immediate corollary of Theorem 7.1.1 is the following.

Theorem 7.1.3 *If there is $k \geq 2$, k Set Cover can be solved in $O(n^{k-\varepsilon})$ time for a collection of n sets over a universe of size $\mathrm{poly}(\log n)$, then Sat has an accelerated algorithm on instances with $\mathrm{poly}(n)$ clauses.*

7.1.1 A Partial Converse: Using Sat to Solve k-Dominating Set

An intriguing question is whether or not a converse to Theorem 7.1.1 holds. That is, does the existence of a good Sat algorithm imply the existence of a good k-Dominating Set algorithm? One would like a method that encodes any graph into a small CNF formula, whereby an assignment to $k \log n$ variables satisfies the formula iff the graph has a k dominating set. We can prove a partial converse of this type. First we define a variant on Sat:

Problem: Cnf-Sat-S

Input: A pair (F, S), where F is a Boolean CNF formula and S is a subset of F's variables.

Output: An assignment a to the variables of S such that $F[S = a]$ is a satisfiable Horn formula.

That is, in Cnf-Sat-S, we wish to find an assignment to the variables of S so that, after plugging in variables, satisfiability of the remaining formula can be decided in linear time. Cnf-Sat-S is perhaps more difficult than Sat in terms of exact algorithms, in that we are only allowed to set variables within a certain subset (other variables are out of our control), and the assignment we find must not only extend to a satisfying assignment for the formula, but also extend *trivially* to a satisfying assignment. On the other hand, this problem can be solved efficiently when $|S|$ is small: if $|S| = k$, then CNF-SAT-S is in $O((m+n) \cdot 2^k)$ time.[3] We show that a time improvement in solving this problem with respect to $|S|$ implies a better dominating set algorithm, by succinctly encoding the k-dominating set problem in a Boolean formula.

Theorem 7.1.4 *If* CNF-SAT-S *is in* $O(f(m+n) \cdot 2^{\delta|S|})$ *time for some* $\delta \in (0,1)$ *and function* f*, then* k-DOMINATING SET *is in* $O(f(kn^2) \cdot n^{\delta k})$ *time.*

Notice that for any $\delta < 1$ and constant $c > 1$, $n^{c+\delta k} \in O(n^{k-\varepsilon})$ for sufficiently large k and sufficiently small $\varepsilon > 0$. So if f is a polynomial in the above, then the implication is indeed an improved dominating set algorithm for large enough k.

Proof of Theorem 7.1.4. Let $G = (V, E)$ be given and let $n = |V|$. We will set up a formula F_G. Define a set of variables $S = \{x_{1,1}, \ldots, x_{1,\log n}, x_{2,1}, \ldots, x_{2,\log n}, \ldots, x_{k,1}, \ldots, x_{k,\log n}\}$. These variables will represent the binary encoding of a dominating set– specifying an assignment to the $O(k \log n)$ variables of S will be equivalent to specifying a k-set of vertices in G. For $j \in [k]$ and $i \in [n]$, let $v_{j,i}$ be kn variables representing the n vertices in the graph G. Informally, we'll have $v_{j,i} = 1$ if and only if the jth vertex in the candidate dominating set does *not* dominate the ith vertex of G.

The clauses of F_G check that the k-set guessed by S is indeed dominating. Define $x^1_{i,j} := x_{i,j}$, and $x^0_{i,j} := \neg x_{i,j}$. Fix a vertex $u \in V$ in the following. Let $b_1 b_2 \cdots b_{\log n}$ be a binary encoding of u. Define the neighborhood $N(u) := \{v \mid \{u, v\} \in E\}$. Let us index the elements of $V - N(u)$ as

$$V - N(u) = \{u_{i_1}, \ldots, u_{i_{n-\deg(u)}}\}.$$

Then for all $j = [k]$ and $d = 1, \ldots, n - \deg(u)$, add the clause:

$$(x^{1-b_1}_{j,1} \vee \cdots \vee x^{1-b_{\log n}}_{j,\log n} \vee v_{j,i_d})$$

to F_G. Intuitively, this clause says that the i_dth vertex is not dominated by the jth vertex in the candidate dominating set. Note there are $O(kn^2)$ clauses of this kind, one for each possible setting of j, u, and d. For all vertices $i = 1, \ldots, n$, add the clause

$$(\neg v_{1,i} \vee \neg v_{2,i} \vee \cdots \vee \neg v_{k,i})$$

to F_G. These clauses stipulate that at least one of the k vertices in the candidate dominating set must dominate the ith vertex, for all i. This completes the description of F_G.

Observe that once all of the variables in S are set to values (say, an assignment a), all remaining clauses in F_G are either of the form (x) or $(\neg x \vee \neg y \vee \cdots \vee \neg z)$. That is, the remaining formula is Horn, and thus satisfiability for it can be determined in linear time.

We claim that the Horn formula $F_G[S = a]$ is satisfiable if and only if a denotes a dominating set of G. First, since every clause in $F_G[S = a]$ is either a positive literal or a collection of negative literals, observe that $F_G[S = a]$ is unsatisfiable if and only if the clauses $(v_{1,i})$,$(v_{2,i})$, …, $(v_{k,i})$ appear in $F_G[S = a]$, for some $i = 1, \ldots, n$. A clause $(v_{j,i})$ appears iff the literals $x^{1-b_1}_{j,1}$, …,

[3] In the terminology of parameterized complexity theory, one would say that CNF-SAT-S is *fixed parameter tractable* with respect to the parameter S.

$x_{j,\log n}^{1-b_{\log n}}$ are set false and $(x_{j,1}^{1-b_1} \vee \cdots \vee x_{j,\log n}^{1-b_{\log n}} \vee v_{j,i})$ is a clause in F_G. But $x_{j,1}^{1-b_1}, \ldots, x_{j,\log n}^{1-b_{\log n}}$ are false iff the jth vertex in the set S has binary encoding $b_1 b_2 \cdots b_{\log n}$, and the above clause is in F_G iff the vertex with binary encoding $b_1 b_2 \cdots b_{\log n}$ does not have the ith vertex as a neighbor. Therefore the clauses $(v_{1,i})$,$(v_{2,i})$, ..., $(v_{k,i})$ appear in $F_G[S = a]$ iff for all $j = 1, \ldots, n$, the jth vertex in S does not have the ith vertex as a neighbor, *i.e.* the set S is not dominating.

Hence the pair $(F, \{x_{i,j} \mid i \in [k],\ j \in [n]\})$ is an instance of CNF-SAT-S with $|S| = O(k \log n)$ and $|F| = O(kn^2)$. From the above discussion, it follows that a satisfying assignment to S is equivalent to a dominating set in the graph. □

7.2 A Variant of 2-SAT Can Help Solve SAT

2-SAT is the well-studied restriction of SAT to instances with at most two literals per clause, and is known to be solvable in linear time [APT79]. One possible direction for achieving an accelerated algorithm for SAT is to try reducing the problem to 2-SAT in some interesting way. As we do not believe $\mathsf{P} = \mathsf{NP}$, this reduction should be exponential, but hopefully not *terribly* exponential (*e.g.* it might create a formula of $2^{(1-\varepsilon)n}$ total size for some $\varepsilon > 0$). If such a reduction existed, the linear time algorithm for 2-SAT would imply an accelerated SAT algorithm.

The results in this section are inspired by this direction. We present a minor generalization of 2-SAT, which we call 2-SAT+2CLAUSES. This problem admits a straightforward quadratic ($O(mn + n^2)$) time algorithm. We prove that if it has a sub-quadratic time algorithm, then SAT has an accelerated algorithm, via a "mildly exponential" reduction.

Define an instance of 2-SAT+2CLAUSES to be a 2-SAT instance, conjoined with at most two additional clauses of arbitrary length. For example,

$$(\neg x_1 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_5 \vee x_6) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5 \vee \neg x_6)$$

is a 2-SAT+2CLAUSES instance. Such "mixed" instances have been studied in the past, especially in the average-case setting (cf. [MZKBT99]) where one analyzes the solvability of randomly chosen formulas.

Let us first give a simple quadratic time algorithm for solving this problem.

Theorem 7.2.1 2-SAT+2CLAUSES *is in $O(mn+n^2)$ time, where n is the number of variables and m is the total number of clauses.*

Proof. Let F be an instance and C_1, C_2 be its two arbitrary-size clauses. Construct a directed graph G of clauses of $F - C_1 - C_2$, where each node of G is a literal and there is an edge $\ell_i \rightarrow \ell_j$ iff $(\neg \ell_i \vee \ell_j) \in F - C_1 - C_2$.

First, some preprocessing is performed on G. Compute the transitive closure of G in $O(mn+n^2)$ time using standard techniques (cf. [CLRS01], pp.632-633). More precisely, construct a Boolean matrix M where $M[i,j] = 1 \iff \ell_i \rightarrow \ell_j$ in this time, for literals ℓ_i and ℓ_j. If there is a variable x such that $x \rightarrow \neg x$ and $\neg x \rightarrow x$ then return *unsatisfiable.*

Now we run a $O(n^2)$ algorithm that examines the clauses C_1 and C_2, using M. For every pair of literals ℓ_i, ℓ_j from C_1 and C_2 respectively, observe that $(\ell_i \wedge \ell_j)$ is equivalent to $\neg(\neg\ell_i \vee \neg\ell_j) = \neg(\ell_i \rightarrow \neg\ell_j)$. Thus for each such pair, we look up in $O(1)$ time if $(\ell_i \rightarrow \neg\ell_j)$ in M. If the corresponding entry is a 1, then we try the next pair, otherwise return *satisfiable*. If all pairs of literals have been exhausted without satisfaction, return *unsatisfiable*. □

Our second hypothesis is that the time complexity of 2-SAT+2CLAUSES can be slightly improved upon.

Hypothesis 7.2.1 *For some $\varepsilon > 0$, 2-SAT+2CLAUSES is in $O((m+n)^{2-\varepsilon})$ time.*

We have less intuition concerning the truth of Hypothesis 7.2.1, compared to our intuitions about Hypothesis 1. At any rate, we do not know of a good reason to rule it out. If one could show that solving 2-SAT+2CLAUSES requires computing a transitive closure, that would be good evidence against Hypothesis 7.2.1; but this possibility seems unlikely to us. For some evidence that the hypothesis may be true, consider the related problem of 2-SAT+3CLAUSES (with the obvious definition). Using fast matrix multiplication, 2-SAT+3CLAUSES can be solved in $O(mn+n^{\omega})$ time. (Essentially, one computes the transitive closure as before, but in order to determine if there are literals ℓ_i, ℓ_j, ℓ_k from each of the three clauses such that $(\ell_i \wedge \ell_j \wedge \ell_k)$ is consistent with the transitive closure graph, we set up a triangle-finding problem using the transitive closure edges.)

Theorem 7.2.2 *Hypothesis 7.2.1 implies that* SAT *has an accelerated algorithm of the form* $\text{poly}(m) \cdot 2^{\delta n}$.

Proof. We show how to transform a CNF formula F into an (exponentially sized) 2-SAT+2CLAUSES instance F'. In particular, if F has n variables and m clauses, then F' has $O(2^{n/2}+m+n)$ variables and $O(m2^{n/2}+mn)$ clauses. This immediately implies the theorem.

The variables of F' will be of the form x_S, where S is a proposition that is either a conjunction of literals in F, or a disjunction of literals in F. Intuitively, we want x_S to be true if and only if S is true. We therefore have $\neg x_S \iff x_{(\neg S)}$, which we capture by having the clauses

$$(x_S \vee x_{(\neg S)}) \wedge (\neg x_S \vee \neg x_{(\neg S)})$$

for every proposition S in the below.

We split the set of variables arbitrarily into two parts of roughly $n/2$ size each. For both parts, list all the possible $2^{n/2}$ partial assignments P to the variables of that part. We think of each P as a conjunction of $n/2$ literals, so for example the partial assignment

$$y_1 = 1, y_2 = 0, y_3 = 1$$

corresponds to the conjunction

$$y_1 \wedge \neg y_2 \wedge y_3.$$

For each P, make two variables x_P and $x_{(\neg P)}$ in F'. Let $P_1, \ldots, P_{2^{n/2}}$ and $Q_1, \ldots, Q_{2^{n/2}}$ be the partial assignments of the first and second part, respectively. The two arbitrary size clauses in F' are:

$$(x_{P_1} \vee \cdots \vee x_{P_{2^{n/2}}}) \text{ and } (x_{Q_1} \vee \cdots \vee x_{Q_{2^{n/2}}}).$$

Now we show how the remaining clause structure of F can be represented using 2-CNF clauses. For each clause C of F, let C_1 (C_2) be the disjunction of literals in C involving variables from the first (second) part, respectively. Make variables x_{C_1}, $x_{(\neg C_1)}, x_{C_2}$, and $x_{(\neg C_2)}$, and include the clause

$$x_{C_1} \vee x_{C_2}.$$

Finally, we relate the clause variables to the partial assignment variables. For each variable y_i of F, we have the variables x_{y_i} and $x_{\neg y_i}$ in F'. For every C_i of the form $(y_i \vee D)$, F' has the clause

$$x_{(\neg C_i)} \rightarrow x_{\neg y_i}.$$

For every partial assignment P that sets $y_i = 1$, F' has

$$x_P \rightarrow x_{y_i}.$$

Similar clauses are introduced between negative literals $\neg y_i$, and the partial assignments and clauses involving them. For every partial assignment P that falsifies a clause C_i, F' has

$$x_P \rightarrow x_{(\neg C_i)}.$$

We now prove that F' is satisfiable iff F is satisfiable. It is not hard to see that, if F is satisfiable by assignment a, then F' is satisfied by setting $x_S = 1$ if and only if the proposition S is satisfied by a.

The other direction (F' is satisfiable implies F is satisfiable) is a little more involved. First, we claim that if $x_{P_j} = 1$ for an $n/2$-variable partial assignment P_j, then for all $j \neq i$, $x_{P_i} = 0$. Let y be a variable in which P_i and P_j differ in assignment. Without loss of generality, $x_{P_j} \rightarrow x_y$ and $x_{P_i} \rightarrow x_{\neg y} \rightarrow \neg x_y$, therefore only one of x_{P_i} and x_{P_j} can be true for all $i \neq j$.

Suppose there is a satisfying assignment to F', and it sets $x_P = 1$ and $x_Q = 1$, where P (Q) is a partial assignment for the variables in the first (respectively, second) part. We claim that the variable assignment obtained by P and Q satisfies F. For suppose this assignment falsified a clause C, and C_1 (C_2) is the disjunction of literals in C involving variables from the first (respectively, second) part. Then by definition, $x_P \rightarrow x_{(\neg C_1)}$ and $x_Q \rightarrow x_{(\neg C_2)}$. But the satisfying assignment to F' sets $x_P = 1$ and $x_Q = 1$, thus $x_{(\neg C_1)} \wedge x_{(\neg C_2)}$ is satisfied by the assignment, and hence $(\neg x_{C_2}) \wedge (\neg x_{C_2})$ is also satisfied. However, the satisfying assignment to F' also satisfies the clause $(x_{C_1} \vee x_{C_2})$ in F'. This is a contradiction. $\square$

Notice that if we combined the $O(mn + n^2)$ algorithm for 2-Sat+2Clauses and the above reduction, we would only obtain an $O(m2^n)$ algorithm for Sat, which is not much of an improvement over brute force search.

7.3 A Variant on Horn-Sat Can Help Solve Sat

Similar to 2-Sat, the Horn-Sat problem is another restriction of Sat that is known to be solvable in linear time [DG84]. An instance of Horn-Sat is a CNF formula with at most one non-negative literal per clause. Horn-Sat is considered to be a more powerful restriction of Sat than 2-Sat, since Horn-Sat is P-complete and 2-Sat is NL-complete. Somewhat analogous to the previous section, we will show that better algorithms for an extension of Horn-Sat imply better algorithms for Sat. Owing to the power of Horn-Sat, the result we prove here is more general.

For a constant $k \geq 2$, we define Horn-Sat+kClauses to be a CNF of a Horn formula conjoined with k additional disjunctions of arbitrary size. Clearly, an instance of this problem can be solved in $O(n^k \cdot (m+n))$ time, where n is the number of variables and m is the number of clauses.

Hypothesis 7.3.1 *There is a k such that* Horn-Sat+kClauses *is in $O((n+m)^{k-\varepsilon})$ time.*

Theorem 7.3.1 *Hypothesis 7.3.1 implies that* Sat *has an accelerated algorithm of the form* $\text{poly}(m) \cdot 2^{\delta n}$.

We must admit that we are less certain in the truth of Hypothesis 7.3.1, compared to the previous two hypotheses, since the "gap" between the runtime of the best algorithm we know and the time bound we would like is larger than the other two cases. The proof of is similar to the one for 2-Sat+2Clauses, but with a few modifications.

Proof. (Sketch) Let F be a CNF of n variables and m clauses. Divide the set of n variables into k equal-sized parts, and form all possible $2^{n/k}$ assignments for the variables in each part. For each of the $k2^{n/k}$ partial assignments a, we make an "assignment variable" x_a for it in the new formula. For each part $i = 1, \ldots, k$, we make an "assignment clause", which is the disjunction of all assignment variables from part i.

For each literal ℓ in F, we make a "literal variable" x_ℓ in the new formula, and clauses

$$(\neg x_a \vee x_\ell)$$

for each literal ℓ implied by a partial assignment a. Thus an x_a implies exactly n/k variables of type x_ℓ. We also make clauses

$$(\neg x_\ell \vee \neg x_{\neg \ell}),$$

forbidding two opposing literal variables to both be true. (For this reason, at most one assignment variable from an assignment clause can possibly be true.)

For each clause C from F, define C_i to be the restriction of C to variables from part i. Each clause C from F will have $2k$ "clause variables" in the new formula. In particular, for each part, there are two clause variables x_{C_i} and $x_{\neg C_i}$, representing C_i. A clause C in F is represented by the Horn clause

$$(\neg x_{\neg C_1} \vee \cdots \vee \neg x_{\neg C_{k-1}} \vee x_{C_k}).$$

Suppose $C_i = (\ell_1 \vee \cdots \vee \ell_j)$. Then the corresponding Horn clauses are made:

$$(\neg x_{\neg \ell_1} \vee \neg x_{\neg \ell_j} \vee x_{\neg C_i}), (\neg x_{\ell_1} \vee x_{C_i}), \ldots, (\neg x_{\ell_j} \vee x_{C_i}).$$

Finally, we make clauses forbidding both x_{C_i} and $x_{\neg C_i}$:

$$(\neg x_{C_i} \vee \neg x_{\neg C_i}).$$

We claim that this new formula is satisfiable iff F is satisfiable. Namely, the chosen partial assignment variables from each of the k assignment clauses correspond to a satisfying assignment for F. $\square$

7.4 Chapter Summary

In this chapter, we showed how improvements in the runtime of several polynomial time solvable problems would lead to an accelerated algorithm for satisfiability in conjunctive normal form. Our reductions are exponential-sized ones, and therefore somewhat unorthodox, but they build on the split-and-list approach from the previous chapter. To complement these results, it would be interesting if one could find more concrete evidence for the *impossibility* of an improved CNF-SAT algorithm.

Chapter 8

Epilogue: Future Work

In this thesis, we have studied both the limitations and possibilities for computers solving difficult and fundamental computational problems. On the one hand, we found new time lower bounds for solving NP-complete problems on space-bounded random access machines; on the other, we found accelerated algorithms for a large class of NP-complete problems. Below we summarize some specific problems for further work in this area that we believe are interesting.

1. Further Directions for Lower Bounds:

   - Prove that $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[n^{2-\varepsilon}, \log n]$ for all $\varepsilon > 0$, under the random-access model. A weaker goal is to simply improve upon the best time lower bound we have obtained so far. Recall that our best result is that $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTISP}[n^{2\cos(\pi/7)-\varepsilon}, n^{o(1)}]$, and that our automated proof search strongly suggests that an improvement on this lower bound will require truly new ideas and techniques.
   - Our experiments indicate that there a fundamental limitation on what lower bounds can be proved with the "alternation-trading scheme" for indirect diagonalization in the standard sense, and this limitation is related to the amount of speedup involved. However, we do not know yet how to *prove* that this limitation is indeed inevitable, and we cannot rule out the idea of using a new, unforseen time hierarchy theorem to prove better lower bounds by indirect diagonalization. What kinds of lower bounds would *not* be provable using this method, even in the presence of new and improved time hierarchies? General theoretical results along these lines would be very interesting, considering the widespread use of the alternation-trading scheme in proving time lower bounds.
   - Can the alternation-trading arguments be extended to lower bounds for problems not based in nondeterminism? For example, can better time-space lower bounds be found for the MAJORITY SAT problem? (In this problem, one must determine if a given Boolean formula is satisfied by at least 1/2 of its assignments.) Very recently, we have made some progress on this question, proving time-space lower bounds for counting the number of satisfying assignments to a Boolean formula modulo an integer [Wil07].

2. Further Directions for Algorithms:

   - Find an accelerated algorithm for Max Cut and Max 2-Sat that does not require fast matrix multiplication. A more combinatorial approach would be enlightening. Along these lines, it would be interesting to improve the space usage of our algorithm for Weighted 2-CSP. Currently, $O(1.74^n)$ time and $O(2^{2n/3}) \leq O(1.588^n)$ space are required. A very interesting open question is if there is a $O(1.9^n)$ time algorithm for these problems that uses only polynomial space. This question would have a positive answer if one could find an algorithm for solving the k-Clique problem that uses logarithmic space and $n^{k-\delta}$ time for some $\delta > 0$.
   - Generalize the accelerated algorithm for Max 2-Sat to get an accelerated algorithm for Max k-Sat, for any positive integer k. Our current formulation would require one to give a more efficient algorithm for finding a constant-sized hyperclique in a hypergraph. However, no general results are known for this problem, to our knowledge. We conjecture that for all $k \geq 2$, Max k-Sat is in $O^*(2^{n(1-\frac{1}{k+1})})$ time, based on the conjecture that matrix multiplication is in $n^{2+o(1)}$ time.
   - Find a $n^{k-\delta}$ algorithm for k-Dominating Set, for some $k \geq 3$ and some $\delta > 0$. As proved in Chapter 7, this would yield an accelerated algorithm for Sat. Alternatively, find weaker algorithmic hypotheses that would also imply an accelerated Sat algorithm, or give stronger evidence that no such algorithm exists.

Bibliography

[AGN01] J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics* 229:3–27, Elsevier, 2001.

[AS80] D. B. Arnold and M. R. Sleep. Uniform random generation of balanced parenthesis strings. *ACM Transactions on Programming Languages and Systems* 2(1):122–128, 1980.

[AS03] V. Arvind and R. Schuler. The quantum query complexity of 0-1 knapsack and associated claw problems. In *Proceedings of International Symposium on Algorithms and Computation (ISAAC)*, Springer LNCS:168–177, 2003.

[APT79] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters* 8(3):121–123, 1979.

[Ajt02] M. Ajtai. Determinism versus Nondeterminism for Linear Time RAMs with Memory Restrictions. *Journal of Computer and System Sciences* 65:2–37, 2002.

[BNS92] L. Babai, N. Nisan, and M. Szegedy. Multiparty Protocols, Pseudorandom Generators for Logspace, and Time-Space Trade-Offs. *Journal of Computer and System Sciences* 45(2): 204–232, 1992.

[BR99] N. Bansal and V. Raman. Upper bounds for Max-Sat: Further Improved. In *Proceedings of International Symposium on Algorithms and Computation (ISAAC)*, Springer LNCS 1741:247–258, 1999.

[BBS86] L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing* 15:364–383, 1986.

[CIP06] C. Calabro, R. Impagliazzo, and R. Paturi. A Duality between Clause Width and Clause Density for SAT. In *IEEE Conference on Computational Complexity (CCC)*, 252–260, 2006.

[CJW06] J. Carlson, A. Jaffe, and A. Wiles (eds). *The Millennium Prize Problems.* American Mathematical Society, 2006.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM* 28(1):114–133, 1981.

[CK04] J. Chen and I. A. Kanj. Improved exact algorithms for MAX-SAT. *Discrete Applied Mathematics*, 142(1-3), 17–27, 2004.

[CHKX04] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Linear FPT Reductions and Computational Lower Bounds. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, 212–221, 2004.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Edition*, MIT Press, 2001.

[CS76] A. K. Chandra and L. J. Stockmeyer. Alternation. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)* 98–108, 1976.

[Cob66] A. Cobham. The recognition problem for the set of perfect squares. In *IEEE Symposium on Switching and Automata Theory (SWAT)*, 78–87, 1966.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, 151–158, 1971.

[Coo88] S. A. Cook. Short Propositional Formulas Represent Nondeterministic Computations. *Information Processing Letters* 26(5): 269-270 (1988)

[CW90] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. *Journal of Symbolic Computation* 9(3):251–280, 1990.

[Cop97] D. Coppersmith. Rectangular Matrix Multiplication Revisited. *Journal of Complexity* 13:42–49, 1997.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, Springer LNCS 1807, 392–407, 2000.

[CL07] A. Czumaj and A. Lingas. Finding a Heaviest Triangle is not Harder than Matrix Multiplication. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 986–994, 2007.

[DGHK01] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. MAX-SAT approximation beyond the limits of polynomial-time approximation. *Annals of Pure and Applied Logic* 113(1-3): 81–94, 2001.

[DGHKKPRS02] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schoening. A Deterministic $(2 - 2/(k+1))^n$ Algorithm for k-SAT Based on Local Search. *Theoretical Computer Science* 289(1):69–83, 2002.

[DHW04] E. Dantsin, E. A. Hirsch, and A. Wolpert. Algorithms for SAT based on search in Hamming balls. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, Springer LNCS 2996, 141–151, 2004.

[DW05a] E. Dantsin and A. Wolpert. Derandomization of Schuler's Algorithm for SAT. In *Proceedings of Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, Springer LNCS 3542, 80–88, 2005.

[DW05b] E. Dantsin and A. Wolpert. An improved upper bound for SAT. In *Proceedings of Conference on Theory and Applications on Satisfiability Testing (SAT 2005)*, Springer LNCS 3569, 400–407, 2005.

[DHW05] E. Dantsin, E. A. Hirsch, and A. Wolpert. Clause Shortening Combined with Pruning Yields a New Upper Bound for Deterministic SAT Algorithms. *Electronic Colloquium on Computational Complexity*, Report 102, 2005.

[DW06] E. Dantsin and A. Wolpert. MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In *Proceedings of Conference on Theory and Applications of Satisfiability Testing*, Springer LNCS 4121:266–276, 2006.

[Dew81] A. K. Dewdney. *Fast Turing reductions between problems in NP.* University of Western Ontario DCS Technical Reports #68–75, 1981.

[Dew82] A. K. Dewdney. Linear time transformations between combinatorial problems. *International Journal of Computer Mathematics* 11:91–110, 1982.

[DvM06] S. Diehl and D. van Melkebeek. Time-Space Lower Bounds for the Polynomial-Time Hierarchy on Randomized Machines. *SIAM Journal on Computing* 36: 563-594, 2006.

[DG84] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming* 1(3):267–284, 1984.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[EG04] F. Eisenbrand and F. Grandoni. On the Complexity of Fixed Parameter Clique and Dominating Set. *Theoretical Computer Science* 326(1-3):57–67, 2004.

[FK97] U. Feige and J. Kilian. On Limited versus Polynomial Nondeterminism. *Chicago Journal of Theoretical Computer Science*, 1997.

[For97] L. Fortnow. Nondeterministic Polynomial Time Versus Nondeterministic Logarithmic Space: Time-Space Tradeoffs for Satisfiability. In *Proceedings of IEEE Conference on Computational Complexity (CCC)*, 52–60, 1997.

[FvM00] L. Fortnow and D. van Melkebeek. Time-Space Tradeoffs for Nondeterministic Computation. In *Proceedings of IEEE Conference on Computational Complexity (CCC)*, 2–13, 2000.

[FLvMV05] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-Space Lower Bounds for Satisfiability. *Journal of the ACM* 52(6):835–865, 2005.

[GO95] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry* 5:165–185, 1995.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science* 1:237–267, 1976.

[GJ79] M. Garey and D. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W.H. Freeman, San Francisco, 1979.

[GN00] J. Gramm and R. Niedermeier. Faster exact solutions for MAX2SAT. In *Proceedings of Italian Conference on Algorithms and Complexity (CIAC)*, Springer LNCS 1767:174–186, 2000.

[GS89] Y. Gurevich and S. Shelah. Nearly linear time. *Logic at Botik '89*, Springer LNCS 108–118, 1989.

[GHNR03] J. Gramm, E.A. Hirsch, R. Niedermeier and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Discrete Applied Mathematics* 130(2):139–155, 2003.

[Has86] J. Håstad. *Computational limitations for small depth circuits.* PhD Thesis, MIT Press, 1986.

[HS74] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM* 21:277–292, 1974.

[Hir00] E. A. Hirsch. A $2^{m/4}$-time Algorithm for MAX-2-SAT: Corrected Version. *Electronic Colloquium on Computational Complexity* Report TR99-036, 2000.

[Hir03] E. A. Hirsch, Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics* 130(2):173–184, 2003.

[IP01] R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences* 62(2):367–375, 2001.

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences* 63(4): 512–530, 2001.

[IR78] A. Itai and M. Rodeh. Finding a Minimum Circuit in a Graph. *SIAM Journal on Computing* 7(4):413–423, 1978.

[Kal02] A. Kalai. Efficient Pattern-Matching with Don't Cares. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 655–656, 2002.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 85–103, 1972.

[Kan83] R. Kannan. Alternation and the power of nondeterminism. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, 344–346, 1983.

[Kan84] R. Kannan. Towards Separating Nondeterminism from Determinism. *Mathematical Systems Theory* 17(1):29–45, 1984.

[Kar84] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica* 4:373–395, 1984.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademia Nauk SSSR*, 1093–1096, 1979.

[KS99] . A. Kipnis and A. Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *Proceedings of Annual International Cryptology Conference (CRYPTO)* Springer LNCS 1666:19–30, 1999.

[KMRR05] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Algorithms Based on the Treewidth of Sparse Graphs. In *Proceedings of Workshop on Graph Theoretic Concepts in Computer Science (WG)*, Springer LNCS 3787:385–396, 2005.

[KK06] A. Kojevnikov and A. S. Kulikov. A New Approach to Proving Upper Bounds for MAX-2-SAT. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 11–17, 2006.

[KF02] A. S. Kulikov and S. S. Fedin. A $2^{|E|/4}$-time Algorithm for MAX-CUT. *Zapiski nauchnyh seminarov POMI* No.293, 129–138, 2002.

[Lan04] S. Landau. Polynomials in the nations service: using algebra to design the Advanced Encryption Standard. *American Mathematical Monthly* 89–117, February 2004.

[LLZ02] M. Lewin, D. Livnat, and U. Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *Proceedings of Integer Programming and Combinatorial Optimization (IPCO)*, 67–82, 2002.

[LV99] R. J. Lipton and A. Viglas. On the Complexity of SAT. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, 459–464, 1999.

[MR99] M. Mahajan and V. Raman. Parameterizing above Guaranteed Values: MAXSAT and MAXCUT. *Journal of Algorithms* 31(2):335–354, 1999.

[MS87] W. Maass and A. Schorr. Speed-Up of Turing Machines with One Work Tape and a Two-Way Input Tape. *SIAM Journal on Computing* 16(1):195–202, 1987.

[vM04] D. van Melkebeek. Time-Space Lower Bounds for NP-Complete Problems. In G. Paun, G. Rozenberg, and A. Salomaa (eds.), *Current Trends in Theoretical Computer Science* 265–291, World Scientific, 2004.

[vMR05] D. van Melkebeek and R. Raz. A Time Lower Bound for Satisfiability. *Theoretical Computer Science* 348(2-3):311–320, 2005.

[MS72] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of IEEE Symposium on Switching and Automata Theory (SWAT)*, 125–129, 1972.

[MW05] A. Meyerson and R. Williams. On the Complexity of Optimal K-Anonymity. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2004.

[MZKBT99] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. 2+p-SAT: Relation of Typical-Case Complexity to the Nature of the Phase Transition. *Random Structures and Algorithms* 15:414–440, 1999.

[NP85] J. Nesetril and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[NR00] R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms* 26:63–88, 2000.

[NS94] N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity* 4(4):301–313, 1994.

[PR81] W. Paul and R. Reischuk. On time versus space II. *Journal of Computer and System Sciences* 22:312–327, 1981.

[PPST83] W. Paul, N. Pippenger, E. Szemeredi, and W. Trotter. On determinism versus nondeterminism and related problems. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, 429–438, 1983.

[PPSZ05] R. Paturi, P. Pudlak, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *Journal of the ACM* 52(3):337–364, 2005.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pat95] J. Patarin. Cryptoanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, 248–261, 1995.

[Pud98] P. Pudlak. Satisfiability – algorithms and logic. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Springer LNCS 1450, 129–141, 1998.

[RRR98] V. Raman, B. Ravikumar, and S. Srinivasa Rao. A Simplified NP-Complete MAXSAT problem. *Information Processing Letters* 65:1–6, 1998.

[Rob86] M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.

[RN02] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach* (2nd ed.) Prentice Hall, 2002.

[San01] R. Santhanam. Lower bounds on the complexity of recognizing SAT by Turing machines. *Information Processing Letters* 79(5):243–247, 2001.

[Sav70] W. J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences* 4(2):177–192, 1970.

[Sch78] C. Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM* 25(1):136–145, 1978.

[SS81] R. Schroeppel and A. Shamir. A $T{=}O(2^{n/2})$, $S{=}O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM Journal on Computing* 10(3):456–464, 1981.

[Sch99] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 410–414, 1999.

[Sch03] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms* 54(1):40–44, 2005.

[SS03] A. Scott and G. Sorkin. Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In *Proceedings of International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, Springer LNCS 2764:382–395, 2003.

[SS06] A. Scott and G. Sorkin. Solving Sparse Random Instances of Max Cut and Max 2-CSP in Linear Expected Time. *Combinatorics, Probability and Computing* 15:281–315, 2006.

[Str69] V. Strassen. Gaussian Elimination is Not Optimal. *Numerische Mathematik* 13:354–356, 1969.

[Sud92] M. Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximations.* PhD Thesis, University of California at Berkeley, 1992.

[Tou01] I. Tourlakis. Time-Space Tradeoffs for SAT on Nonuniform Machines. *Journal of Computer and System Sciences* 63(2):268–287, 2001.

[Uma01] C. Umans. The Minimum Equivalent DNF Problem and Shortest Implicants. *Journal of Computer and System Sciences* 63(4):597–611, 2001.

[VW06] V. Vassilevska and R. Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, 225–231, 2006.

[Vig02] A. Viglas. *On Hardness and Lower Bounds in Complexity Theory.* PhD Thesis, Princeton University, Technical Report TR-644-02, January 2002.

[WL92] J. Wang and L. Longpre. Nondeterministic and Alternating Computations. *Proceedings of IEEE International Conference on Computing and Information (ICCI)*, 88–91, 1992.

[Wil03] R. Williams. On Computing k-CNF Formula Properties. *Theory and Applications of Satisfiability Testing*, Springer LNCS 2919:330–340, 2004.

[Wil05a] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348(2-3):357–365, 2005.

[Wil05b] R. Williams. Inductive Time-Space Lower Bounds for SAT and Related Problems. *Computational Complexity* 15:433–470, 2007.

[Wil07] R. Williams. Time-Space Tradeoffs for Counting NP Solutions Modulo Integers. In *Proceedings of the IEEE Conference on Computational Complexity (CCC)*, 70–82, 2007.

[Woe03] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization - Eureka! You shrink!*, Springer LNCS 2570:185–207, 2003.

[Woe04] G. J. Woeginger. Space and time complexity of exact algorithms: some open problems. In *Proceedings of the International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, Springer LNCS 3162, 281–290, 2004.

[Yus06] R. Yuster. Finding and counting cliques and independent sets in r-uniform hypergraphs. *Information Processing Letters* 99(4):130–134, 2006.

[Zwi98] U. Zwick, Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 201–210, 1998.