

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

The exponential complexity of satisfiability problems

Permalink

<https://escholarship.org/uc/item/0pk5w64k>

Author

Calabro, Chris

Publication Date

2009

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

The Exponential Complexity of Satisfiability Problems

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Chris Calabro

Committee in charge:

Professor Ramamohan Paturi, Chair
Professor Samuel Buss
Professor Sanjoy Dasgupta
Professor Russell Impagliazzo
Professor David Meyer

2009

Copyright
Chris Calabro, 2009
All rights reserved.

The dissertation of Chris Calabro is approved, and
it is acceptable in quality and form for publication
on microfilm and electronically:

Chair

University of California, San Diego

2009

TABLE OF CONTENTS

Signature Page	iii
Table of Contents	iv
List of Figures	vii
List of Tables	viii
Acknowledgements	ix
Vita and Publications	x
Abstract of the Dissertation	xi
Chapter 1 Introduction	1
Chapter 2 Preliminaries	5
2.1 Mathematical notation	5
2.2 Exponential approximation	7
2.3 Binary entropy function	7
2.4 Chernoff bounds	10
2.4.1 Convex functions	10
2.4.2 General Chernoff bound	11
2.4.3 Large mean case	12
2.4.4 Small mean case	13
2.5 k -wise independent distributions	14
2.6 Switching lemmas	16
2.7 Basic definitions in complexity theory	22
2.7.1 What is exponential complexity?	22
Chapter 3 2-CNF	25
Chapter 4 Resolution	31
4.1 Restricted forms of resolution	34
4.2 Implication trees	35
Chapter 5 Sparsification	38
5.1 Why it's useful	38
5.2 Sparsification algorithm	39
5.3 Basic properties	40
5.4 Choosing the θ_i	44
5.5 Open problems	48
5.6 Rate of growth of the θ_i	48
Chapter 6 k -SAT upper bounds	52
6.1 DPLL method	53
6.2 PPZ algorithm	53
6.2.1 Derandomization of PPZ	57
6.3 PPSZ	58
6.3.1 Derandomization of PPSZ, unique solution case	59
6.4 Local search	61

	6.4.1 Derandomization of LocalSearch	64
6.5	Combine PPSZ with LocalSearch, $k \in \{3, 4\}$	66
6.6	Minor improvements	67
	6.6.1 Deterministic local search, $k = 3$	67
	6.6.2 Improved analysis for Iwama-Tamaki, $k = 3$	68
6.7	Summary and open problems	68
Chapter 7	Quantified k -SAT	71
	7.1 Π_2 3-SAT lower bound	72
	7.2 Algorithms for 2 special cases	74
	7.3 Reduction to a canonical form	76
	7.4 Parameter trade-off at higher levels of the hierarchy	76
Chapter 8	Circuit-SAT upper bound	78
	8.1 Motivation	78
	8.2 The algorithm	79
	8.2.1 Definitions	80
	8.2.2 High level description	80
	8.2.3 Detailed description	80
	8.3 Run time analysis	81
	8.4 Open problems	84
Chapter 9	Unique satisfiability	87
	9.1 Valiant-Vazirani reduction	87
	9.1.1 Hash function representation	91
	9.1.2 Derandomization of the Valiant Vazirani reduction for $2^{o(n)}$ solutions	95
	9.2 Lower bound on the complexity of Unique- k -SAT	97
	9.3 Limitations of oblivious isolation	102
	9.3.1 Conclusions	105
	9.4 Deciding unique satisfiability	106
	9.4.1 Previous work	107
	9.4.2 Efficient reductions	108
	9.4.3 Constraint satisfaction problems	109
	9.4.4 Extending the result up the PH	110
	9.4.5 Solution optimization problems	112
	9.4.6 Constraint optimization problems	114
	9.4.7 Conclusions	116
Chapter 10	Clause width, density duality for SAT	117
	10.1 Definitions	119
	10.2 Previous work	119
	10.3 Relating the SAT constants	122
	10.3.1 Relating k -SAT to SAT_Δ	122
	10.3.2 Relating k -SAT to f -Freq-SAT	123
	10.3.3 Putting the bounds together	124
	10.4 Impossibility of general sparsification	125
	10.5 Conclusion	126
Chapter 11	Summary and open problems	127
	11.1 Impact of quantum computers	128

Bibliography	129
------------------------	-----

LIST OF FIGURES

Figure 2.1: binary entropy function and several approximations	8
Figure 5.1: sparsification algorithm	40
Figure 5.2: randomized sparsification algorithm	48
Figure 6.1: PPZ algorithm solving k -SAT	54
Figure 6.2: Schönning's local search algorithm	61
Figure 6.3: deterministic local search	70
Figure 8.1: linear size, constant depth circuit solver	85
Figure 8.2: linear size, constant depth circuit solver, backtracking version	86
Figure 9.1: generating r	95
Figure 10.1: width reduction algorithm	120

LIST OF TABLES

Table 6.1: performance of k -SAT algorithms	68
---	----

ACKNOWLEDGEMENTS

To my advisor Ramamohan Paturi for the core ideas on sparsification, circuit upper bounds, and for spending far more time on his advisees than is typical for an advisor; to Russell Impagliazzo for the intuitive proof of (2.3), for suggesting how to use the switching lemma, and for philosophical discussions through the years; to Valentine Kabanets for helping to improve my writing style; to the UCSD theory lab denizens – Kirill Levchenko, Sashka Davis, Yi-Kai Liu, Vadim Lyubashevsky, Ragesh Jaiswal, Nathan Segerlind – who have all patiently entertained my mathematical meanderings, even when there was little rational motive; to Julie Conner for guiding me through all administrative matters; to my wife Paola Robotti for being the only one who believed in me; and last to the committee for slugging through this massive document.

The material from chapters 5, 7, 8, 10 is joint work with Russell Impagliazzo and Ramamohan Paturi. That from chapters 5, 10 appeared in the Proceedings of the 21st Annual IEEE Conference on Computational Complexity, 2006. That from chapter 8 will be presented at the Fourth International Workshop on Parameterized and Exact Computation, 2009. That from chapter 7 is unpublished. That from §9.2 and theorem 6.1 is joint work with Russell Impagliazzo, Ramamohan Paturi, and Valentine Kabanets and was published in the Journal of Computer and System Sciences 74(3), 2008. That from §9.3 is unpublished joint work with Russell Impagliazzo. That from §9.4 is joint work with Ramamohan Paturi, was presented at Computer Science in Russia, 2009, and appears in Lecture Notes in Computer Science 5675.

VITA

- 2000 B. S. in Computer Science and Mathematics, Rutgers University
- 2003-2009 Graduate Teaching Assistant, University of California, San Diego
- 2009 Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

- C. Calabro, R. Impagliazzo, and R. Paturi, “The Complexity of Satisfiability of Small Depth Circuits”, *IWPEC*, 2009, (to appear in IWPEC 2009, LNCS).
- C. Calabro and R. Paturi, “ k -SAT is No Harder than Decision-Unique- k -SAT”, *CSR*, 59–70, LNCS 5675, 2009.
- C. Calabro, “A Lower Bound on the Size of Graphs Dense in Long Paths”, *ECCC*, Report TR08-110, 2008.
- C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi, “The Complexity of Unique k -SAT: An Isolation Lemma for k -CNFs”, *JCSS*, 74(3):386–393, 2008.
- C. Calabro, R. Impagliazzo, and R. Paturi, “A Duality between Clause Width and Clause Density for SAT”, *CCC*, 252–260, 2006.

ABSTRACT OF THE DISSERTATION

The Exponential Complexity of Satisfiability Problems

by

Chris Calabro

Doctor of Philosophy in Computer Science

University of California San Diego, 2009

Professor Ramamohan Paturi, Chair

The exponential complexity of a parameterized problem P is the infimum of those c such that P can be solved in time $\text{poly}(|x|)2^{cn}$ on inputs x of parameter n . For example, any (d, k) -constraint satisfaction problem over n variables can be solved in time $\text{poly}(|x|)d^n$, and so has exponential complexity at most $\lg d$. We thoroughly explore the exponential complexity of k -SAT. We

- show that the exponential complexities of k -SAT and SAT with clause density or frequency Δ are approximately the same when $\lg \Delta$ is between $\Omega(k)$ and $O(k \lg k)$.
- upper bound the gap between the exponential complexities of k -SAT and Unique- k -SAT by $O(\frac{\lg^2 k}{k})$, and show that this is nearly optimal for current techniques. For the non-promise version of Unique- k -SAT, we reduce this gap to 0.
- lower bound the exponential complexity of Π_2 3-SAT, a canonical problem at the 2nd level of the polynomial hierarchy, by that of k -SAT, independent of k , showing that a major technical hurdle must be jumped to construct nontrivial algorithms for this problem.
- give what is likely the first nontrivial algorithm for the satisfiability of circuits of constant depth and linear size.

Chapter 1

Introduction

This work will explore the exponential complexity of satisfiability problems, i.e. answering questions of the form $\exists x \phi$ where ϕ is some predicate. Of course, this class is as general as the computable functions, but we will always have some natural parameter n , such as the number of variables or graph nodes in the problem instance, and only consider problems that can be solved in time $\leq \text{poly}(|x|)2^{cn}$ where $c = 1$ and x is the instance. Given a practical problem P where it is not obvious how to make $c = 0$, just how small c can be made becomes important, and roughly this is what we will call the exponential complexity of P . We mostly focus on $P = k\text{-SAT}$, since it is standard, practical, and much can be said about it.

Random $k\text{-SAT}$ with n variables and Δn clauses is known to have a threshold for Δ , namely $2^k \ln 2 - O(k)$ ([AM02, AP03]), below which the formula is satisfiable with high probability and above which the formula is unsatisfiable with high probability. It is also increasingly hard to certify or refute satisfiability as Δ gets closer to the threshold. Thus linear sized $k\text{-CNFs}$ appear to be the hardest, at least in the random input setting. For worst case complexity, it was shown in [IPZ01] that the hardest cases of $k\text{-SAT}$ have size linear in the number of variables. The constant of linearity, however, was doubly exponential in k , which is very far from the threshold for random instances. In chapter 5, we tighten the constant of linearity to exponential in $O(k \lg k)$, which is much closer to the value we would expect.

In chapter 6, we review the currently best worst-case algorithms for $k\text{-SAT}$, deriving or at least explaining their exponential complexity. This serves several purposes. First, we will see how well we can currently do, and what the current best methods are. If one wants to select an algorithm or design the next best one, it helps to have the currently best ones all in one place, explained as a coherent whole, to understand where the bottlenecks are. Indeed, the best 3-SAT algorithm is simply a combination of 2 other very different 3-SAT algorithms.

Next, several times throughout, we will use nontrivial properties of these algorithms which can only be exposed by careful analysis, i.e. it may not suffice to say, “ X ’s algorithm

obviously has property Y ” – X may not have cared about property Y at the time of publication and so it might not be obvious from X ’s original exposition. Indeed, the reader may want to check for properties this author did not think to consider.

Lastly, there will be 2 obvious trends to observe: that progress has significantly slowed, and that all k -SAT algorithms have exponential complexity converging to 1 as $k \rightarrow \infty$. The former is the basis for the Exponential Time Hypothesis (ETH), which conjectures that there is a positive exponential complexity for 3-SAT below which no algorithm can go. The latter is the basis for the Strong Exponential Time Hypothesis (SETH), which conjectures that every sequence of algorithms A_k , the k th of which solving k -SAT, must have exponential complexity converging to 1.

Apart from organizing the information here, the only contribution this author made to the area of k -SAT upper bounds was to show an upper bound on the performance of the PPZ algorithm that decreases as the number of solutions in the problem instance increases, which is in accord with a sort of naive intuition about all satisfiability algorithms, but contradicts the original intuition actually used to analyze PPZ. This should embolden others to prove a similar property about the PPSZ algorithm, which is based on PPZ, since that would dramatically improve the best known upper bound on 3-SAT, rendering the currently best hybrid algorithms obsolete.

In chapter 7, we consider quantified k -SAT. There are currently no nontrivial algorithms for this problem even when restricted to Π_2 3-CNF, i.e. instances of the form $\forall x \exists y \phi$ where $\phi \in 3\text{-CNF}$. We give a weak alibi for this: Π_2 3-SAT is at least as hard as k -SAT independent of k . Thus SETH would preclude ever finding a nontrivial algorithm for this problem. We also show several syntactic restrictions that admit nontrivial algorithms, giving evidence that these are not the hardest and encouraging only focusing on a simple canonical form.

In chapter 8, we consider the satisfiability of circuits of constant depth d and linear size, giving what we believe are the first nontrivial upper bounds. The ideas there are similar in spirit to the switching lemma, but technically different. As should be expected, the bound for moderate d is quite close to 1, so the algorithm is not practical for inverting actual circuits of moderate depth, but it does make the philosophical point that exponential speed ups are possible for inverting functions even in these fairly powerful computational models. Note that most previous research has focused on the *size* of restricted classes of circuits that compute particular functions, whereas we consider the complexity of deciding their satisfiability.

In chapter 9, we explore the relationship between the complexity of the satisfiability problem and that of the unique satisfiability problem for several problems P . There are at least 2 ways of interpreting unique satisfiability. One is Unique- P , the problem of deciding whether the instance has a solution under the promise that there is at most one. The other is Decision-Unique- P , the problem of deciding whether the instance has exactly 1 solution.

Unique- P is the more important problem because the promise may allow the design of better algorithms. For example, as mentioned before, the PPSZ algorithm is able to ex-

exploit uniqueness of a solution to yield better performance. It would be nice to claim that this automatically yields improved performance for the non-unique case as well.

A second example is Grover’s quantum quadratic speedup algorithm. As originally stated in [Gro96], it takes a black box predicate Q with exactly one solution, where Q takes an n -bit input and is evaluable in time t , and finds that solution in time roughly $O(t2^{n/2})$. This was greatly strengthened by [BBHT96] to the following: any probabilistic algorithm running in time t and with success probability p can be transformed into a quantum algorithm with running time $O(t/\sqrt{p})$ and with constant success probability. In other words, [Gro96] and [BBHT96] showed the same result, only [Gro96] required the assumption of a unique solution. The point here is that algorithmic design may benefit by first assuming a unique solution and then removing this assumption later. This is similar to how many algorithms are first designed by using randomness, and later a clever derandomization is found.

The question of whether Unique- P has the same complexity as P for a given problem P is really asking whether the second phase of this algorithmic design process can be automated. Can we *always* remove the assumption of uniqueness without an exponential performance penalty?

The first tool for analyzing this question was the randomized hashing technique of Valiant and Vazirani [VV86]. We analyze this technique in detail, and show how to derandomize it in the case that the input formula has $2^{o(n)}$ solutions. [VV86] showed that if Unique-3-SAT $\in$ RP, then NP = RP. But this is almost entirely a negative result, only showing that Unique-3-SAT is in some sense hard (unless you believe the rather strong hypothesis, that is). It does not provide a means to solve practical problems if we can only solve Unique-3-SAT in exponential time.

We show a somewhat more practical result: that the exponential complexity of k -SAT is not too much larger than that of Unique- k -SAT, where the gap shrinks to 0 as a function of k . This also implies that ETH is equivalent to the assertion that Unique-3-SAT has positive exponential complexity. (Unfortunately, this asymptotic result still does not say much for very small k if ETH is true, but does become significant for moderate k .) We further show that the gap we find is nearly optimal for oblivious hashing techniques, i.e. those that ignore the input formula except to see how many variables it has. Currently there are no other techniques known.

The second version of the uniqueness question, Decision-Unique- P , which is equivalent to asking whether a problem instance defines a unique constant, is of lesser importance, but still may be of practical value, and we will prove much stronger results about it. We will show that the exponential complexity of Decision-Unique- P and P are the same for many standard problems P – including k -SAT, Max- k -SAT, Vertex Cover, Independent Set – both in the randomized and deterministic settings. This resolves an open question of [GB97], and considerably more.

The lack of algorithms for Formula-SAT or even CNF-SAT with exponential complexity < 1 naturally leads us to consider syntactically constrained versions of the problem. The most common constraints considered bound the width of each clause, the number of clauses, or the frequency of each variable (i.e. the number of clauses in which it appears). For each of these

types of constraint, we have nontrivial algorithms, and we now name a few results in this area.

[BKS07] consider 3-SAT with restricted variable frequency. They give a poly time algorithm for 3-SAT where $O(1)$ variables occur 4 times and the rest ≤ 3 times, and show that if all variables occur ≤ 3 times except for one, which we allow to occur an arbitrary number of times, then the complexity jumps to NP-hard. Ek -SAT (the 'E' means each clause has width *exactly* k) restricted to instances where each variable occurs $\leq s$ times is trivial when $s \leq f(k)$, in the sense that such instances are automatically satisfiable, and NP-hard when $s \geq f(k)+1$ for some function $f(k) \in \Theta(2^k/k)$ [KST93, Geb09] a result that is in concert with both the satisfiability threshold for random k -SAT and the Lovász Local Lemma. Interestingly, $f(3) = 3, f(4) = 4, f(5) \in [5, 7]$, but we don't know $f(k)$ for any $k \geq 5$. It is not even known whether f is a computable function [HS06].

[Wah05] gives an algorithm for CNF-SAT when the average variable frequency is $\leq \bar{f}$ with exponential complexity $\leq .17364(\bar{f} - 2)$. [Sch05] gives an algorithm for CNF-SAT when the clause density is $\leq \Delta$ with exponential complexity $\leq 1 - \frac{1}{\lg \Delta}$.

In chapter 10, we show an intuitive functional relationship between the exponential complexities of CNF-SAT with these various restriction types. In particular, assuming ETH, k -SAT and SAT with clause density Δ (or maximum variable frequency Δ or average variable frequency Δ) have almost the same exponential complexity when $\lg \Delta$ is between $\Omega(k)$ and $O(k \lg k)$. So we have an automatic way to make progress in each area if we can make progress in one (modulo some error terms); or, put another way, we are somewhat free to choose the restriction type we feel will lead to the easiest algorithmic design.

Many of the results in all of these areas use the same basic mathematical tools – Chernoff bounds, binary entropy approximations, switching lemmas, k -wise independence – and so we begin in chapter 2 by collecting these tools, and extending several of them in interesting ways. Occasionally, the added precision there over standard statements will come in handy.

Chapter 2

Preliminaries

2.1 Mathematical notation

We will consistently use $\ln, \lg$ to be the natural and base 2 logarithms. The set of natural numbers is $\mathbb{N} = \{0, 1, 2, \dots\}$. For $n \in \mathbb{N}$, $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Interval notation, such as $[a, b]$ may refer to the reals between a and b or the integers between a and b , depending on context. For a finite set S , $x \in_u S$ means that x is chosen uniformly at random from S . If $n \in \mathbb{N}$ is a prime power, $\mathbb{F}_n$ will denote the field of order n . For any field F , $F[x]$ will denote the polynomials in x with coefficients in F .

We will frequently want to discuss the limit of monotone sequences: if $s_1 \leq s_2 \leq \dots$ is a nondecreasing bounded sequence of reals, we let $s_\infty = \lim_{k \rightarrow \infty} s_k$.

In set theory, frequently the natural number $n \in \mathbb{N}$ is defined recursively as $\{0, \dots, n-1\}$, which provides a convenient canonical set of size n . E.g. $2 = \{0, 1\}$. If A, B are sets, A^B is the set of maps from B to A . This is ambiguous if A, B are both natural numbers (e.g. is 2^2 a set of maps or 4?), but this will rarely be a problem given context. If A is a ring and $m, n \in \mathbb{N}$, then $A^{m \times n}$ is the set of $m \times n$ matrices with entries in A . The power set of a set S is $\mathcal{P}(S) = \{T \subseteq S\}$.

When $a_1, \dots, a_k$ are all related to b by the binary relation R , we may write $a_1, \dots, a_k R b$, as in $\forall x, y > 0$; but we will refrain from using $b R a_1, \dots, a_k$ to mean $b R a_1, \dots, b R a_k$ since it is not necessary when the relation reverse to R is available, and since we will want to reserve this notation for making lists that contain both variables and binary relation predicates.

In first order formulas, after function symbols, quantifiers and the unary logical operator $\neg$ have the next highest precedence. The binary logical operators $\wedge, \vee$ have the next highest, and $\rightarrow, \leftrightarrow, \oplus$ have the lowest. All binary logical operators associate from left to right. The universe over which a quantifier is quantifying is often given explicitly, but otherwise is left to be divined from context. A comma separated list after a quantifier indicates that the quantifier is to be distributed among the items of the list. The space after the list indicates where the quantified

subformula begins. The $\exists!$ notation is a minor extension of $\exists$, meaning “there is a unique”. Thus

$$\exists x, y > 0, z A \rightarrow B \rightarrow \neg C \wedge D$$

means “((there is a positive x and positive y and a (not necessarily positive) z such that $A \rightarrow B \rightarrow ((\neg C) \wedge D)$)”.

A predicate in parentheses will denote its indicator function. Thus $(x \in S)$ is 1 if $x \in S$ and 0 otherwise.

The Boolean operators $\neg, \wedge, \vee, \oplus, \rightarrow, \leftrightarrow$ operate on 2, e.g. $1 \wedge 0 = 0$. Occasionally we will spell them out: NOT, AND, OR, XOR, IMPLIES, IFF. We overload them to operate on formulas as well, e.g. if α, β are both formulas, then $\neg(\alpha \wedge \beta)$ is as well. Let V be a set of variables. The set of *Boolean formulas* w.r.t. a basis B of Boolean functions is the closure of V under the Boolean formula building operations corresponding to the functions in B . To get around needing to define the Boolean functions and their corresponding formula building operations, we could model a formula formally as the set of rooted, ordered trees with each internal node labeled with an element of B and each leaf labeled with an element of V . However this is overly pedantic, and when representing formulas by sequences of symbols, we will allow conventional ambiguity, blurring the distinction between a function and its corresponding formula building operator. E.g. $x \wedge y$ may represent a formula or a truth value. If a Boolean function op labeling an internal node v of a formula is symmetric in some k of its arguments, we may consider the corresponding k children of v to be unordered.

Some Boolean operators, such as $\wedge, \vee, \oplus$, are associative and commutative and so, unless specified otherwise, we will not restrict the number of their arguments. Thus, a depth 2 formula over the basis $\{\wedge, \vee, \neg\}$, by default, has unrestricted fan-in.

Circuits are defined the same way as formulas but are represented with directed acyclic graphs instead of trees.

The variables-of map defines a homomorphism from the Boolean formulas to $\mathcal{P}(V)$: $\text{var}(\alpha)$ is the set of variables that appear in α , i.e. if $x \in V$, op is a k -ary Boolean formula building operator, $\alpha_1, \dots, \alpha_k$ are Boolean formulas, then $\text{var}(x) = \{x\}$; $\text{var}(op(\alpha_1, \dots, \alpha_k)) = \bigcup_{i=1}^k \text{var}(\alpha_i)$. Negating a variable x is so common that we will occasionally use the more compact $\bar{x}$ to denote $\neg x$. A variable or its negation is called a *literal*. We will adopt the convention that $\bar{\bar{x}} = x$ so that the set of literals is the closure of V under $\neg$.

An *assignment* to V is a map $a : V \rightarrow 2$. The evaluation-at- a map defines a homomorphism from the Boolean formulas to 2: $a(op(\alpha_1, \dots, \alpha_k)) = op(a(\alpha_1), \dots, a(\alpha_k))$, where $\alpha_1, \dots, \alpha_k$ are Boolean formulas and op is a k -ary formula building operator on the left hand side (LHS) and the corresponding Boolean operator on the right hand side (RHS). We will also write $\alpha(a)$ to mean $a(\alpha)$. If $i \in V, a \in 2^V$, then $a \oplus i$ is the assignment a but with the i th bit

flipped, i.e.

$$(a \oplus i)(j) = \begin{cases} a(j) & \text{if } j \neq i \\ 1 - a(j) & \text{if } j = i \end{cases}.$$

2.2 Exponential approximation

We commonly use that $1 + x \approx e^x$ for x close to 0. Here are more exact statements that will be frequently useful.

Theorem 2.1.

$$\forall x \in \mathbb{R} \quad 1 + x \leq e^x$$

$$\forall x \in [0, 1] \quad 1 + x \geq 2^x$$

$$\forall n \geq 1, x \in [0, 1] \quad \left(1 - \frac{x}{n}\right)^{n-1} \geq e^{-x}.$$

Proof. The 1st statement is shown in most calculus books. To show the 2nd statement, take the $\ln$ of both sides: we want to show $f(x) = \ln(1+x) - x \ln 2 \geq 0$ for $x \in [0, 1]$. $f(0) = f(1) = 0$ and f is concave down on $[0, 1]$ since $f''(x) = -(1+x)^{-2} < 0$ for $x \in [0, 1]$. To show the 3rd statement, take the $\ln$ of both sides: we want to show $g(x) = (n-1) \ln(1 - \frac{x}{n}) + x \geq 0$ for $x \in [0, 1]$. $g(0) = 0$, and $g'(x) = (n-1) \frac{-\frac{1}{n}}{1 - \frac{x}{n}} + 1 = \frac{1-x}{n-x}$ is ≥ 0 when $x \in [0, 1]$. So $\forall x \in [0, 1] \quad g(x) \geq 0$. $\square$

In particular, taking $x = 1$ in theorem 2.1 gives

$$\forall n \geq 1 \quad \left(1 - \frac{1}{n}\right)^{n-1} \geq e^{-1}. \quad (2.1)$$

2.3 Binary entropy function

Let X be a random variable uniform on $[0, 1]$ and $p \in [0, 1]$. Let I be the indicator random variable for the event that $X \in [0, p]$ so that I is Bernoulli with parameter p . The *binary entropy function*¹

$$h(p) = -p \lg p - (1-p) \lg(1-p) \quad (2.2)$$

is the expected number of bits of information about X we learn when we observe the value of I . E.g. if $p = \frac{1}{2}$, then upon observing I , we learn exactly the most significant bit of X , and so $h(p) = 1$. The idea of what a *bit* is needs to be treated more carefully when there are not an integer number of them, but we can take (2.2) as our definition and leave the rest to intuition.

¹Throughout this section, expressions not officially defined on a point of their intended domain will represent their continuous extensions. E.g. $-p \lg p$ will represent 0 at $p = 0$.

We will often need to approximate $h(p)$ by something simpler. The first 2 derivatives of h are

$$h'(p) = \lg(1-p) - \lg p$$

$$h''(p) = -\frac{1}{(\ln 2)p(1-p)}.$$

Following is a plot of h and 3 approximations: the first term $h_1(p) = -p \lg p$ in the definition of h , the quadratic $h_2(p) = 1 - 4(p - \frac{1}{2})^2$ that agrees with h at the points $p = 0, \frac{1}{2}, 1$, and a quadratic $h_3(p) = 1 - \frac{2}{\ln 2}(p - \frac{1}{2})^2$.

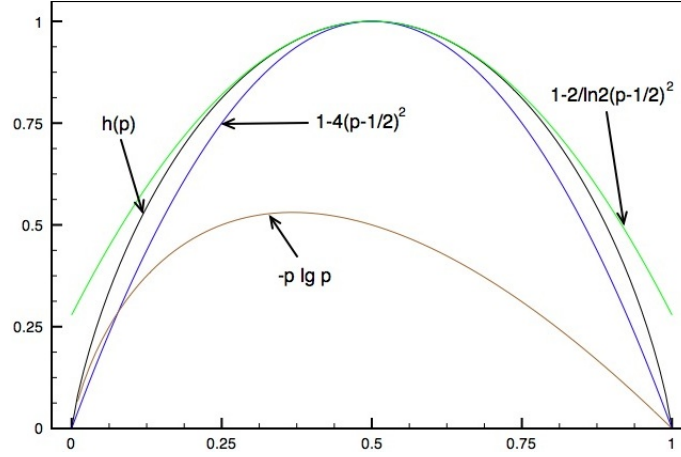


Figure 2.1: binary entropy function and several approximations

h_1, h_2 can be used for lower bounding and h_3 for upper bounding:

$$\forall p \in [0, 1] \quad h_1(p), h_2(p) \leq h(p) \leq h_3(p).$$

Since $\lim_{p \rightarrow 0^+} h'(p) = \infty$, for small values of p , say $p < .01$, h_1 becomes a much better approximation than h_2 . But h_2, h_3 are more useful for p near $\frac{1}{2}$.

To upper bound h for small values of p , first we note that

$$\forall p \in \left[0, \frac{1}{2}\right] \quad -(1-p) \lg(1-p) \leq 2p,$$

which can be seen by differentiating. So

$$\forall p \in \left[0, \frac{1}{2}\right] \quad h(p) \leq -p \lg p + 2p = p \lg \frac{4}{p}.$$

Let $l(p) = p \lg \frac{1}{p}, u(p) = p \lg \frac{4}{p}$ be our lower and upper bounds for $h(p)$ and restrict their domains to $[0, \frac{1}{2}]$ so that we can take inverses. Since both are strictly monotonic, we have $\forall p \in [0, \frac{1}{2}], x \in [0, 1]$,

$$l(p) \leq h(p) \leq u(p)$$

$$u^{-1}(x) \leq h^{-1}(x) \leq l^{-1}(x).$$

We now derive a simple upper bound for l^{-1} and a simple lower bound for u^{-1} . Let $p \in [0, \frac{1}{2}]$, and suppose $p \lg \frac{1}{p} = x$. Then $p \leq x$. So

$$p = \frac{x}{\lg \frac{1}{p}} \leq \frac{x}{\lg \frac{1}{x}}.$$

Now suppose $p \in [0, \frac{1}{2}]$ and $p \lg \frac{4}{p} = x$. Then $x^2 \leq 9p$, which can be seen by differentiating. This implies

$$p = \frac{x}{\lg \frac{4}{p}} \geq \frac{x}{2 \lg \frac{6}{x}}.$$

To summarize,

Theorem 2.2.

$$\begin{aligned} \forall p \in [0, 1] \quad & p \lg \frac{1}{p}, 1 - 4\left(p - \frac{1}{2}\right)^2 \leq h(p) \leq 1 - \frac{2}{\ln 2}\left(p - \frac{1}{2}\right)^2 \\ \forall p \in \left[0, \frac{1}{2}\right] \quad & h(p) \leq p \lg \frac{4}{p} \\ \forall x \in [0, 1] \quad & \frac{x}{2 \lg \frac{6}{x}} \leq h^{-1}(x) \leq \frac{x}{\lg \frac{1}{x}}. \end{aligned}$$

Our primary use for h is to approximate the binomial coefficient $\binom{n}{k}$ when k is $\theta(n)$. Let X be a binomial random variable with parameters (n, p) , where $pn \in \mathbb{Z}$, so that $\mu = \mathbb{E}(X) = pn$, $\sigma^2 = \text{var}(X) = np(1-p)$. Then μ is the most likely value of X . In fact,

$$\Omega\left(\frac{1}{\sigma}\right) \leq \Pr(X = \mu) = \binom{n}{pn} p^{pn} (1-p)^{(1-p)n} = \binom{n}{pn} 2^{-h(p)n} \leq 1,$$

which proves the bound

$$\Omega\left(\frac{1}{\sqrt{n}}\right) 2^{h(p)n} \leq \binom{n}{pn} \leq 2^{h(p)n}. \quad (2.3)$$

A much less intuitive proof of the lower bound - but more compelling if one is more willing to take for granted facts about factorials than about binomial distributions - relies on Stirling's approximation [Wei]: $\forall n \geq 1$,

$$e^{\frac{1}{12n+1}} \sqrt{2\pi n} n^n e^{-n} < n! < e^{\frac{1}{12n}} \sqrt{2\pi n} n^n e^{-n}.$$

For our purposes, we can make due with the coarser approximation that $\forall n \geq 1$,

$$\sqrt{2\pi n} n^n e^{-n} < n! < 1.1 \sqrt{2\pi n} n^n e^{-n}.$$

So $\forall n \geq 1, p \in (0, 1)$ for which $pn \in \mathbb{Z}$,

$$\begin{aligned} \binom{n}{pn} &= \frac{n!}{(pn)!((1-p)n)!} \geq \frac{1}{1.21 \sqrt{2\pi p(1-p)n}} p^{-pn} (1-p)^{-(1-p)n} \\ &\geq \frac{1}{2\sqrt{n}} 2^{h(p)n}, \end{aligned}$$

making use of the fact that $p(1-p) \leq \frac{1}{4}$.

We may also want to bound the size of the Hamming ball $B_{pn}(0^n)$ of radius pn about 0^n . For $p \in [0, \frac{1}{2}]$,

$$\begin{aligned} 1 &\geq \sum_{k=0}^{\lfloor pn \rfloor} \binom{n}{k} p^k (1-p)^{n-k} \quad \text{from the binomial theorem} \\ &\geq \sum_{k=0}^{\lfloor pn \rfloor} \binom{n}{k} p^{pn} (1-p)^{(1-p)n} \quad \text{since } p^k (1-p)^{n-k} \text{ is nonincreasing in } k \\ &= |B_{pn}(0^n)| 2^{-h(p)n}. \end{aligned}$$

For p very close to $\frac{1}{2}$, the simpler bound $|B_{pn}(0^n)| \leq 2^n$ is usually sufficient. Summarizing,

Theorem 2.3. *Let $p \in [0, \frac{1}{2}]$, $n \geq 1$, $p' = \frac{\lfloor pn \rfloor}{n}$ (so $p' = p$ if $pn \in \mathbb{N}$). Then*

$$\frac{1}{2\sqrt{n}} 2^{h(p')n} \leq \binom{n}{p'n} \leq |B_{pn}(0^n)| \leq 2^{h(p)n}.$$

2.4 Chernoff bounds

2.4.1 Convex functions

Jensen's inequality says that if f is convex and you concentrate the probability mass of a random variable X around its mean, then $E(f(X))$ goes down. We want a theorem that goes in the opposite direction: if you spread the probability mass of X out towards some endpoints in such a way that the mean is preserved, then $E(f(X))$ goes up.

Theorem 2.4. *Let $f : [0, 1] \rightarrow \mathbb{R}$ be convex, $X \in [0, 1]$, $Y \in \{0, 1\}$ be random variables, $E(X) = E(Y) = p$. Then $E(f(X)) \leq E(f(Y))$.*

Proof. First assume that f is continuous. Then f is uniformly continuous. Given $\epsilon > 0$, choose n so large that on each interval of size $\leq \frac{1}{n}$, f varies by $\leq \epsilon$. Define the following partition of $[0, 1]$: $I_i = [\frac{i-1}{n}, \frac{i}{n}]$, $i = 1, \dots, n-1$ and $I_n = [1 - \frac{1}{n}, 1]$. Set $p_i = \Pr(X \in I_i)$ and $x_i = E(X \mid X \in I_i)$ (or any value if $p_i = 0$). We approximate X by the discrete distribution Z : $\Pr(Z = x_i) = p_i$. Notice that $E(Z) = p$.

Define new random variables $X_i \in \{0, 1\}$ s.t. $E(X_i) = x_i$. Then

$$\begin{aligned} E(f(Z)) &= \sum p_i f(x_i) = \sum p_i f(E(X_i)) \\ &\leq \sum p_i E(f(X_i)) \quad \text{by Jensen's inequality} \\ &= \sum p_i ((1-x_i)f(0) + x_i f(1)) \\ &= (1-p)f(0) + pf(1) \quad \text{since } \sum p_i x_i = p, \sum p_i = 1 \\ &= E(f(Y)). \end{aligned}$$

So

$$\begin{aligned}
E(f(X)) &= \sum p_i E(f(X) \mid X \in I_i) \\
&\leq \sum p_i (f(x_i) + \epsilon) \\
&= E(f(Z)) + \epsilon \leq E(f(Y)) + \epsilon.
\end{aligned}$$

Since ϵ was arbitrary, we have $E(f(X)) \leq E(f(Y))$.

Now we remove the assumption that f is continuous. Since f is convex, f is bounded on $[0, 1]$ and continuous on $(0, 1)$, $f(0^+) = \lim_{x \rightarrow 0^+} f(x)$ exists and is $\leq f(0)$, $f(1^-) = \lim_{x \rightarrow 1^-} f(x)$ exists and is $\leq f(1)$. We would like to reduce to the case where f is continuous, the problem being that f might have jump discontinuities at $0, 1$. Define

$$f'(x) = \begin{cases} f(x) & \text{if } x \in (0, 1) \\ f(0^+) & \text{if } x = 0 \\ f(1^-) & \text{if } x = 1 \end{cases}.$$

Let X' have the conditional distribution of X given $X \in (0, 1)$ and set $q_0 = \Pr(X = 0)$, $q_1 = \Pr(X = 1)$. Let $Y' \in \{0, 1\}$ with $E(Y') = E(X') = \frac{p - q_1}{1 - q_0 - q_1}$. Notice that f' is continuous and convex. So

$$\begin{aligned}
E(f'(X')) &\leq E(f'(Y')) \\
&= \left(\frac{1 - q_0 - p}{1 - q_0 - q_1} f(0^+) + \frac{p - q_1}{1 - q_0 - q_1} f(1^-) \right) \\
&\leq \left(\frac{1 - q_0 - p}{1 - q_0 - q_1} f(0) + \frac{p - q_1}{1 - q_0 - q_1} f(1) \right).
\end{aligned}$$

So

$$\begin{aligned}
E(f(X)) &= E(f'(X'))(1 - q_0 - q_1) + q_0 f(0) + q_1 f(1) \\
&\leq (1 - q_0 - p)f(0) + (p - q_1)f(1) + q_0 f(0) + q_1 f(1) \\
&= (1 - p)f(0) + pf(1) = E(f(Y)).
\end{aligned}$$

□

2.4.2 General Chernoff bound

Lemma 2.5 (Markov inequality). *Let X be a nonnegative random variable and let $a > 0$. Then $\Pr(X \geq a) \leq \frac{E(X)}{a}$.*

Proof. Define the indicator $I = \begin{cases} 1 & \text{if } X \geq a \\ 0 & \text{else} \end{cases}$. Then $I \leq \frac{X}{a}$, and so

$$\Pr(X \geq a) = E(I) \leq E\left(\frac{X}{a}\right) = \frac{E(X)}{a}.$$

Notice that the inequality holds even if X does not have finite expectation.

□

The family of probabilistic bounds known as *Chernoff bounds* all come from the application of the following simple idea: if X is a random variable and $a, t \in \mathbb{R}, t > 0$, then from the Markov inequality

$$\Pr(X \geq a) = \Pr(e^{tX} \geq e^{ta}) \leq e^{-ta} E(e^{tX}). \quad (2.4)$$

The expectation $E(e^{tX}) : \mathbb{R} \rightarrow \mathbb{R}$ is known as the *moment generating function* of X , since its i^{th} derivative, when evaluated at $t = 0$, is $E(X^i)$, the i^{th} moment of X .

Since (2.4) holds for all $t > 0$, it is advantageous to choose a t that minimizes the right hand side. The best choice will often depend on the distribution of X , and hence there are many different Chernoff bounds.

2.4.3 Large mean case

The following is useful when bounding the probability of large deviation from the mean of a binomial random variable with parameters (n, p) and p is large, say $\geq \frac{2}{3}$, although the hypotheses of the theorem are weaker than this.

Theorem 2.6. *Let $X_1, \dots, X_n$ be a sequence of independent random variables taking values in the interval $[0, 1]$ (X_i is not necessarily Bernoulli). Let $X = \sum X_i, \mu = E(X), a > 0$. Then*

$$\begin{aligned} \Pr(X - \mu \geq a) &\leq e^{-\frac{a^2}{2n}} \\ \Pr(X - \mu \leq -a) &\leq e^{-\frac{a^2}{2n}}. \end{aligned}$$

Proof. Define new random variables $Y_i = X_i - E(X_i)$ so that $E(Y_i) = 0$. Also set $Y = \sum Y_i$. Let $t > 0$. (2.4) gives

$$\Pr(X - \mu \geq a) = \Pr(Y \geq a) \leq e^{-ta} E(e^{tY}) = e^{-ta} \prod_{i=1}^n E(e^{tY_i}), \quad (2.5)$$

the last equality coming from the independence of the Y_i .

To bound $E(e^{tY_i})$, we use a trick. Since $t > 0$, the function $y \mapsto e^{ty}$ is convex. The chord joining the points at $y = \pm 1$ can be expressed as

$$f(y) = \frac{e^t + e^{-t}}{2} + y \frac{e^t - e^{-t}}{2} = \cosh t + y \sinh t,$$

and so for all $y \in [-1, 1]$, $e^{ty} \leq f(y)$. So

$$\begin{aligned} E(e^{tY_i}) &\leq E(\cosh t + Y_i \sinh t) \\ &= \cosh t + E(Y_i) \sinh t = \cosh t \\ &= 1 + \frac{t^2}{2!} + \frac{t^4}{4!} + \dots \\ &\leq 1 + \frac{t^2}{2 \cdot 1!} + \frac{t^4}{2^2 \cdot 2!} + \dots && \text{since } \forall k \geq 0 \quad 2^k \cdot k! \leq (2k)! \\ &= e^{\frac{t^2}{2}}. \end{aligned}$$

Continuing (2.5), we have

$$Pr(X - \mu \geq a) \leq e^{-ta + \frac{t^2 n}{2}}. \quad (2.6)$$

Calculus shows that the exponent is minimized at $t = \frac{a}{n}$. Substituting this value of t back into (2.6) gives

$$Pr(X - \mu \geq a) \leq e^{-\frac{a^2}{n} + \frac{a^2}{2n}} = e^{-\frac{a^2}{2n}}.$$

The second bound is obtained in the same way by analyzing $Pr(-X - (-\mu) \geq a)$. $\square$

2.4.4 Small mean case

The following is more useful when $p < \frac{2}{3}$.

Theorem 2.7. *Let $X_1, \dots, X_n$ be a sequence of independent random variables taking values in the interval $[0, 1]$ (X_i is not necessarily Bernoulli), $X = \sum X_i$, $\mu = E(X)$. Then*

$$\begin{aligned} Pr(X - \mu \geq \delta\mu) &\leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}} \right)^\mu && \text{if } \delta > 0 \\ Pr(X - \mu \leq -\delta\mu) &\leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}} \right)^\mu && \text{if } \delta \in (0, 1) \\ Pr(|X - \mu| \leq \delta\mu) &\leq 2e^{-\frac{\delta^2 \mu}{3}} && \text{if } \delta \in (0, 1). \end{aligned}$$

This last form is useful when the first 2 forms prove unwieldy.

Proof. Let $t > 0$ and define new independent random variables $Y_i \in \{0, 1\}$ with $E(Y_i) = E(X_i) = p_i$. Then by theorem 2.4,

$$\begin{aligned} E(e^{tX_i}) &\leq E(e^{tY_i}) = (1 - p_i) + p_i e^t \\ &= 1 + p_i(e^t - 1) \leq e^{p_i(e^t - 1)}. \end{aligned} \quad (2.7)$$

$$\begin{aligned} Pr(X - \mu \geq \delta\mu) &= Pr(e^{tX} \geq e^{t(1+\delta)\mu}) \\ &\leq e^{-t(1+\delta)\mu} E(e^{tX}) \quad \text{by the Markov inequality} \\ &\leq e^{-t(1+\delta)\mu + (e^t - 1)\mu} \quad \text{by independence and (2.7)} \end{aligned}$$

and substituting $t = \ln(1 + \delta)$ gives the first bound. A similar analysis on $Pr(X - \mu \leq -\delta\mu) = Pr(e^{-tX} \geq e^{-t(1-\delta)\mu})$ and substituting $t = -\ln(1 - \delta)$ gives the second bound. The last bound follows from the first 2 and calculus by showing that $\frac{e^\delta}{(1+\delta)^{1+\delta}}, \frac{e^{-\delta}}{(1-\delta)^{1-\delta}} \leq -\frac{\delta^2}{3}$. $\square$

2.5 k -wise independent distributions

In the design of algorithms, one frequently needs a sequence of random variables $X = (X_1, \dots, X_l)$, which are generated as some function f of a sample space S . An obvious way to remove the randomness completely is to run the algorithm on $X = f(s)$ for every point $s \in S$, which increases the runtime by a factor of $|S|$.

How to combine the answers to get a single answer depends on what kind of problem the original algorithm was solving and its performance guarantees. E.g. if the original algorithm solved a decision problem with 2-sided error $< \frac{1}{2}$, then we can take the majority answer; if the original algorithm had 0 soundness error, then we can take the maximum answer; if the original algorithm was an unbiased estimator of some value, then we can take the average answer.

If the X_i are independent and uniform over some set U of size ≥ 2 , then this technique of iterating over S is usually prohibitive, since $|S| \geq |U|^l$. However, in the analysis of the algorithm, it may become apparent that the X_i need not be fully independent to get the desired result, thus opening up an opportunity to reduce $|S|$.

For example, if we only care that the variance of $\sum_i X_i$ behaves as a binomial random variable, then we can replace X by a pairwise independent distribution, which requires a much smaller sample space. More generally, we may only need that X be *k-wise independent*, meaning that every subset of size $\leq k$ of the X_i is independent. If we also require that each individual X_i is uniform over U , we will say that such a distribution is *k-universal over U* (in the sense that the map $i \mapsto X_i$ is a strongly k -universal hash function).

If, on an input of parameter n , $|S|$ can be reduced to $2^{o(n)}$, then the randomness can be completely removed from the algorithm without increasing the exponential complexity. Since this work is mainly concerned with the exponential complexity of problems, such a reduction in randomness is often good enough for our purposes.

In this section, we show how to generate l random variables with a k -universal distribution over n and give precise bounds on the size of the sample space and the resources needed to generate a sample.

Theorem 2.8. *Let $n, k, l \in \mathbb{N}$ and let $\prod_{i=1}^m p_i^{e_i}$ be the prime factorization of n , so that the p_i are distinct primes and each $e_i \geq 1$. Let $e'_i = \max\{e_i, \lceil \log_{p_i} l \rceil\}$, $n_i = p_i^{e'_i}$, $N = \prod_{i=1}^m n_i$. There is a probability space S of size*

$$N^k \leq (nl^m)^k$$

on which we can define random variables $X_1, \dots, X_l$ that are k -universal over n . To sample requires $O(k \lg N)$ random bits and $O(ml(k + \lg n))$ operations on words of size $O(\lg N)$, where $\lg N \leq O(\lg n + m \lg l)$. If $m = 1$, we only need $O(lk)$ operations. If $m = 1$ and $n \geq l$, then $N = n$.

Proof. Assume $k \leq l$, else we can replace k by l . Also note that $e'_i \leq e_i + \log_{p_i} l$, so $N \leq \prod_i (p_i^{e_i} l) = nl^m$.

First suppose $m = 1, n = p^e, e' = \max\{e, \lceil \log_p l \rceil\}, n' = p^{e'}$. Let

$$S = \{g \in \mathbb{F}_{n'}[x] \mid \deg(g) < k\},$$

and then choose $f \in_u S$. We can think of f as a uniformly random vector in $\mathbb{F}_{n'}^k$, and so our sample space has size n'^k . Let $b : n' \rightarrow \mathbb{F}_{n'}$ be any bijection (Computationally, b will probably be implemented as a simple type-cast and so takes 0 time to compute.). Note that $l \leq n'$. Define random variables

$$X'_i = b^{-1}(f(b(i))), \quad i \in l.$$

The X'_i 's are k -universal over n' because, from Lagrange interpolation, given distinct $x_1, \dots, x_k \in \mathbb{F}_{n'}$ and (not necessarily distinct) $y_1, \dots, y_k \in \mathbb{F}_{n'}$, there is exactly 1 polynomial $g \in \mathbb{F}_{n'}[x]$ of degree $< k$ such that $\forall j \in [k] \ g(x_j) = y_j$, which implies that for distinct $i_1, \dots, i_k \in l$ and (not necessarily distinct) $y_1, \dots, y_k \in n'$,

$$\Pr(\forall j \in [k] \ X'_{i_j} = y_j) = n'^{-k}.$$

Our random variables are then

$$X_i = \left\lfloor \frac{X'_i n}{n'} \right\rfloor,$$

which is uniform over n since $n|n'$.

Now we handle the case where $m > 1$. For each $i \in [m]$ define a sample space S_i as above so that $|S_i| = n_i^k$ and define random variables $Y'_{i,j}$ on S_i so that for each i , $\{Y'_{i,j} \mid j \in l\}$ is k -universal over n_i . Then set $S = \prod_{i=1}^m S_i$, so that $|S| = N^k$, and

$$Y_{i,j} = \left\lfloor \frac{Y'_{i,j}}{p_i^{e'_i - e_i}} \right\rfloor.$$

Then for each i , $\{Y_{i,j} \mid j \in l\}$ is k -universal over $p_i^{e_i}$. For $j \in l$, define X_j to be the unique solution mod n to

$$\begin{aligned} X_j &\equiv Y_{1,j} \pmod{p_1^{e_1}} \\ &\vdots \\ X_j &\equiv Y_{m,j} \pmod{p_m^{e_m}} \end{aligned}$$

guaranteed by the Chinese remainder theorem, which we will abbreviate as $CR(Y_{1,j}, \dots, Y_{m,j})$. CR is bijective and for each $i \in [m]$, $\{Y_{i,j} \mid j \in l\}$, is k -universal, so the X_j are also k -universal.

Now we analyze the time required to sample. All arithmetic operations are on words of size $O(\lg N) = O(\lg n + m \lg l)$ where $N = \prod_i n_i$. To evaluate the m degree $< k$ polynomials

each on l points takes $O(mlk)$ operations. The extended Euclidean algorithm on an input x, y requires $O(\lg x + \lg y)$ operations on words of size $O(\lg x + \lg y)$. CR on a system with moduli $n_1, \dots, n_m$ can be computed by m applications of the extended Euclidean algorithm, the i th call with inputs $n_i, \frac{N}{n_i}$, and so requires $\sum_i O(\lg n_i + \lg \frac{N}{n_i}) \leq O(m \lg N)$ operations. l such calls requires $O(ml \lg N)$ operations. Other parts of the algorithm take less time than the polynomial evaluations and Chinese remaindering. The total is then $O(ml(k + \lg N))$ operations. Note that if $m = 1$, then Chinese remaindering can be avoided, and the complexity is reduced to $O(lk)$ operations. $\square$

If we settle for exact k -wise independence but only approximate uniformity, then we can reduce the size of the sample space even further than in the previous construction and avoid arithmetic operations in fields of order p^e where $e > 1$.

Theorem 2.9. *Let $n, k, l \in \mathbb{N}, \epsilon > 0, N = \max\{\frac{n}{\epsilon}, l\}$ so that $\lg N \leq \lg(\frac{n}{\epsilon} + l)$. There is a probability space S of size $(2N)^k$ on which we can define random variables $X_1, \dots, X_l$ with codomain n that are k -wise independent and with $\forall i \in n, j \in [m]$,*

$$\left| Pr(X_j = i) - \frac{1}{n} \right| \leq \frac{\epsilon}{n}.$$

To sample requires $O(k \lg N)$ random bits and $O(lk)$ operations on words of size $O(\lg N)$.

Proof. Choose a prime $p \in [N, 2N]$, let

$$f \in_u \{g \in \mathbb{F}_p[x] \mid \deg(g) < k\} = S,$$

and for $j \in l$, define $X'_j = f(j)$ and $X_j = \lfloor \frac{X'_j n}{p} \rfloor$. Then the conclusions are immediate. $\square$

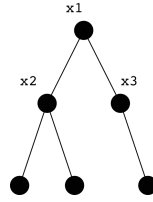
2.6 Switching lemmas

Switching lemmas say that after applying a suitable random restriction to a program in some source computational model, what remains can be represented in some target computational model with high probability. Thus we can *switch* between 2 computational models after applying a suitable restriction.

Håstad's switching lemma [Hås86] allows transformation, with failure probability $\leq (5pk)^{k'}$, from k -CNF to k' -DNF, or k -DNF to k' -CNF, by applying a p -restriction, which leaves each variable free independently with probability p and otherwise sets it uniformly randomly. Because of the desire for generality and the best possible constants, the proof in [Hås86] is complex. Razborov has a short constructive proof, which can be found in [Bea94], allowing transformation, with failure probability $\leq (7pk)^{k'}$, from a k -CNF or k -DNF to a k' -DT (decision tree of depth k' , a generalization of both k' -CNF and k' -DNF), by applying a restriction that chooses a random set of variables of size exactly pn to remain free and uniformly randomly sets the rest.

The main application for these switching lemmas is to provide lower bounds on the size of bounded depth circuits that compute particular functions. For example, note that any restriction of a parity function is again a parity function. By applying a variant of Håstad's switching lemma, the bottom level of a depth d circuit computing parity can be “switched” from AND to OR, or vice versa, allowing collapse of that level w.h.p. while still leaving a linear number of variables free. Repeated application allows collapse into a depth 2 circuit computing parity with a linear number of variables free. Such a circuit must still be large to compute parity, so the original circuit must be large too. [Bop97] uses such an argument to give tight lower bounds on the size of any bounded depth circuit computing a sensitive function.

In this section, we will state several of these switching lemmas precisely. Let V be a set of variables. A *decision tree* is a binary tree T where each internal node is labeled with a variable and the left (right) edges are labeled with 0 (1) and the leaves are labeled 1. The *solutions* $\text{sol}(T)$ of T are the assignments consistent with T ; i.e. those assignments $a \in 2^V$ such that $\exists$ a root-to-leaf path $p = i_1 \cdots i_{r+1}$ where each i_j is a node and $\forall j \in [r]$ $a(\text{label}(i_j)) = \text{label}(\{i_j, i_{j+1}\})$. Let k -DT be the decision trees of depth $\leq k$. E.g.



represents $\text{sol}(\bar{x}_1\bar{x}_2 + \bar{x}_1x_2 + x_1\bar{x}_3)$, where the left (right) edges are implicitly labeled 0 (1). The key property of decision trees is that a tree of depth k can be represented by either a k -CNF or a k -DNF, as we show below.

Letting $f : 2^n \rightarrow 2$ be a Boolean function, the *CNF*, *DNF*, and *DT complexities* of f are

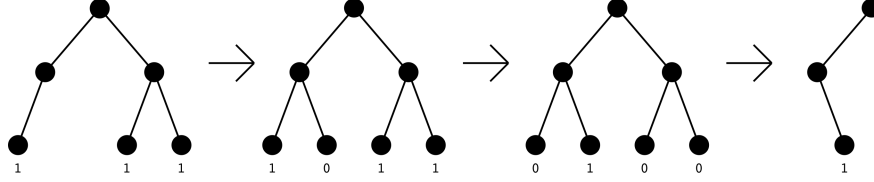
$$\text{CNF}(f) = \min\{k \in \mathbb{N} \mid \exists F \in k\text{-CNF } f^{-1}(1) = \text{sol}(F)\}$$

$$\text{DNF}(f) = \min\{k \in \mathbb{N} \mid \exists G \in k\text{-DNF } f^{-1}(1) = \text{sol}(G)\}$$

$$\text{DT}(f) = \min\{k \in \mathbb{N} \mid \exists H \in k\text{-DT } f^{-1}(1) = \text{sol}(H)\}.$$

Lemma 2.10. *Let $f : 2^V \rightarrow 2$ be a Boolean function. Then $\text{DT}(f) = \text{DT}(\bar{f})$ and $\text{CNF}(f) = \text{DNF}(\bar{f})$.*

Proof. Let T be a decision tree for f . Add leaves labeled with 0 for non-solutions. Then interchange 0's and 1's on the leaf labels. Then recursively trim 0 leaves. What remains is a decision tree for $\bar{f}$. E.g.



For the second statement, note that any $F \in k\text{-CNF}$ and $G = \text{dual}(F) \in k\text{-DNF}$ (obtained by interchanging $\wedge, \vee$ and negating all literals) satisfy $\text{sol}(F) = \overline{\text{sol}(G)}$. $\square$

Lemma 2.11. *If $f \in 2^V \rightarrow 2$, then*

$$\text{DT}(f) \geq \max\{\text{CNF}(f), \text{DNF}(f)\}.$$

Proof. To show $\text{DT}(f) \geq \text{DNF}(f)$, let $T \in k\text{-DT}$. We can construct an $F \in k\text{-CNF}$ s.t. $\text{sol}(F) = \text{sol}(T)$ by having a k' -term for each leaf of T at depth k' . To show $\text{DT}(f) \geq \text{CNF}(f)$, use lemma 2.10:

$$\text{DT}(f) = \text{DT}(\bar{f}) \geq \text{DNF}(\bar{f}) = \text{CNF}(f).$$

$\square$

A *restriction* of a variable set V is a map $\rho : V \rightarrow \{0, 1, *\}$. If $a \in 2^V, S \subseteq V$, define

$$\rho_{a,S} = \begin{cases} * & \text{if } i \in S \\ a(i) & \text{if } i \notin S \end{cases}.$$

The restriction of Boolean function $f : 2^V \rightarrow 2$ to ρ is $(f \upharpoonright \rho) : \rho^{-1}(*) \rightarrow 2$ defined by $(f \upharpoonright \rho)(a) = f(b)$ where $b(i)$ is $a(i)$ if $i \in \text{dom}(a)$ and $\rho(i)$ otherwise.

Say $\rho_{a,S}$ has distribution $\mathcal{D}_{V,p}$ iff $a \in {}_u 2^V$ and $S \in {}_u \{S' \subseteq V \mid |S'| = p|V|\}$. Say $\rho_{a,S}$ has distribution $\mathcal{D}'_{V,p}$ iff $a \in {}_u 2^V$ and each $i \in V$ is in S independently with probability p .

Theorem 2.12 (Razborov's switching lemma). $\forall G \in k\text{-DNF}, V = \text{var}(G), k' \geq 0, p \in [0, 1]$ s.t. $p|V| \in \mathbb{N}$,

$$\Pr_{\rho \in \mathcal{D}_{V,p}}(\text{DT}(G \upharpoonright \rho) \geq k') \leq (7pk)^{k'}.$$

Theorem 2.13 (Håstad's switching lemma). $\forall F \in k\text{-CNF}, V = \text{var}(G), k' \geq 0, p \in (0, 1]$

$$\Pr_{\rho \in \mathcal{D}'_{V,p}}(\text{DNF}(F \upharpoonright \rho) \geq k') \leq \alpha^{k'}$$

where α is the unique positive solution to

$$\left(1 + \frac{4p}{1+p} \frac{1}{\alpha}\right)^k = \left(1 + \frac{2p}{1+p} \frac{1}{\alpha}\right)^k + 1. \quad (2.8)$$

Håstad pointed out that $\alpha \leq 5pk$ if $p \in o(1)$. Actually, we do not need this assumption.

Lemma 2.14. *In (2.8), $\alpha \leq 5pk$.*

Proof. Let $f_p(x) = (1 + \frac{4p}{1+p} \frac{1}{x})^k$, $g_p(x) = (1 + \frac{2p}{1+p} \frac{1}{x})^k + 1$. The derivative of $f_p - g_p$ is < 0 when $x, p > 0$. Also, $\lim_{x \rightarrow 0^+} (f_p - g_p)(x) = \infty$, so indeed $f_p - g_p$ has at most 1 root α . We can upper bound α by $\beta = 5pk$ by showing that $(f_p - g_p)(\beta) < 0$. Let

$$\begin{aligned} F(p) &= f_p(\beta) = \left(1 + \frac{4}{5} \frac{1}{1+p} \frac{1}{k}\right)^k \\ G(p) &= g_p(\beta) = \left(1 + \frac{2}{5} \frac{1}{1+p} \frac{1}{k}\right)^k + 1. \end{aligned}$$

The derivative of $F - G$ is < 0 . From theorem 2.1, $F(0) \leq e^{\frac{4}{5}}$ and $G(0) \geq 2^{\frac{2}{5}} + 1$. So $(F - G)(0) < 0$, which implies $\forall p \in (0, 1]$ $(f_p - g_p)(\beta) = (F - G)(p) < 0$. $\square$

As long as we only want to transform between k -CNF and k' -DNF, theorem 2.13 is at least as strong as 2.12, mainly because the distributions $\mathcal{D}_{V,p}, \mathcal{D}'_{V,p}$ are so similar. We can make this more formal.

Theorem 2.15. $\forall G \in k\text{-DNF}, V = \text{var}(G), k' \geq 0, p \in [0, 1] \text{ s.t. } p|V| \in \mathbb{N}$,

$$Pr_{\rho \in \mathcal{D}_{V,p}}(\text{CNF}(G \upharpoonright \rho) \geq k') \leq 2Pr_{\rho \in \mathcal{D}'_{V,p}}(\text{CNF}(G \upharpoonright \rho) \geq k').$$

Proof. First note that for any nonnegative, nondecreasing function f , random variable X , and $x \in \mathbb{R}$, we have from Markov's inequality (lemma 2.5)

$$f(x)Pr(X \geq x) \leq f(x)Pr(f(X) \geq f(x)) \leq E(f(X)). \quad (2.9)$$

Let $n = |V|$ and apply (2.9) with $f(y) = Pr_{\rho \in \mathcal{D}_{V, \frac{p}{n}}}(\text{CNF}(G \upharpoonright \rho) \geq k')$, X a binomial random variable with parameters (n, p) , and $x = np$. We claim that $x \in \mathbb{N}$ is sufficient to conclude that $Pr(X \geq x) \geq \frac{1}{2}$, which ought to be at least plausible from the central limit theorem. (Surprisingly, a complete proof of this simple fact did not surface until 1970. This fact and a thorough treatment of the relationship between the mean, median, and mode of binomial random variables can be found in [RK80].) So we have

$$\begin{aligned} &Pr_{\rho \in \mathcal{D}_{V,p}}(\text{CNF}(G \upharpoonright \rho) \geq k') \\ &= f(x) \leq \frac{E(f(X))}{Pr(X \geq x)} \leq 2E(f(X)) \\ &= 2Pr_{\rho \in \mathcal{D}'_{V,p}}(\text{CNF}(G \upharpoonright \rho) \geq k'). \end{aligned}$$

$\square$

It's not as clear how to use Razborov's switching lemma to upper bound the switching failure probability in Håstad's switching lemma. The problem is that the number of free variables in the restriction $\rho \in \mathcal{D}'_{V,p}$ is binomial, and has a tail that's just a bit too heavy. However, if $k' \leq O(pn)$, we get something.

Theorem 2.16. $\forall G \in k\text{-DNF}, V = \text{var}(G), n = |V|, k' \geq 0, \alpha > 0, p \in [0, 1] \text{ s.t. } \alpha pn \in \mathbb{N},$

$$\Pr_{\rho \in \mathcal{D}'_{V,p}}(\text{CNF}(G \upharpoonright \rho) \geq k') \leq \Pr_{\rho \in \mathcal{D}_{V,\alpha p}}(\text{CNF}(G \upharpoonright \rho) \geq k') + \left(\frac{e^{\alpha-1}}{\alpha^\alpha}\right)^{pn}.$$

Proof. The first term on the RHS is an upper bound on the LHS conditioned on the event that $\rho \in \mathcal{D}'_{V,p}$ leaves $\leq \alpha pn$ variables free. The second term on the RHS is an upper bound from the Chernoff bound (theorem 2.7) on the probability that ρ leaves $\geq \alpha pn$ variables free. $\square$

Lemma 2.17. *We can interchange DNF, CNF in each of theorems 2.12, 2.13, 2.15, 2.16.*

Proof. For theorem 2.12, let $F \in k\text{-CNF}, V = \text{var}(F), k' \geq 0, p \in [0, 1], \rho \in \mathcal{D}_{V,p}, G = \text{dual}(F)$. Then $\text{sol}(F \upharpoonright \rho) = (\text{sol}(F)) \upharpoonright \rho = \overline{(\text{sol}(G))} \upharpoonright \rho = \overline{\text{sol}(G \upharpoonright \rho)}$. From theorem 2.12,

$$\Pr_{\rho}(\text{DT}(F \upharpoonright \rho) \geq k') = \Pr_{\rho}(\text{DT}(G \upharpoonright \rho) \geq k') \leq (7pk)^{k'}.$$

The argument for the other theorems is similar. $\square$

Below we show that theorem 2.13 is tight up to the constant in the parentheses, at least, for any fixed $k \geq 2, p \in (0, \frac{1}{4k}]$.

Theorem 2.18. $\forall k \geq 2, p \in (0, \frac{1}{4k}] \exists n_0 \forall n \geq n_0 \exists G \in k\text{-DNF}, k' \geq 0 \text{ with } V = \text{var}(G), |V| = n \text{ s.t.}$

$$\Pr_{\rho \in \mathcal{D}'_{V,p}}(\text{CNF}(G \upharpoonright \rho) \geq k') \geq \left(\frac{1}{16}pk\right)^{k'}.$$

Proof. We claim that for any monotone DNF G consisting of terms with disjoint sets of variables,

$$\text{CNF}(G) = \begin{cases} 0 & \text{if } 1 \in G \\ |G - \{0\}| & \text{if } 1 \notin G \end{cases}. \quad (2.10)$$

To see this, if $1 \in G$, then G is a constant, and so can be represented by the empty CNF. Otherwise, if $1 \notin G$, then $\text{CNF}(G) \leq t = |G - \{0\}|$ since $F = \prod(G - \{0\})$ is a suitable CNF (here the product is the set of all tuples that select one literal from each term of $G - \{0\}$). Conversely, let F be a CNF representing G and suppose indirectly that there is a clause $C \in F$ with $|C| < t$. Then by the pigeonhole principle, there is a term $T \in G$ with $\text{var}(T) \cap \text{var}(C) = \emptyset$. Let $a \in \text{sol}(T)$ and $b = (a \text{ but with the literals of } C \text{ that are true at } a \text{ flipped})$. Then $b \in \text{sol}(T) \subseteq \text{sol}(G) \subseteq \text{sol}(C)$, a contradiction. So every clause of F has size $\geq t$. If F has at least one clause, we are done. The form of G prevents it from being a tautology, so F has at least one clause.

For n' specified later, let V be $n = n'k$ variables and $T_1, \dots, T_{n'}$ be a partition of V into n' blocks each of size k . Let G be the monotone k -DNF consisting of terms $T_1, \dots, T_{n'}$. Let

$\rho \in \mathcal{D}'_{V,p}$. For any k -term T ,

$$\begin{aligned}
Pr(T \upharpoonright \rho \neq 0) &= \left(p + (1-p)\frac{1}{2}\right)^k = \left(\frac{1+p}{2}\right)^k \\
Pr(T \upharpoonright \rho = 1) &= \left(\frac{1-p}{2}\right)^k \\
p' &\stackrel{\text{def}}{=} Pr(T \upharpoonright \rho \neq 0 \mid T \upharpoonright \rho \neq 1) \\
&= \frac{Pr(T \upharpoonright \rho \neq 0) - Pr(T \upharpoonright \rho = 1)}{Pr(T \upharpoonright \rho \neq 1)} \\
&= \frac{\left(\frac{1+p}{2}\right)^k - \left(\frac{1-p}{2}\right)^k}{1 - \left(\frac{1-p}{2}\right)^k}.
\end{aligned}$$

$p \leq \frac{1}{k}$ implies that $(1+p)^k \in [1+kp, 1+2kp]$ and $(1-p)^k \in [1-kp, 1]$. So $p' \geq \frac{kp}{2^k}$ and $p' \leq \frac{3kp}{2^{k-1}} \leq \frac{4kp}{2^k}$, since $k \geq 2$. So $p' \in [\frac{kp}{2^k}, \frac{4kp}{2^k}]$. From (2.10), given that $1 \notin G \upharpoonright \rho$, $\text{CNF}(G \upharpoonright \rho)$ is $X = |\{T \in G \upharpoonright \rho \mid T \neq 0\}|$, and this is binomial with parameters (n', p') . So

$$\begin{aligned}
Pr(\text{CNF}(G \upharpoonright \rho) \geq k') &= Pr(X \geq k')Pr(1 \notin G \upharpoonright \rho) \\
&= Pr(X \geq k')\left(1 - \left(\frac{1-p}{2}\right)^k\right)^{n'} \\
&\geq Pr(X = k')(1 - 2^{-k})^{n'} \\
&= \binom{n'}{k'} p'^{k'} (1-p')^{n'-k'} (1 - 2^{-k})^{n'}
\end{aligned} \tag{2.11}$$

Choose $k' = \frac{n'}{2^k}$. Then the first 2 factors in (2.11) are

$$\begin{aligned}
\binom{n'}{k'} p'^{k'} &\geq \frac{1}{2\sqrt{n'}} 2^{h(2^{-k})n'} p'^{k'} \quad \text{using theorem 2.3} \\
&\geq \frac{1}{2\sqrt{n'}} 2^{kk'} \left(\frac{kp}{2^k}\right)^{k'} \quad \text{using theorem 2.2} \\
&\geq \frac{1}{2\sqrt{n'}} (kp)^{k'}.
\end{aligned} \tag{2.12}$$

Note that $1 - p' \geq 1 - \frac{4kp}{2^k} \geq 1 - 2^{-k}$. So the last 2 factors in (2.11) are

$$\begin{aligned}
&\geq (1 - 2^{-k})^{2n'-k'} \\
&= (1 - 2^{-k})^{(2^k-1)\frac{2^k}{2^k-1} \frac{2n'-k'}{2^k}} \\
&\geq e^{-\frac{2^k}{2^k-1}(2k' - \frac{k'}{2^k})} \quad \text{using theorem 2.1} \\
&\geq e^{-\frac{4}{3}(2k' - \frac{k'}{2^k})} \quad \text{since } k \geq 2 \\
&\geq 2^{-4k'} \omega(\sqrt{n'}).
\end{aligned} \tag{2.13}$$

Combining (2.12), (2.13), we get that for sufficiently large n' ,

$$Pr(\text{CNF}(G \upharpoonright \rho) \geq k') \geq \left(\frac{1}{16}pk\right)^{k'}.$$

□

2.7 Basic definitions in complexity theory

The *relative error* of an approximation x to a value $y \in \mathbb{R}$ is

$$\text{relererror}(x, y) = \begin{cases} \left| \frac{x-y}{y} \right| & \text{if } y \neq 0 \\ 0 & \text{if } x = y = 0 \\ \infty & \text{if } x \neq y = 0 \end{cases}.$$

A *Fully Polynomial Randomized Approximation Scheme* (FPRAS) for a function $f : \Sigma^* \rightarrow \mathbb{R}$ is a randomized algorithm A such that $\forall \epsilon > 0, x \in \Sigma^*$

$$\Pr(\text{relererror}(A(\epsilon, x), f(x)) > \epsilon) \leq \frac{1}{4}$$

and A runs in time $\text{poly}(|x|, \frac{1}{\epsilon})$.

2.7.1 What is exponential complexity?

If a parameterized problem P with parameter n can be solved in time $t = 2^{cn}$, then a reasonable question is how small c can be made, which we may call the *exponential complexity* c of P (to be defined more precisely later), but we must first ask what it means to be able to solve a problem in time t : in what computational model? The strong Church-Turing thesis says that every reasonable model of computation is polynomially simulable by any other, meaning that for any 2 ‘reasonable’ models A, B , there is a k so that t steps in A can be simulated by $O(t^k)$ steps in B . But this only helps to define c up to its big-Theta class, which isn’t very helpful when c is already a constant.

For explicit upper bound results, this is an unimportant point, since a statement such as “ k -SAT can be solved in time 2^{41n} ” means that it can be done in so much time on some modern computer – if you want to solve it in that much time, at worst you need to find that computer, and probably you have one sitting on your desk right now. But for lower bound or implicit upper bound results, we should be a little more careful.

One reason this is rarely addressed is that there are no strong lower bounds and perhaps we will only cross that bridge when we come to it! Another is that the polynomial simulability between reasonable models is mostly a theoretical curiosity that applies mostly to simulating k -tape Turing machines by 1-tape Turing machines, and similar such models that are unpopular on the consumer market (my last laptop certainly wasn’t a k -tape Turing machine). Experience has shown us that simulability between popular machines is actually *very* efficient. Modern computers easily simulate many older ones with essentially only a constant factor penalty in terms of number of clock cycles, and even that is more than negated by increased cycle frequency. One clock cycle of a new microprocessor instruction can nearly always be simulated by a constant number of instructions in the old set, and one extra register can be simulated by loading and storing to external memory.

Thus, a reasonable way out is to fix the computational model to something similar to modern computers, e.g. a RAM machine, and define c in terms of that.

One may worry that this may reduce the applicability of results, since, unlikely though it may seem now, if another, polynomially more efficient computational model becomes popular, then one would either have to reprove old results, or use general reductions that simulate one machine by another, which may weaken the results substantially. We will give a few reasons for why this should not be a big concern.

One is that current results only (implicitly) assume that the computational model is *at least* as strong as this minimal model, only assuming things like function call can be done in time linear in the input size or that arithmetic on words of size $\lg$ of the input size can be done in constant time. This is reasonable since future computers, even if polynomially faster on some computation, are unlikely to be polynomially *slower* on some other computation used in a proof, since that may render legacy code impractical, making such new machines hard to sell.

A second reason is that the RAM model seems to capture the energy requirements of a computation for realistically-sized inputs. It's hard to imagine a physical machine that uses polynomially less than n^2 energy for matrix-vector multiply. (Although quantum computation challenges such statements for some specialized computations. We will have more to say on that later.)

Our last justification for fixing a computational model comes from the fact that many current exponential time algorithms are exponential iterations of poly time algorithms. The exponential complexity of such an algorithm is undisturbed by a change in machine model, provided the target machine is still powerful enough to count in a loop efficiently.

From now on, all computations are assumed to take place in a randomized RAM model: with $O(1)$ registers and instructions load/store immediate/memory/random bit, add/subtract, bit shift immediate, jump if zero. Deterministic computations simply don't get access to the load-random-bit instruction. With our model in hand, we now make some definitions.

Let Σ be a finite alphabet. A *language* over Σ is a set $L \subseteq \Sigma^*$. An algorithm A *solves* instance $x \in \Sigma^*$ of L iff

$$A(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}.$$

A *promise problem* is a pair $P = (L, I)$ where $L, I \subseteq \Sigma^*$ represent the language and promised instances of the problem. $x \in \Sigma^*$ *meets the promise* of P iff $x \in I$. An algorithm A *solves* instance $x \in \Sigma^*$ of P iff

$$A(x) = \begin{cases} 1 & \text{if } x \in I \cap L \\ 0 & \text{if } x \in I - L \end{cases}.$$

Notice that we still say A solves x if $x \notin I$. We could also equivalently model a promise problem

by a pair (Y, N) representing the *yes, no* instances of the problem, but we take this approach to name the promise explicitly. P is equivalent to the language L when $I = \Sigma^*$.

A *parameterized problem* P includes a collection of metrics $\mu_1, \dots, \mu_k : \Sigma^* \rightarrow \mathbb{N}$ on instances. We will not usually name these metric functions explicitly. Rather, for nearly all problems treated herein, we will use n, m to refer to the number of variables and constraints, for problems involving Boolean formulas, or the number of nodes and edges for problems involving graphs.

The exponential complexity of a randomized algorithm A , implemented on a RAM machine and solving a parameterized promise problem $P = (L, I)$ with 2-sided error $\leq \frac{1}{3}$, is

$$c_A = \inf\{c \geq 0 \mid \forall x \in I \text{ time } A(x) \leq \text{poly}(|x|)2^{cn}\},$$

or ∞ if there is no such c . The *exponential complexity* of P is

$$c_P = \inf\left\{c_A \mid \text{randomized algorithm } A \text{ solves } P \text{ with 2-sided error } \leq \frac{1}{3}\right\}.$$

The *deterministic exponential complexity* of P is

$$dc_P = \inf\left\{c_A \mid \text{deterministic algorithm } A \text{ solves } P\right\}.$$

For most problems considered here (but not, for example, CNF-SAT), $|x| \leq \text{poly}(n)$, so that the $\text{poly}(|x|)$ can usually be dropped from the above definition. Clearly $c_P \leq dc_P$. The reverse inequality is not known, and even a strong hypothesis like $P = BPP$ does not obviously imply it since, for all we know, derandomization may square running time, assuming it can be done at all.

Chapter 3

2-CNF

Let V be a finite set of variables. The *solutions* of a Boolean formula ϕ is

$$\text{sol}(\phi) = \{a : \text{var}(\phi) \rightarrow 2 \mid \phi(a) = 1\}.$$

ϕ is *satisfiable* iff $\text{sol}(\phi) \neq \emptyset$. Formula-SAT is the problem of deciding whether a given Boolean formula ϕ is satisfiable.

A *literal* is a variable $v \in V$ or its negation $\bar{v}$. Recall our convention that $\bar{\bar{v}} = v$. Two literals l_1, l_2 *conflict* iff $\bar{l}_1 = l_2$. A *clause* is a set of literals, representing their disjunction (i.e. OR), and it is *proper* iff no two of its literals conflict. Unless stated otherwise, all clauses in the text are assumed to be proper. A *k-clause* (*($\leq k$)-clause*) is a clause of size $= k$ ($\leq k$). A *conjunctive normal form* formula (CNF) is a set of clauses, representing their conjunction (i.e. AND). A *k-CNF* is a set of ($\leq k$)-clauses.¹

We represent CNFs by sets of sets of literals for convenience, evaluation at the assignment $a : \text{var}(\phi) \rightarrow 2$ is still defined as one would expect:

- for a literal l , $l(a)$ is $a(l)$ if l is a variable, or $\overline{a(\bar{l})}$ otherwise;
- for a clause C , $C(a) = \bigvee_{l \in C} l(a)$;
- for a CNF ϕ , $\phi(a) = \bigwedge_{C \in \phi} C(a)$.

A *partial assignment* to $\text{var}(\phi)$ is a map $a : V' \rightarrow 2$ where $V' \subseteq \text{var}(\phi)$. The *restriction* $\phi|_a$ of ϕ to a is obtained from ϕ by removing all false literals and true clauses at a . Partial assignments are sometimes denoted by predicates, e.g. $\phi|_{x_1=0}$, $\phi|_{C=0}$.

CNF-SAT (*k*-SAT) is the problem of deciding whether a given CNF (*k*-CNF) is satisfiable, and is among the first combinatoric problems shown to be NP-complete [Coo71]. We first focus on 2-CNFs, surveying the complexity of various related problems. Though deciding their

¹The literature occasionally uses *k*-CNF to mean a set of *k*-clauses, e.g. in the Lovász Local Lemma.

satisfiability is only in P, several other problems involving 2-CNFs are NP-hard, and many of the tools for analyzing 2-CNFs will be useful for developing nontrivial algorithms, and avoiding dead ends, for other NP-hard problems.

A 2-CNF ϕ not containing $\emptyset$ can be represented by its (directed) *implication graph* $G = (V, E)$ where V is the set of literals of ϕ and $E = \{(a, b) \mid \{\bar{a}, b\} \in \phi\}$. Note that each 2-clause of ϕ will correspond to 2 edges in E , and each 1-clause will correspond to 1 edge. Improper clauses would correspond to self-loops. Conversely, if $G = (V, E)$ is a digraph where V is a set of literals, we say G *represents* the formula $\phi = \{\{\bar{a}, b\} \mid (a, b) \in E\}$. G obeys *path duality* iff $\forall a, b \in V (\exists \text{ a path from } a \text{ to } b \text{ in } G \leftrightarrow \exists \text{ a path from } \bar{b} \text{ to } \bar{a} \text{ in } G)$. The following is obvious.

Lemma 3.1 (2-CNF graph representation). *Let $\phi \in 2\text{-CNF}$ and G be its implication graph. Then G obeys path duality. If $\emptyset \notin \phi$, then G is the unique digraph representing ϕ that obeys path duality. If $\emptyset \in \phi$, then no digraph represents ϕ . Furthermore, any digraph obeying path duality represents a unique 2-CNF not containing $\emptyset$.*

In chapter 4, we will show that if $\emptyset \notin \phi$, then every clause logically implied by ϕ can be deduced from path connectivity in the implication graph of ϕ , which implies that 2-SAT can be solved in linear time.

However, there are still difficult problems involving 2-CNFs. Max-2-SAT (“given a 2-CNF with non-negative clause weights, is there an assignment so that the total weight of the satisfied clauses is at least some given value?”) is NP-complete (theorem 3.6), but can be approximated to within $\alpha \approx .94017$ of optimal² by a poly time algorithm [LLZ02], and this is amazingly close to optimal: unless $P = NP$, no poly time algorithm with approximation ratio of $\geq \frac{21}{22} + \epsilon \approx .9545 + \epsilon$ is possible [Hås01]; and, assuming the Unique Games Conjecture (UGC), [LLZ02] is tight in that no poly time algorithm with approximation ratio $\geq \alpha + \epsilon$ is possible [Aus07]. (It is also interesting that the inapproximability ratio for Max-E3-SAT matches the approximation ratio [Hås01].) Approximation of the unit weight version is essentially no easier: if we take a Max- k -SAT instance to be a *multiset* of clauses, then we can reduce to the unit weight case by approximation of the weights and repetition of the clauses.

The power of the implication graph may give the impression that the structure of the solution space of a 2-CNF is easy to grasp, but this is far from the truth. Deciding the simultaneous satisfiability of a monotone 2-CNF and a single linear inequality is NP-complete. To see this, reduce from Independent Set: given a graph $G = (V, E)$ and $k \in [0, n]$, $n = |V|$, treat V as a set of variables and construct $F = \{\{\bar{a}, \bar{b}\} \mid (a, b) \in E\} \in 2\text{-CNF}$ and the linear equation $\sum_{v \in V} v \geq k$. The problem remains NP-complete if the linear inequality is changed to an equality: add a set U of $n - k$ slack variables and change the linear inequality to $\sum_{v \in U \cup V} v = n$. The problem remains NP-complete if the inequality is replaced by a sparse linear system by reduction

² α was computed with a numerical optimization in Matlab, not by formal proof. [Aus07] demotes the state of this bound to “the believed approximation ratio”.

from 3-Coloring: given a graph $G = (V, E)$, create a variable $x_{i,j}$ for each node i and color j that means “assign i to j ”, set $F = \{\{\bar{x}_{i,j}, \bar{x}_{i',j}\} \mid \{i, i'\} \in E\}$, and create a linear system with equations $x_{i,1} + x_{i,2} + x_{i,3} = 1, i \in V$. Notice that the reduction still works even if we treat the equations as being over the field $\mathbb{Z}_2$.

#2-SAT (counting the number of solutions to a 2-CNF) is #P-complete [Val79b], even when restricted to monotone formulas (to be discussed later), although counting the number of maximum weight assignments to a weighted 2-CNF can be solved in time $O(2^{.317n})$ and poly space [FK07]. #Monotone 2-SAT is equivalent to #Vertex-Cover, which is NP-hard to approximate to within $2^{n^{1-\epsilon}}$ [Rot96]. If we restrict further so that the formula is monotone and, when viewed as the edge set of a graph, is bipartite (#Monotone Bipartite 2-SAT), then this is equivalent to #Bipartite Vertex Cover, and to this author the status of approximability is unknown. By contrast, there is an FPRAS for #DNF-SAT [KLM89] (theorem 3.7).

To prove the #P-completeness of #2-SAT is surprisingly difficult, considering that the #P-completeness of #3-SAT follows almost immediately from Cook’s computation tableaux reduction [Coo71]. We will give only an outline to highlight the techniques involved. The main problem is that any attempt to construct a 2-CNF whose solutions are in a k -to-1 relation with those of a given 3-CNF is doomed to fail, unless $P = NP$, since $2\text{-SAT} \in P$. This obstacle can be averted by first reducing to the problem of computing the permanent, a sort of weighted counting problem that allows negative weights. A more direct reduction would be interesting.

The *permanent* of an $n \times n$ matrix A is $\text{Perm}(A) = \sum_{\sigma} \prod_i A_{i,\sigma(i)}$, where the sum is over all permutations of n . A *cycle cover* of an edge-weighted graph $G = (V, E, w)$ is a set of node-disjoint simple cycles that covers V , and its weight is the product of the weights of the edges used. If A is the adjacency matrix of G , with $A_{i,j} = w(i, j)$ if $(i, j) \in E$ and 0 otherwise, then $\text{Perm}(A)$ is the sum of the weights of the cycle covers of G . Similarly, if $G = (L, R, E, w)$ is an edge-weighted bipartite graph with $|L| = |R|$ and A is its biadjacency matrix, which is the same as the adjacency matrix except that the row (column) indexes only range over L (R), then $\text{Perm}(A)$ is the sum of the weighted perfect matchings of G .

#3-SAT can be reduced to Permanent where the matrix entries are only in the set $\{-1, 0, 1\}$ ([Val79a, BDH93] plus some trivial observations) by constructing from the input 3-CNF F a graph G with a node for each variable and a gadget for each clause. The gadget representing clause C has 3 input edges and 3 output edges, each indexed with the literals of C . Edges outside the gadgets are called *external* and have weight 1, the other edges are *internal*. External edges are added so that for each literal l involving variable x , there is a cycle through x and the gadgets corresponding to clauses containing l . We want that each solution to F corresponds to a set of cycle covers with total weight $12^{|F|}$ so that knowing the permanent of the adjacency matrix of G allows deduction of the number of solutions of F .

To make this happen, the gadgets need the property that each contributes a factor of 12 to the total cycle cover weight iff the set s of input indexes it uses is the same as the set of

output indexes it uses and $s \neq \emptyset$. This can be achieved with a 7 node graph, although it's fairly complex and has edge weights in the set $\{-1, 1, 2, 3\}$. Also, although the *weight* of the cycle covers corresponding to a single solution to F is a constant, the *number* of them is not, and it is this clever weighting that forms much of the magic of the construction.

A simple gadget can replace each edge with weight 2 or 3 by a graph with edge weights 1 so that the matrix only has entries in $\{-1, 0, 1\}$. So this proof also shows that approximating the permanent of a $\{-1, 0, 1\}$ -matrix to within any factor is NP-hard. By contrast there is an FPRAS for nonnegative integer matrices [JSV04]. (Unfortunately, one does not easily reduce to the other: the standard reduction, to be discussed, from permanent with negative entries to one with nonnegative entries using modular arithmetic would require an exponentially small approximation factor to distinguish between 0 and > 0 .)

The integer permanent problem can be reduced to the nonnegative integer permanent problem by replacing A by $A \bmod Q$ when Q is any integer $>$ twice an upper bound on $\text{Perm}(A)$. This can be reduced to the permanent problem where each entry is 0 or a power of 2 by replacing each edge with weight a by a simple gadget representing the binary representation of a . This can be reduced to the permanent problem for 0-1 matrices by replacing each edge with weight 2^k by a simple gadget allowing k 2-way decisions for a cycle cover that passes through it.

The permanent problem for 0-1 matrices is the same as counting the number of perfect matchings in a bipartite graph. This can be reduced, via $O(n)$ oracle calls, to counting the number of matchings in a bipartite graph with a brilliant arithmetic trick [Val79b]. Given a bipartite graph $G = (L, R, E)$ and a parameter k , we construct a new graph G_k from G by attaching new nodes to each node in R so that there are many more choices of matchings possible. By being clever, we can arrange for the number of matchings in G_k to be a polynomial in k , where the coefficients are the number of matchings of each size in G . The oracle allows us to evaluate this polynomial. By evaluating at $|L| + 1$ points, we can interpolate to find the coefficients and therefore the number of perfect matchings of G (or for that matter, the number of matchings of any size).

Counting the number of matchings in a bipartite graph easily reduces to #Monotone 2-SAT by having a variable for each edge indicating whether it is in the matching and clauses forbidding inclusion of edges that touch.

Theorem 3.2. *There is an FPRAS for counting the number of matchings in a bipartite graph.*

Proof. Let $G = (L, R, E)$ be bipartite. Let $n = \min\{|L|, |R|\}$ and $r \in [0, n]$. Let L', R' be sets of new nodes, $L' \cap R' = \emptyset$, $|L'| = |R'| = r$. Define $G_r = (L \cup R', R \cup L', E \cup L \times L' \cup R' \times R)$. Then the number of matchings of G of size $n - r$ is the number of perfect matchings in G_r , and there is an FPRAS for the latter [JSV04]. By adding up $1 + \epsilon$ factor approximations for the number of perfect matchings of G_r for each $r \in [n]$ we get a $1 + \epsilon$ factor approximation for the number of matchings of G . $\square$

The reduction from counting the number of matchings of a bipartite graph to #Monotone 2-SAT is so simple that it is surprising, given theorem 3.2, that there is not an FPRAS for #Monotone 2-SAT, unless $P = NP$. To this author, the status of approximating #Monotone Bipartite 2-SAT is unknown. This restricted version is equivalent, via parsimonious reductions, to #Bipartite Vertex Cover, which is equivalent to the following combinatoric problem, which we will call *Exponential Intersection Counting*: given a set system $S \subseteq \mathcal{P}(U)$ where S is a multiset, evaluate $\sum_{T \subseteq S} 2^{|\cap T|}$.

Lemma 3.3. *#Monotone Bipartite 2-SAT, #Bipartite Vertex Cover, Exponential Intersection Counting are poly time parsimoniously interreducible.*

Proof. The first 2 are the same problem. To reduce from #Bipartite Vertex Cover to Exponential Intersection Counting, given $G = (L, R, E)$, let S be the multiset of complements of neighbor sets of nodes in L (or R). To reduce from Exponential Intersection Counting to #Bipartite Vertex Cover, given S , set $G = (S, \bigcup S, \{\{A, x\} \mid A \in S, x \in \bar{A}\})$. $\square$

Although the status of approximating #Bipartite Vertex Cover seems to be unknown, the very special case of #3-Regular Planar Bipartite Vertex Cover is #P-complete [XZ06]. Apparently, hardness results for restricted versions of standard counting problems are rare and inapproximability results are rarer due to the need for restriction-preserving reductions [Vad01].

We now show the NP-completeness of Max-2-SAT. Given $F \in \text{CNF}$, define the *not all equal* 3-SAT problem as follows.

$$\begin{aligned} \text{sol}_{\text{NAE}}(F) &= \{a \in 2^{\text{var}(F)} \mid \forall C \in F \exists l_1, l_2 \in C \ l_1(a) \neq l_2(a)\} \\ \text{NAE-3-SAT} &= \{F \in \text{3-CNF} \mid \text{sol}_{\text{NAE}}(F) \neq \emptyset\}. \end{aligned}$$

Lemma 3.4. $\forall F \in \text{CNF}, a \in \text{sol}_{\text{NAE}}(F) \ \bar{a} \in \text{sol}_{\text{NAE}}(F)$.

Proof. Let $C \in F$. Then $\exists l_1, l_2 \in C \ l_1(a) \neq l_2(a)$. So $l_1(\bar{a}) = \overline{l_1(a)} \neq \overline{l_2(a)} = l_2(\bar{a})$. So $\bar{a} \in \text{sol}_{\text{NAE}}(F)$ too. $\square$

Theorem 3.5. $\text{NAE-3-SAT} \in \text{NP-complete}$.

Proof. $\text{NAE-3-SAT} \in \text{NP}$ since we can verify whether an assignment is a NAE solution to a given formula by checking that each clause has a true and false literal. To show NP-hardness, we reduce from 3-SAT. Let $F \in \text{3-CNF}$. $\forall C = \{l_1, l_2, l_3\} \in F$, add 2 clauses $\{l_1, l_2, z_C\}, \{l_3, \bar{z}_C, b\}$ to formula G where z_C is a new variable for each clause and b is just one new variable. We claim that $F \in \text{3-SAT} \Leftrightarrow G \in \text{NAE-3-SAT}$.

$(\Rightarrow)$ Let $a \in \text{sol}(F)$. Extend a to an assignment $a' \in 2^{\text{var}(G)}$ as follows: $a'(b) = 0$, $\forall C = \{l_1, l_2, l_3\} \in F \ a'(z_C) = \overline{l_1(a)} \wedge \overline{l_2(a)}$. If $\overline{l_1(a)} \wedge \overline{l_2(a)} = 1$, then $l_1(a') = 0 \neq 1 = z_C(a')$ and $l_3(a') = 1 \neq 0 = b(a')$. If $\overline{l_1(a)} \wedge \overline{l_2(a)} = 0$, then one of $l_1(a'), l_2(a')$ is 1 while $z_C(a') = 0$ and $\bar{z}_C(a') = 1 \neq 0 = b(a')$. So $a' \in \text{sol}_{\text{NAE}}(G)$.

($\Leftarrow$) Let $a' \in \text{sol}_{\text{NAE}}(G)$. By lemma 3.4, $\text{wlog } b(a') = 0$. We claim that the restriction $a = a'|_{\text{var}(F)}$ is a solution to F . Let $C = \{l_1, l_2, l_3\} \in F$. If $l_3(a') = 0$, then $\bar{z}_C(a') = 1$, and so one of $l_1(a'), l_2(a')$ is 1. So $a \in \text{sol}(F)$. $\square$

Theorem 3.6. Max-2-SAT $\in$ NP-complete.

Proof. We reduce from NAE-3-SAT. Let $F \in 3\text{-CNF}$. We will construct an equivalent instance G of weighted 2-SAT. For each clause $C = \{l_1, l_2, l_3\} \in F$, add 6 clauses to G : $\{l_i, l_j\}, \{\bar{l}_i, \bar{l}_j\}$, for $i \neq j$. Notice that an assignment satisfies 1 or 2 literals in C iff ≥ 5 of the corresponding 6 clauses are satisfied. So G has an assignment satisfying $\geq 5|F|$ clauses iff F has a NAE satisfying assignment. That Max-2-SAT is in NP is clear. $\square$

Theorem 3.7 ([KLM89]). #DNF-SAT $\in$ FPRAS.

Proof. Let $F = \{T_1, \dots, T_m\} \in \text{DNF}$, $S_i = \text{sol}(T_i)$, $S = \text{sol}(F)$. Each S_i has 3 useful properties: $|S_i|$ can be computed in poly time, S_i can be uniformly sampled in poly time, membership in S_i can be decided in poly time. Let $U = \{(i, a) \mid a \in S_i\}$. Then U has these same 3 properties. Let $s = \max_i |S_i|$. Then $|S|, |U| \in [s, ms]$ and $|S| \leq |U|$. So $p = \frac{|S|}{|U|} \in [\frac{1}{m}, 1]$.

If we sample $(i, a) \in U$ uniformly at random, the probability that T_i is the first term satisfied by a is exactly p , and this can be tested in poly time. If we do this test n times and let X be the number of successes, then $\frac{X}{n}$ is a good estimator for p and $Y = \frac{X}{n}|U|$ is a good estimator for $|S|$. More precisely, by the Chernoff bound (theorem 2.7), for $\epsilon \in (0, 1)$,

$$\Pr(|Y - |S|| \geq |S|\epsilon) = \Pr(|X - np| \geq n\epsilon) \leq 2e^{-\frac{\epsilon^2 np}{3}} \leq 2e^{-\frac{\epsilon^2 n}{3m}},$$

which is $< \frac{1}{3}$ if we choose $n = \frac{6m}{\epsilon^2}$. $\square$

Chapter 4

Resolution

In this chapter, we show 2 basic proof systems on which the main k -SAT algorithms of chapter 6 are based. The notion of a contradictory cycle, in particular, will come in handy several times later.

Clauses C, D are *resolvable* iff they differ on exactly 1 variable, i.e. $\exists!$ literal l (whose variable is called the *conflict variable*) such that $l \in C$ and $\bar{l} \in D$. In this case, the *resolvent* of C, D is $R = \text{resolve}(C, D) = C \cup D - \{l, \bar{l}\}$.

Define a *resolution dag* G for $\phi \in \text{CNF}$ as a dag with a clause labeled at each node s.t. the label of each leaf node (i.e. with indegree 0) is in ϕ , and the label of each interior node (i.e. with indegree ≥ 1) is the resolvent of the labels of its 2 children. $\psi \in \text{CNF}$ is *deducible through resolution* from ϕ , and we write $\phi \vdash \psi$ iff there is a resolution dag for ϕ containing the clauses of ψ as node labels. We can extend this notation to individual clauses as well, e.g. $\phi \vdash C$. Say that a formula ϕ *implies* ψ , and write $\phi \models \psi$, iff $\text{sol}(\phi) \subseteq \text{sol}(\psi)$. Obviously $\vdash, \models$ are both reflexive and transitive.

Say G is *complete* iff the resolvent of every resolvable pair of node labels in G is already a node label in G . Though complete resolution dags are not necessarily unique, the set of their node labels is, since this is simply the closure of ϕ under the resolution operation. The dag itself can be seen as a proof that these clauses are in the closure.

Theorem 4.1 (Power of resolution). *Let $\phi \in \text{CNF}$ and C be a clause.*

(Soundness) If $\phi \vdash C$, then $\phi \models C$.

(Completeness) If $\phi \models C$, then $\exists D \subseteq C$ $\phi \vdash D$.

It would be cleaner if we could say $\phi \models C$ iff $\phi \vdash C$, but this is not true. We could make it true by allowing the resolution operator to produce supersets of clauses, but such an extension adds no real power and would make it onerous in some cases to exhaustively search all resolution proofs.

Proof. (Soundness) This can be shown by induction on the length l of a longest path from C to a leaf in the resolution dag. Obviously it holds for $l = 0$ since then C is at a leaf. Now suppose it holds for some l and let $l + 1$ be the length of a longest path from a node x labeled C to a leaf. Let y, z be the children of x and A, B be their labels. Then the length of a longest path from y or z to a leaf is $\leq l$. By the induction hypothesis, $\phi \models \{A, B\}$. Wlog suppose some variable $v \in A, \bar{v} \in B$. Then for any assignment a satisfying both A, B , if a sets v to true, then $B - \{\bar{v}\}$ is true, else $A - \{v\}$ is true. So $\phi \models \{A, B\} \models C$.

(Completeness) We will show that $\exists D \subseteq C$ and a resolution tree T for ϕ with D at the root by induction on $n = |\text{var}(\phi)|$. Suppose $n = 0$, so that $C = \emptyset$. If $\phi = \emptyset$, then the hypothesis $\phi \models C$ is false¹, so $\phi = \{\emptyset\}$ and a tree with just one node labeled C suffices.

Now suppose $n \geq 1$ and let $x \in \text{var}(\phi)$. If $C = \emptyset$, then ϕ is unsatisfiable, and so is $\phi|_{x=1}$. By the induction hypothesis, there is a resolution tree T_0 for $\phi|_{x=1}$ with $\emptyset$ at the root. Adding x back into the clauses of T_0 generates a resolution tree T'_0 for ϕ with either $\{\bar{x}\}$ or $\emptyset$ at the root. Similarly, there is a resolution tree T'_1 for ϕ with either $\{x\}$ or $\emptyset$ at the root. If neither of T'_0, T'_1 has $\emptyset$ at the root, then the labels at their roots can be resolved to yield $\emptyset$.

Suppose $C \neq \emptyset$. $\phi|_{C=\emptyset}$ is unsatisfiable and has $< n$ variables. By the induction hypothesis, there is a resolution tree for $\phi|_{C=\emptyset}$ with $\emptyset$ at the root. Adding the variables of C back into the tree yields a resolution tree for ϕ with some subset of C at the root. $\square$

Call a resolution tree for $\phi \in \text{CNF}$ with $\emptyset$ at the root a *refutation tree* for ϕ . Note that if $\phi \in 2\text{-CNF}$, then each of its non-root node labels $\{a, b\}$ can be interpreted as either the edge $\bar{a} \rightarrow b$ or $\bar{b} \rightarrow a$.

Lemma 4.2. *Let T be a resolution tree for $\phi \in 2\text{-CNF}$ with no node label equal to $\emptyset$. For each of the 2 interpretations $a \rightarrow b, \bar{b} \rightarrow \bar{a}$ of the label $\{a, b\}$ at the root as an edge, say $c \rightarrow d$, there is an ordering of the children of each interior node of T and an interpretation of each non-root node label as an edge such that the inorder sequence of node labels at the leaves forms a path in the implication graph G of T from c to d .*

Proof. We use induction on the depth of T . If T has depth 0, then the lemma is trivial, so assume T has positive depth. Let x, y be the children of the root and A, B be their labels. Let e be the conflict variable of A, B . The node label of at least one of x, y contains $\bar{e}$. Order x, y so that this node, say x , is a left child. Interpret A as an edge to a literal whose variable is e , and B as an edge from a literal whose variable is e . Now invoke the induction hypothesis to complete the proof. $\square$

Define a *contradictory cycle* in the implication graph as a path from some literal l to $\bar{l}$ to l .

¹being careful to interpret the left hand side as a set of clauses but the right hand side as a single clause

Corollary 4.3. *Let T be a refutation tree for $\phi \in 2\text{-CNF}$ and assume $\emptyset \notin \phi$. There is an ordering of the children of each interior node of T and an interpretation of each non-root node label as an edge such that the inorder sequence of node labels at the leaves forms a contradictory cycle in the implication graph G of T .*

Proof. Since $\emptyset \notin \phi$, T has positive depth. Let the left (right) child of the root have label $\bar{a}$ (a). The left (right) subtree of T does not contain $\emptyset$ as a node label since $\emptyset$ is not resolvable with any clause. By lemma 4.2, for the left (right) subtree, the children of each interior node can be ordered and the node labels can be interpreted so that the inorder sequence of node labels at the leaves form a path from a to $\bar{a}$ ($\bar{a}$ to a), which completes the proof. $\square$

We can define another proof system based on paths in the implication graph. Let $\phi \in 2\text{-CNF}$, $\emptyset \notin \phi$, G be the implication graph of ϕ , and let C be a (≤ 2) -clause. Say that ϕ *path deduces* C , and write $\phi \vdash_{\text{path}} C$, iff there is a path in G from $\bar{a}$ to b where $C = \{a, b\}$, or there is a contradictory cycle in G in the case that $C = \emptyset$.

Notice that path deduction is just slightly weaker than resolution in that the implication graph does not represent whether $\emptyset \in \phi$, which is why we add the hypothesis $\emptyset \notin \phi$ to the following corollaries.

Corollary 4.4. *Let $\phi \in 2\text{-CNF}$, $\emptyset \notin \phi$, C be a (≤ 2) -clause. Then $\phi \vdash C$ iff $\phi \vdash_{\text{path}} C$.*

Proof. $(\Rightarrow)$ Follows from lemma 4.2.

$(\Leftarrow)$ First suppose $C \neq \emptyset$. Let $C_1, \dots, C_m$ be the edge path in the implication graph G of ϕ deducing C . Interpreting each C_i as a clause, define the following left-heavy resolution tree T : let $D_1 = C_1$ and for $i = 2, \dots, m$, let D_i be the resolvent of D_{i-1} and C_i . Now suppose $C = \emptyset$ and suppose $\phi \vdash_{\text{path}} \{\bar{a}\}, \{a\}$. From the first case, $\phi \vdash \{\{\bar{a}\}, \{a\}\} \vdash \emptyset$. $\square$

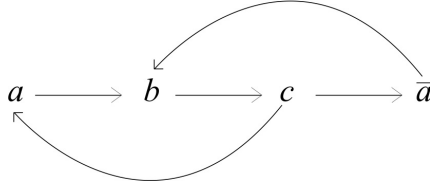
Corollary 4.5. *Let $\phi \in 2\text{-CNF}$, $\emptyset \notin \phi$. ϕ is unsatisfiable iff its implication graph contains a contradictory cycle.*

The advantage of path deductions for 2-CNFs is that we may restrict our attention to *simple paths* without losing any deductive power. We will find a nice application of this idea when we develop algorithms for $\Pi_23\text{-SAT}$ in §7.2. An application we can state here is to use Tarjan's linear time strong components algorithm [Tar72], and then detect whether any literal is in the same strong component as its complement. This gives a linear time 2-SAT algorithm. In fact, Tarjan et al [APT79] show that 2-QBF, the problem of deciding the truth value of a 2-CNF where the variables are quantified arbitrarily, is solvable in linear time using almost the same algorithm.

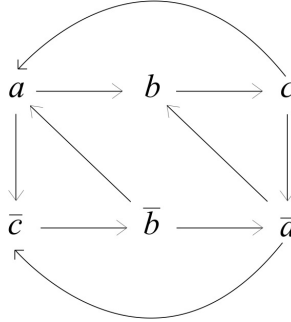
4.1 Restricted forms of resolution

Notice in the proof of completeness in theorem 4.1 that the only case in which node labels get reused (i.e. appear on more than 1 node) is when trying to derive $\emptyset$. One might ask whether resolution would still be complete in the sense above if we restricted the resolution tree to never reuse node labels. Certainly this would be an advantage if one were trying to enumerate all resolution trees. But alas, such trees are not as powerful, as the following example will show (corollary 4.6).

Corollary 4.5 provides an easy way to construct unsatisfiable 2-CNFs. Below we construct a graph on 4 literals such that the only such cycle uses some edge twice, namely the one from b to c .



We can extend this graph to satisfy the path duality property of lemma 3.1:



This graph G_0 represents the 2-CNF $\phi_0 = \{\{\bar{a}, b\}, \{\bar{b}, c\}, \{\bar{c}, \bar{a}\}, \{\bar{a}, b\}, \{\bar{c}, a\}\}$, which has 3 variables and 5 clauses. It can be checked by brute force that every contradictory cycle in this graph uses some edge twice, extending the intuition we developed about the first graph.

Corollary 4.6. *Every refutation tree T for ϕ_0 uses some node label twice.*

Proof. By corollary 4.3, some ordering and interpretation of the leaves of T is a contradictory cycle in G_0 , and so uses some edge twice. So some node label is used twice. $\square$

We can weaken our notion of tree-with-unique-labels by allowing the leaves to have repeated labels but the labels of the interior nodes must be distinct. Such a proof system is powerful enough to refute unsatisfiable 2-CNFs, as we now show.

Lemma 4.7. *Every unsatisfiable $\phi \in 2\text{-CNF}$ has a refutation tree with no repeated interior node labels.*

Proof. There is a *simple* contradictory cycle p from some variable a to $\bar{a}$ to a , i.e. where both the path from a to $\bar{a}$ and that from $\bar{a}$ to a is simple (though viewed as a single path, p might not be simple). Applying the proof of corollary 4.4 yields the desired tree. To see this, note that each interior node label in the left (right) subtree contains $\bar{a}$ (a) and none in the right (left) contains $\bar{a}$ (a), and each subtree individually does not repeat interior node labels because the path it represents is simple. $\square$

4.2 Implication trees

Resolution dags represent deductions made by DPLL-like algorithms (discussed in chapter 6): the labels of interior nodes being intermediate theorems deduced. We now propose a combinatoric structure, related to tableaux proofs, more closely representing random walk algorithms.

Let $\phi \in \text{CNF}$ with literal set S . A tree $T = (V, E)$ with node labels $L : V \rightarrow S$ is an *implication tree* iff for each interior node v_1 with children $v_2, \dots, v_k$ and where $l_i = L(v_i)$, there is a clause $\{\bar{l}_1, l_2, \dots, l_k\} \in \phi$. Note that this definition permits a node labeled $\bar{l}$ to include l among the labels of its children, which allows unit clauses to generate edges. The following obvious lemma captures the semantics of implication trees.

Lemma 4.8. *Let T be an implication tree for $\phi \in \text{CNF}$. If $a \in \text{sol}(\phi)$, then at a , each interior node with a true label has a child with a true label.*

Say a root-to-leaf path is a *contradiction path* iff $\exists l \in S$ such that the path contains labels l followed by $\bar{l}$ followed by l . An implication tree is a *contradiction tree* iff it is nonempty and each root-to-leaf path contains a contradiction path. We now show the soundness and completeness of this proof system.

Theorem 4.9 (Power of implication trees). *Let $\phi \in \text{CNF}$.*

(Soundness) If ϕ has a contradiction tree, then $\phi \notin \text{SAT}$.

(Completeness) If $\phi \notin \text{SAT}$, then $\emptyset \in \phi$ or ϕ has a contradiction tree.

Proof. (Soundness) Let T be a contradiction tree for ϕ and suppose indirectly that $a \in \text{sol}(\phi)$. At a , let v be a highest true node in T so that its ancestors are all false. (If there is no true node, then in the following, just set p to be any root-to-leaf path.) By lemma 4.8, some path p' starting at v and going to a leaf is all true. Let p be the root-to-leaf path that includes p' . Then p starts with all true nodes and is followed by all false. But p contains some literal l followed by $\bar{l}$ followed by l . By the pigeonhole principle, some 2 consecutive of these literals is in the same region and so are assigned the same way, a contradiction.

(Completeness) Say that $\phi \in \text{CNF}$ is *minimal unsatisfiable* iff $\phi \notin \text{SAT}$ and $\forall C \in \phi, \phi - \{C\} \in \text{SAT}$. An *unsatisfiable core*² of $\phi \in \text{CNF-SAT}$ is a subset of ϕ that is minimal unsatisfiable. Obviously every unsatisfiable ϕ contains an unsatisfiable core.

We use induction on the number of variables n to prove something subtle: for every minimal unsatisfiable $\phi \in \text{CNF}$ and each literal l of a variable actually occurring in ϕ , there is a contradiction tree T for ϕ with l at the root.

If $n = 0$, then ϕ has no literals in it and we are done. Otherwise, let l be a literal. We want to construct a contradiction tree with l at the root. Let $\psi = \phi|_{l=1}$. $\psi \notin \text{SAT}$ since ψ is a restriction of ϕ . We claim that

$$\begin{aligned} &\text{some unsatisfiable core } \psi' \text{ of } \psi \text{ contains a clause } C \\ &\text{such that } C \cup \{\bar{l}\} \in \phi. \end{aligned} \tag{*}$$

Otherwise $\psi' \subseteq \phi$. $\bar{l}$ occurs in some clause in ϕ since ϕ is minimal unsatisfiable. So $\psi' \subset \phi$ and so $\psi' \in \text{SAT}$, a contradiction. So (*) holds. By the induction hypothesis, for each literal l' in ψ' , there is a contradiction tree for ψ' with l' at the root.

Case I: $\psi' = \{\emptyset\}$. Then $\{\bar{l}\} \in \phi$. In this case, define tree T to contain l at the root and a single child labeled $\bar{l}$.

Case II: $\psi' \neq \{\emptyset\}$. Then $\emptyset \notin \psi'$ since ψ' is minimal unsatisfiable. So $C \neq \emptyset$, say $C = \{l_1, \dots, l_k\}$. For $i = 1, \dots, k$, there is a contradiction tree T_i for ψ' with l_i at the root. Reinserting $\bar{l}$ into T_i (i.e. as a child of those internal nodes whose children came from a clause D of ψ' where $D \cup \{\bar{l}\} \in \phi$) yields an implication tree T'_i for ϕ with l_i at the root and each root-to-leaf path leads to a contradiction or to $\bar{l}$. Define tree T with l at the root and subtrees T'_i .

For both cases I and II, T is an implication tree for ϕ with l at the root and each root-to-leaf path leads to a contradiction or to $\bar{l}$. By symmetry, there is an implication tree T' for ϕ with $\bar{l}$ at the root and each root-to-leaf path leads to a contradiction or to l . Glue copies of T' to the end of each $\bar{l}$ in T to get the desired contradiction tree. $\square$

Consider the following simple upper bounds on the depth $d(n)$ and size $s(n)$ of the contradiction trees produced by the above proof. To get a tree for n variables, we take 2 trees each for $n-1$ variables, add ≤ 1 node to each internal node of one, and glue copies of the second onto the first. A tree for 1 variable has depth ≤ 2 and size ≤ 3 . So

$$\begin{aligned} d(n) &\leq 2d(n-1) & d(1) &\leq 2 \\ s(n) &\leq s(n-1) + s(n-1)^2 & s(1) &\leq 3. \end{aligned}$$

So $d(n) \leq 2^n$. The recurrence for $s(n)$ yields an upper bound somewhere between $\sqrt{2}^{2^n}$ and $\sqrt{6}^{2^n}$. But these bounds don't take advantage of the fact that gluing trees together opens up

²The literature calls these *minimal unsatisfiable cores*, but this seems silly since then 'core' becomes another word for 'subset'.

opportunities for contradiction paths that are partially in each. We now show much tighter upper bounds.

Theorem 4.10. *If $\phi \in \text{CNF}$ has a contradiction tree, then it has one of depth $\leq (2n)^2$ and size $\leq n^{(2n)^2}$ where $n = |\text{var}(\phi)|$.*

Proof. Let T be a contradiction tree with the smallest possible size among those with the least possible depth, and let $v_0, \dots, v_d$ be a root-to-leaf path. Let l be a literal and $v_{i_1}, \dots, v_{i_k}$ with $i_1 < \dots < i_k < d$ be the nodes other than the leaf that are labeled l . For $j = 1, \dots, k$, let S_j be the set of labels on $v_1, v_2, \dots, v_{i_j}$.

We claim $S_1 \subset \dots \subset S_k$. For suppose indirectly that $S_j = S_{j+1}$ for some j . Then we could replace the subtree rooted at v_{i_j} with that at $v_{i_{j+1}}$ to obtain a contradiction tree of no greater depth and strictly smaller size, contradicting the minimality of T . So $S_1 \subset \dots \subset S_k$, which implies $k \leq 2n$, which implies $d \leq (2n)^2$. The size bound comes from observing that the branching factor of T is $\leq n$. $\square$

The above bound is still not small enough for a contradiction tree to be a viable alternative to resolution as a general purpose refutation method, but it still may be handy in special cases or as a conceptual tool.

Chapter 5

Sparsification

[IPZ01] presents a subexponential time Turing reduction from k -SAT to bounded-clause-density- k -SAT. In particular, they give an algorithm A so that for each $k, \epsilon > 0, \phi \in k$ -CNF, $A_{k,\epsilon}(\phi)$ outputs $\phi_1, \dots, \phi_s \in k$ -CNF, with $n^{O(k)}$ delay (meaning $n^{O(k)}$ time passes before outputting each successive element of the sequence) and so that $s \leq 2^{\epsilon n}$, $\text{sol}(\phi) = \bigcup_i \text{sol}(\phi_i)$, and the number of occurrences of each literal in each ϕ_i is bounded above by a constant, called the *sparsification constant*, depending only on k, ϵ .

The original proof [IPZ01] of the lemma bounds the sparsification constant from above by a double exponential in k , roughly $(\frac{8k2^k}{\epsilon})^{2^{k-1}}$. We will bring this down to $O(\frac{k}{\epsilon})^{3k}$, rewriting the proof of the sparsification lemma for clarity. A sketch of this proof also appeared in [CIP06]. The improvement comes from tightening a single combinatoric inequality, propagating it through the rest of the analysis, and analyzing a curious sequence, each term of which is the convolution of the preceding terms.

5.1 Why it's useful

As long as we are aiming to produce an exponential time algorithm, we may usually assume that any input $\phi \in k$ -CNF is sparsified, which is especially useful in algorithms that need a linear number of clauses, as this bounds the degree of interaction between variables and between clauses. For example, define the *variable dependence graph* to have the variables as nodes and an edge between nodes iff they are in a clause together. Similarly we can define the *clause dependence graph* to have clauses as nodes and an edge between nodes iff they have a variable in common. The variable and clause dependence graphs of sparsified formulas are both bounded in degree by a constant depending only on k, ϵ .

As a concrete example of how sparsification can be used, let Unique- k -SAT be k -SAT

but where the input is promised to have ≤ 1 solution, and define

$$\begin{aligned} s_k &= c_{k\text{-SAT}} & s_\infty &= \lim_{k \rightarrow \infty} s_k \\ \sigma_k &= c_{\text{Unique-}k\text{-SAT}} & \sigma_\infty &= \lim_{k \rightarrow \infty} \sigma_k. \end{aligned} \tag{5.1}$$

In [CIKP08] (and here in theorem 9.17) it is shown that $\sigma_3 = 0 \Rightarrow s_\infty = 0$ by demonstrating a subexponential time randomized reduction from k -SAT to Unique-3-SAT that only linearly increases the number of variables. To make this reduction work, it is crucial that the input formula have only $m = O(n)$ clauses because reducing the clause width to 3 by the standard algorithm increases the number of variables to $O(n + m)$. For this to be linear in n we must first sparsify ϕ .

There is a close relationship between the complexity of k -SAT and SAT_Δ (the satisfiability problem for CNF formulas with clause-to-variable ratio $\leq \Delta$). In fact, we suspect that the problems can be reduced to each other in subexponential time when $\Delta = \Theta(2^k)$. To show such an assertion, we need tight reductions. In particular, to show that k -SAT reduces to SAT_Δ in subexponential time for $\Delta = O(2^k)$, it would suffice to bound the sparsification constant by $O(2^k)$. We do not accomplish this, but do step in the right direction. Also, in chapter 10, we show a slightly different assertion: that c_{SAT_Δ} is not too much larger than $c_{k\text{-SAT}}$ when $k = \Theta(\lg \Delta)$.

5.2 Sparsification algorithm

The idea of the algorithm is simple: if a subexpression H of ϕ occurs frequently enough and we branch on it, then both branches will greatly simplify ϕ . The only trick is how to decide what constitutes ‘frequently enough’ and in what order to branch. Greater depth of the design philosophy can be found in [IPZ01].

Recall that a *clause* is a set of noncontradictory literals so that every subset of a clause is a clause. A *k-clause* is a clause of size (exactly) k . A *k-CNF* ϕ is a set of clauses each of size $\leq k$. ϕ is *reduced* iff no clause is a subset of any other. The *reduction* of $\phi = \text{red } \phi = \{\subseteq\text{-minimal elements of } \phi\}$.

Let θ_i be a positive increasing sequence of integers to be specified later. For now, let us insist that $\theta_0 = 2$. Define

$$\begin{aligned} \phi_c &= \{C \in \phi \mid |C| = c\} \\ \phi_{c,h} &= \{H \mid |H| = h \geq 1 \wedge |\{C \in \phi_c \mid H \subseteq C\}| \geq \theta_{c-h}\}. \end{aligned}$$

c, h stand for clause size, heart size. The sparsification algorithm is in figure 5.1.

Several optimizations come to mind: before calling $A_{k,\epsilon}(\phi \cup P)$, we could set each $l \in H$ to 0 and simplify the formula; we could also terminate early if ϕ contains contradictory unit clauses. These optimizations would only change a few of the constants in the following analysis, so we don’t bother making them.

```

1   $A_{k,\epsilon}(\phi \in k\text{-CNF})$ 
2   $\phi \leftarrow \text{red } \phi$ 
3  if  $\exists(c, h) \phi_{c,h} \neq \emptyset$  {
4      select the lexicographically first  $(c, h)$ 
5      (meaning smallest  $c$ , then largest  $h$ ) such that  $\phi_{c,h} \neq \emptyset$ 
6      select an arbitrary  $H \in \phi_{c,h}$ 
7       $P \leftarrow \{C - H \mid H \subseteq C \in \phi_c\}$ 
8      /* branch: if we set  $H$  to 1, we call this a heart operation
9      if we set  $H$  to 0, we call this a petal operation */
10      $A_{k,\epsilon}(\phi \cup \{H\})$ 
11      $A_{k,\epsilon}(\phi \cup P)$ 
12 }
13 output  $\phi$  /*  $\phi$  is sparsified */

```

Figure 5.1: sparsification algorithm

We break up the analysis into 2 parts. First we show properties that hold regardless of our choice of the θ_i , then we choose the θ_i and show what new properties our choices cause.

5.3 Basic properties

Let T be the binary tree of the computation, each node of which is labeled with a k -CNF. The root is labeled $\text{red } \phi$. If node x is labeled ψ and the condition on line 3 is satisfied, then x will have 2 children, else 0 children (so T is *full*). Let (c, h) be as in line 4 and $\text{choice}(\phi_{c,h})$ denote the arbitrary choice made on line 6. Let

$$\begin{aligned}
 H(\psi) &= \text{heart of } \psi = \text{choice}(\psi_{c,h}) \\
 F(\psi) &= \text{flower of } \psi = \{C \in \psi_c \mid H(\psi) \subseteq C\} \\
 P(\psi) &= \text{petals of } \psi = \{C - H(\psi) \mid C \in F(\psi)\}.
 \end{aligned}$$

The left child of x is labeled $\text{red}(\psi \cup \{H(\psi)\})$. Note carefully that this could be smaller than $(\psi - F(\psi)) \cup \{H(\psi)\}$ since $F(\psi) \subseteq \psi_c$. In other words, the flower is defined as a subset of the clauses of size c , but reduction could discard clauses of any size (smaller or larger than c) as long as $H(\psi)$ is a proper subset of them. The right child of x is labeled $\text{red}(\psi \cup P(\psi))$. Again, we may be discarding more than $F(\psi)$.

A *c-flower* F of ϕ is a subset of ϕ_c with nonempty intersection. Let $H = \bigcap F, h = |H|$. F is *admissible* iff $|F| \geq \theta_{c-h}$. Say ϕ is *c-sparsified* iff $\forall c' \leq c$, ϕ has no admissible c' -flower. Also, ϕ is *sparsified* iff ϕ is k -sparsified.

Lemma 5.1. *Suppose ψ is not sparsified. (e.g. the label of an interior node of T) Then $\emptyset \notin P(\psi) \neq \emptyset$ (so that $0 < h < c$) and $H(\psi) = \bigcap F(\psi)$.*

Proof. If $\emptyset \in P(\psi)$ or $P(\psi) = \emptyset$, then since each element of P has the same size, we have $|P(\psi)| \leq 1 < \theta_0 \leq \theta_{c-h}$, a contradiction. Now we show $H(\psi) = \bigcap F(\psi)$.

($\subseteq$) Clear.

($\supseteq$) This depends on the lexicographical ordering of the (c, h) pairs. Suppose indirectly that $H = H(\psi) \subset \bigcap F(\psi) = H'$. Then

- $h' = |H'| > |H| = h$
- H' occurs as a subset of clauses in ψ_c with the same frequency f as H does, since $\{C \in \psi_c \mid H' \subseteq C\} = \{C \in \psi_c \mid H \subseteq C\}$.

So $h' > h \geq 1 \wedge f \geq \theta_{c-h} \geq \theta_{c-h'} \wedge \psi_{c,h'} \neq \emptyset$, contradicting the minimality of (c, h) . $\square$

Let us say that CNF α is a *refinement* of β iff $\forall a \in \alpha \exists b \in \beta a \subseteq b$. Note that $\text{red } \psi$ is a refinement of ψ since $\text{red } \psi \subseteq \psi$. α is a *proper refinement* of β iff α is a refinement of β and $\alpha \neq \beta$. Refinement is transitive, but proper refinement is *not*, although it is if we restrict to reduced CNFs.

Lemma 5.2. *If ψ, ψ' are reduced, ψ is not sparsified, and ψ is a refinement of ψ' , then*

$$\begin{aligned} H(\psi) &\in \text{red}(\psi \cup \{H(\psi)\}) - \psi' \\ P(\psi) &\subseteq \text{red}(\psi \cup P(\psi)) - \psi'. \end{aligned}$$

Proof. If $H(\psi) \notin \text{red}(\psi \cup \{H(\psi)\})$, then $\exists a, b \in \psi a \subset H(\psi) \subset b$, contradicting that ψ is reduced. Similarly, if $\exists a \in P(\psi) - \text{red}(\psi \cup P(\psi))$, then a fails to be in $\text{red}(\psi \cup P(\psi))$ either because it is a proper superset of some $b \in \psi$ or some $c \in P(\psi)$. The first case contradicts that ψ is reduced; the second, that the elements of $P(\psi)$ have the same size.

If $a \in (\{H(\psi)\} \cup P(\psi)) \cap \psi'$, then we would have $\exists b \in \psi, c \in \psi' a \subset b \subseteq c$, contradicting that ψ' is reduced. $\square$

Let us use the notation $l(x)$ to refer to the label of node x in T .

Lemma 5.3. *Let y be a proper ancestor of x in T . Then $\alpha = l(x)$ is a proper refinement of $\beta = l(y)$. Furthermore, the number of clauses in the labels of the nodes from y to x forms a nonincreasing sequence.*

Proof. Notice that for any nonsparsified CNF ψ and any 2 refinements γ, δ , we have that $\psi, \{H(\psi)\}, P(\psi), \text{red}(\psi), \gamma \cup \delta$ are all refinements of ψ , and refinement is transitive. By induction on the distance from x to y , α is a refinement of β .

To show that $\alpha \neq \beta$, let $z = \text{parent}(x)$ and $\gamma = l(z)$. If x is the left child of z , then by lemma 5.2, $H(\gamma) \in \alpha - \beta$. If x is the right child of z , then by lemma 5.1, $\exists a \in P(\gamma)$, and by lemma 5.2, $a \in \alpha - \beta$. Either way we conclude that α is a proper refinement of β .

The last assertion can be shown by induction: for any nonsparsified CNF ψ ,

$$|\text{red}(\psi \cup \{H(\psi)\})| \leq |\psi \cup \{H(\psi)\} - F(\psi)| \leq |\psi|$$

$$\text{and } |\text{red}(\psi \cup P(\psi))| \leq |\psi \cup P(\psi) - F(\psi)| \leq |\psi|.$$

□

Let r be the root of T and p be a path from r to a leaf. A clause a is *introduced* at node x iff either (x is a left child and $a = H(l(\text{parent}(x)))$) or (x is a right child and $a \in P(l(\text{parent}(x)))$). So no clause is introduced at r . We say a is *introduced along* p iff $\exists$ node x along p such that a is introduced at x .

A clause a is *new* at x iff $a \in (l(x) - l(r)) - \bigcup_{\text{proper ancestors } y \text{ of } x} l(y)$. So no clause is new at r . We say a is *new along* p iff $\exists$ node x along p such that a is new at x .

Lemma 5.4. *Clause a is introduced at node x iff a is new at x .*

Proof. ($\Rightarrow$) $x \neq r$. a is new at x by lemmas 5.2, 5.3 applied to $l(\text{parent}(x))$.

($\Leftarrow$) Since $a \in l(x) - l(r)$, $x \neq r$. Let $z = \text{parent}(x)$. Then $a \in l(x) - l(z)$. So $a = H(l(z))$ if x is a left child and $a \in P(l(z))$ if x is a right child. So a is introduced at x . □

This tells us that each clause a can be introduced at most once along any root-to-leaf path in T .

Lemma 5.5. *Let x be a node of T at which a clause of size c is introduced. Then $l(\text{parent}(x))$ is c -sparsified.*

Proof. If $l(\text{parent}(x))$ has an admissible c' -flower for some $c' \leq c$, then a clause of size $< c$ would have been introduced at x instead of a clause of size c . □

Lemma 5.6. *Let ψ be c -sparsified. Then*

$$1. \forall h\text{-clause } a \text{ with } h \geq 1 \quad |\{b \in \psi_c \mid a \subseteq b\}| < \theta_{c-h}$$

$$2. |\psi_c| < \theta_{c-1} \frac{2n}{c}$$

Proof. Let $F = \{b \in \psi_c \mid a \subseteq b\}$ and suppose indirectly that $|F| \geq \theta_{c-h}$. Let $H = \bigcap F$. Then $a \subseteq H$, and so $|H| \geq h$. So $|F| \geq \theta_{c-h} \geq \theta_{c-|H|}$. So F is an admissible c -flower, contradicting that ψ is c -sparsified.

Now suppose indirectly that $|\psi_c| \geq \theta_{c-1} \frac{2n}{c}$. Then the average number of times each literal occurs in ψ_c is

$$\begin{aligned} \frac{1}{2n} \sum_{\text{literals } l} \sum_{b \in \psi_c} \begin{cases} 1 & \text{if } l \in b \\ 0 & \text{else} \end{cases} &= \frac{1}{2n} \sum_{b \in \psi_c} \sum_{\text{literals } l} \begin{cases} 1 & \text{if } l \in b \\ 0 & \text{else} \end{cases} \\ &= \frac{1}{2n} |\psi_c| c \geq \theta_{c-1}. \end{aligned}$$

Some literal l achieves at least the average, which contradicts the first assertion. $\square$

Let us say that a clause a *eliminates* clause b at node x iff $a \in l(x)$, $b \in l(\text{parent}(x))$, and $a \subset b$. Note that it is possible for more than 1 clause to eliminate a given clause at x .

Lemma 5.7. *A clause b can only be eliminated by a new clause a , and only at the node where a is introduced.*

Proof. Let b be eliminated by a at x and let $\psi = l(\text{parent}(x))$. If x is a left (right) child, then $b \in \psi - \text{red}(\psi \cup \{H(\psi)\})$ ($b \in \psi - \text{red}(\psi \cup P(\psi))$). Since ψ is reduced, we must have $a = H(\psi) \subset b$ ($a \in P(\psi)$, $a \subset b$), so a is introduced at x . By lemma 5.4, a is also new at x . $\square$

Lemma 5.8. *Let p be a root-to-leaf path, a be a clause along p . Then a does all of its eliminating precisely at the node where a is introduced. (clauses at the root cannot eliminate anything)*

Proof. If a eliminates at 2 distinct nodes x, y along p , then by lemma 5.7, a is new at both x, y , and one is a proper ancestor of the other, contradicting the definition of new. $\square$

The following lemma is the source of our improvement in the sparsification constant. The analysis in [IPZ01] bounded the number of new c -clauses that a single clause could eliminate by $2\theta_{c-1} - 2$.

Lemma 5.9. *Let a be a clause of size $h < c$. If a eliminates any new clauses of size c , then a can eliminate at most $2\theta_{c-h} - 2$ clauses of size c (new or not) along root-to-leaf path p .*

Proof. Suppose indirectly that a eliminates $\geq 2\theta_{c-h} - 1$ clauses of size c along p and let γ be this set of clauses. By lemma 5.8, a must eliminate γ all at once when a is introduced. Let x be the first node along p such that $\gamma \subseteq l(x) = \alpha$.

Since γ contains a new clause, $x \neq r$. (This seemingly trivial use of the fact that γ contains a new clause is crucial, since this is our only use of this fact and the lemma is actually false if we do not assume γ contains a new clause.) By minimality of x , some c -clause of γ is introduced at x . Let $y = \text{parent}(x)$, $\beta = l(y)$. By lemma 5.5, β is c -sparsified. By lemma 5.6, $|\{b \in \beta_c \mid a \subseteq b\}| \leq \theta_{c-h} - 1$, yet $|\{b \in \alpha_c \mid a \subseteq b\}| \geq 2\theta_{c-h} - 1$. So $\geq \theta_{c-h}$ of the clauses of γ are introduced at x .

If x is a right child, then $\exists$ a flower $F \subseteq \beta$ with heart $H = H(\beta)$ and petals all from γ and $|F| \geq \theta_{c-h}$. (i.e. $\exists H \subseteq \bigcap F \forall C \in F \ C - H \in \gamma$) Suppose indirectly that $a \subseteq H$. Let $b \in \gamma$.

Since b is also a petal of F , $a \cap b \subseteq H \cap b = \emptyset$, contradicting that a eliminates b . So $a - H \neq \emptyset$. But then $H \cup a$ is an even bigger heart of the same flower contradicting that we branched on H at y .

If x is a left child, then $2 = \theta_0 \leq \theta_{c-h} \leq$ the number of clauses introduced at x , which is 1, a contradiction. $\square$

5.4 Choosing the θ_i

We wish to define constants β_c , dependent only upon k, ϵ, c such that the number of new c -clauses in a root-to-leaf path p is $\leq \beta_c n$.

For an $\alpha \geq 2$ to be determined later, set

$$\begin{aligned} \beta_1 &= 2, & \theta_0 &= 2 \\ \beta_c &= \sum_{h=1}^{c-1} 4\alpha\beta_{c-h}\beta_h & & \text{for } c \geq 2 \\ \theta_c &= \beta_c \alpha & & \text{for } c \geq 1. \end{aligned}$$

Since $\alpha \geq 2$, the θ_c are increasing.

Lemma 5.10. *The number of new clauses of size $\leq c$ in any root-to-leaf path p of T is $\leq \beta_c n$.*

Proof. We use induction on c . The number of new 1-clauses is obviously $\leq 2n = \beta_1 n$. For $c > 1$, we have

$$\begin{aligned} & \# \text{ new clauses of size } \leq c \\ & \leq \sum_{h=1}^{c-1} \# \text{ new } c\text{-clauses that got eliminated by a new } h\text{-clause} \\ & \quad + \# \text{ } c\text{-clauses in the leaf of } p + \# \text{ new clauses of size } \leq c-1 \\ & \quad \text{(using lemma 5.7)} \\ & \leq \sum_{h=1}^{c-1} (2\theta_{c-h} - 2)\beta_h n + \theta_{c-1} \frac{2n}{c} + \beta_{c-1} n \\ & \quad \text{(by lemmas 5.6, 5.9, and the induction hypothesis)} \\ & \leq \sum_{h=1}^{c-1} 2\alpha\beta_{c-h}\beta_h n + 2\alpha\beta_{c-1} n \\ & \leq \sum_{h=1}^{c-1} 4\alpha\beta_{c-h}\beta_h n = \beta_c n. \end{aligned}$$

$\square$

Lemma 5.11. *The length of any path p in T is $\leq \beta_{k-1} n$.*

Proof. By lemma 5.4, each edge of T introduces at least new 1 clause. The length of p is at most the number of new clauses, which is at most $\beta_{k-1}n$, by lemma 5.10. $\square$

Lemma 5.12. *The number of petal operations (right edges) in any path p of T is $\leq \frac{(k-1)n}{\alpha}$.*

Proof.

$$\begin{aligned}
& \# \text{ petal operations} \\
&= \sum_{c=1}^{k-1} \# \text{ petal operations that introduce } c\text{-clauses} \\
&\leq \sum_{c=1}^{k-1} \frac{\# \text{ new } c\text{-clauses}}{\theta_c} \\
&\leq \sum_{c=1}^{k-1} \frac{\beta_c n}{\theta_c} \quad \text{by lemma 5.10} \\
&= \frac{(k-1)n}{\alpha}.
\end{aligned}$$

$\square$

Wlog suppose $\epsilon \leq 1, k \geq 3$. Choose $\alpha = \frac{2(k-1)^2}{\epsilon} \lg \frac{32(k-1)^2}{\epsilon}$.

Lemma 5.13. *T has $s \leq 2^{\epsilon n}$ leaves.*

Proof. We will show in §5.6, theorem 5.21, that

$$\forall c \geq 1 \quad \beta_c \leq 4(32\alpha)^{c-1}, \quad (5.2)$$

and this is tight up to a subexponential factor. Since a leaf of T is determined by the sequence of left and right edges taken to reach it from the root, we have from lemmas 5.11, 5.12

$$\begin{aligned}
s &\leq \sum_{j=0}^{\lfloor \frac{Kn}{\alpha} \rfloor} \binom{\beta_K n}{j} \quad \text{letting } K = k-1 \\
&= |B_{\frac{Kn}{\alpha}}(0^{\beta_K n})| \\
&\leq 2^{h(\frac{K}{\alpha\beta_K})\beta_K n} \quad \text{by theorem 2.3,}
\end{aligned}$$

the exponent of which divided by n is

$$\begin{aligned}
&\leq \frac{K}{\alpha} \lg \left(\frac{4\alpha\beta_K}{K} \right) \quad \text{by theorem 2.2} \\
&\leq \frac{K}{\alpha} \lg ((32\alpha)^K) \quad \text{by (5.2)} \\
&= 32K^2 \frac{\lg(32\alpha)}{32\alpha}.
\end{aligned} \quad (5.3)$$

We would like to choose α to make this $\leq \epsilon$. It is easy to show that $\forall y \in (0, \frac{1}{4}]$, if $x \geq \frac{2}{y} \lg \frac{1}{y}$, then $\frac{\lg x}{x} \leq y$. So it is sufficient to choose $32\alpha = \frac{2 \cdot 32K^2}{\epsilon} \lg \frac{32K^2}{\epsilon}$, which is how α was chosen. $\square$

Lemma 5.14. *If ϕ is sparsified, then each disjunction of h literals occurs $\leq O(\frac{k}{\epsilon})^{3(k-h)}$ times in ϕ .*

Proof. By lemma 5.6, the frequency of a subexpression of size h is

$$\begin{aligned} &\leq \sum_{c=0}^{k-h} \theta_c = 2 + \alpha \sum_{c=1}^{k-h} \beta_c \\ &\leq 2 + 4\alpha(32\alpha)^{k-h-1} \quad \text{by theorem 5.21} \\ &= O\left(\frac{k^2}{\epsilon} \lg \frac{k}{\epsilon}\right)^{k-h} \leq O\left(\frac{k}{\epsilon}\right)^{3(k-h)}. \end{aligned} \tag{5.4}$$

The last inequality is slightly loose to get a simpler expression. $\square$

Of course we can also sparsify for $k = 2, h = 1$, in which case, we can use the second to last expression in (5.4) to bound the sparsification constant by $O(\frac{1}{\epsilon} \lg \frac{1}{\epsilon})$. Summarizing,

Theorem 5.15 (Sparsification). $\exists$ algorithm $A \forall k \geq 2, \epsilon \in (0, 1], \phi \in k\text{-CNF}$ with n variables, $A_{k,\epsilon}(\phi)$ outputs a sequence $\phi_1, \dots, \phi_s \in k\text{-CNF}$ with $n^{O(k)}$ delay such that

1. $s \leq 2^{\epsilon n}$
2. $\text{sol}(\phi) = \bigcup_i \text{sol}(\phi_i)$
3. $\forall i \in [s]$ each disjunction of h literals occurs $\leq O(\frac{k}{\epsilon})^{3(k-h)}$ times in ϕ_i , which also implies
 - (a) each literal occurs $\leq O(\frac{k}{\epsilon})^{3k}$ times, or $\leq O(\frac{1}{\epsilon} \lg \frac{1}{\epsilon})$ times for $k = 2$, in ϕ_i (this is our new sparsification constant)
 - (b) $|\phi_i| \leq O(\frac{k}{\epsilon})^{3k} n$.

It is sometimes more useful to have a *randomized* algorithm B with runtime $n^{O(k)}$ and with probability $\geq 2^{-\epsilon n}$ of outputting any particular ϕ_i . We now describe a generic procedure for transforming a backtracking algorithm such as A into an equivalent randomized algorithm B with an essentially optimal tradeoff between speed and success probability. This is a special case of a more general technique that applies to any backtracking algorithm whose running time is bounded by a recurrence relation.

Consider the following algorithm that returns a leaf of a full binary tree given a starting node.

```

B(node  $x, n \in \mathbb{N}, r \in \mathbb{N}$ )
  while  $x$  is not a leaf
     $p \leftarrow \frac{|B_r(0^{n-1})|}{|B_r(0^n)|}$ 
     $(x, n, r) \leftarrow \begin{cases} (x.\text{left}, n-1, r) & \text{with probability } p \\ (x.\text{right}, n-1, r-1) & \text{with probability } 1-p. \end{cases}$ 
  return  $x$ 

```

Theorem 5.16. *Let T be a full binary tree with root R such that each root-to-leaf path has length $\leq n$ and $\leq r$ right edges. Then for each leaf l of T ,*

$$Pr(B(R, n, r) = l) \geq \frac{1}{|B_r(0^n)|}, \quad (5.5)$$

and this bound is tight in the sense that $\forall n, r \exists T$ satisfying the hypothesis so that $\forall$ leaf l of T , (5.5) is an equality.

Proof. We prove (5.5) by induction on n . If $n = 0$ or $r = 0$ or $l = R$, the conclusion is immediate, so suppose $n, r \geq 1, l \neq R$. The identity $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ can be used to prove

$$|B_r(0^n)| = |B_r(0^{n-1})| + |B_{r-1}(0^{n-1})|,$$

which implies that if $p = \frac{|B_r(0^{n-1})|}{|B_r(0^n)|}$, then $1 - p = \frac{|B_{r-1}(0^{n-1})|}{|B_r(0^n)|}$.

B maintains the invariant that for the subtree rooted at x , each root-to-leaf path has length $\leq n$ and $\leq r$ right edges, so the induction hypothesis tells us that the conditional probability that B outputs l given that the first move is left (right) is $\geq \frac{1}{|B_r(0^{n-1})|}$ ($\geq \frac{1}{|B_{r-1}(0^{n-1})|}$). So if l is in the left subtree of R , then

$$\begin{aligned} Pr(B(R, n, r) = l) &= p \cdot Pr(B(R, n, r) = l \mid \text{first move is left}) \\ &\geq \frac{|B_r(0^{n-1})|}{|B_r(0^n)|} \cdot \frac{1}{|B_r(0^{n-1})|} \\ &= \frac{1}{|B_r(0^n)|}, \end{aligned}$$

and if l is in the right subtree of R , then

$$\begin{aligned} Pr(B(R, n, r) = l) &= (1 - p) \cdot Pr(B(R, n, r) = l \mid \text{first move is right}) \\ &\geq \frac{|B_{r-1}(0^{n-1})|}{|B_r(0^n)|} \cdot \frac{1}{|B_{r-1}(0^{n-1})|} \\ &= \frac{1}{|B_r(0^n)|}. \end{aligned}$$

To prove tightness, take T to be any decision tree representing $B_r(0^n)$. □

So we can transform A as follows: we add variables d, r to keep track of the number of branches and right branches, and instead of taking both left and right branches, we branch left with probability p and right with probability $1 - p$. The new algorithm is in figure 5.2.

Putting together lemmas 5.11, 5.12, theorem 5.5, and the proof of lemma 5.13, the probability that $B_{k,\epsilon}(\phi)$ outputs a particular output of $A_{k,\epsilon}(\phi)$ is $\geq 2^{-\epsilon n}$. Summarizing,

Theorem 5.17 (Randomized sparsification). $\exists$ randomized algorithm $B \forall k \geq 2, \epsilon \in (0, 1], \phi \in k\text{-CNF}$ with n variables, $\exists \phi_1, \dots, \phi_s \in k\text{-CNF}$ that are sparsified in the same sense as in theorem 5.15, $B_{k,\epsilon}(\phi)$ outputs some ϕ_i in time $n^{O(k)}$, and $\forall i \in [s] \ Pr(B_{k,\epsilon}(\phi) = \phi_i) \geq 2^{-\epsilon n}$.

```

 $B_{k,\epsilon}(\phi \in k\text{-CNF})$ 
 $(d, r) \leftarrow (\beta_{k-1}n, \lfloor \frac{k-1}{\alpha}n \rfloor)$ 
 $\phi \leftarrow \text{red}(\phi)$ 
while  $\exists(c, h) \phi_{c,h} \neq \emptyset$ 
  select the lexicographically first  $(c, h)$ 
  (meaning smallest  $c$ , then largest  $h$ ) such that  $\phi_{c,h} \neq \emptyset$ 
  select an arbitrary  $H \in \phi_{c,h}$ 
   $P \leftarrow \{C - H \mid H \subseteq C \in \phi_c\}$ 
   $p \leftarrow \frac{|B_r(0^{d-1})|}{|B_r(0^d)|}$ 
   $(\phi, d, r) \leftarrow \begin{cases} (\text{red}(\phi \cup \{H\}), d-1, r) & \text{w/prob } p \\ (\text{red}(\phi \cup P), d-1, r-1) & \text{w/prob } 1-p \end{cases}$ 
return  $\phi$ 

```

Figure 5.2: randomized sparsification algorithm

5.5 Open problems

Can we reduce the sparsification constant to $(O(\frac{1}{\epsilon}))^k$? Is it possible to eliminate the dependence of the sparsification constant on ϵ ?

5.6 Rate of growth of the θ_i

Recursively define the sequence

$$\begin{aligned}
 c_0 &= \frac{1}{4} \\
 c_n &= \sum_{j=0}^{n-1} c_{n-1-j} c_j
 \end{aligned} \tag{5.6}$$

The purpose of this section is to bound c_n , but attacking directly with induction is difficult. We will instead exploit the relationship between convolution and polynomial multiplication and use some heavy theorems from analysis.

Define the power series

$$f(z) = \sum_{n=0}^{\infty} c_n z^n.$$

Lemma 5.18.

$$f(z)^2 z + \frac{1}{4} = f(z). \tag{5.7}$$

Notice that at this point we only claim equality of formal power series, not that $f(z)$ converges for any particular z .

Proof.

$$\begin{aligned}
 f(z)^2 z + \frac{1}{4} &= \sum_{j=0}^{\infty} c_j z^j \sum_{k=0}^{\infty} c_k z^k z + \frac{1}{4} \\
 &= \sum_{n=0}^{\infty} \sum_{j=0}^n c_{n-j} c_j z^{n+1} + \frac{1}{4} \\
 &= \sum_{n=1}^{\infty} \sum_{j=0}^{n-1} c_{n-1-j} c_j z^n + \frac{1}{4} \\
 &= \sum_{n=0}^{\infty} c_n z^n = f(z).
 \end{aligned}$$

□

The implicit function theorem shows that there is a unique function g satisfying (5.7) on an open disk D about 0 and with range in an open disk E about $\frac{1}{4}$, and that g is continuous on D . The quadratic formula gives 2 functions satisfying (5.7):

$$\begin{aligned}
 g_1(z) &= \begin{cases} \frac{1-\sqrt{1-z}}{2z} & \text{if } z \neq 0 \\ \frac{1}{4} & \text{if } z = 0 \end{cases} \\
 g_2(z) &= \begin{cases} \frac{1+\sqrt{1-z}}{2z} & \text{if } z \neq 0 \\ \frac{1}{4} & \text{if } z = 0 \end{cases}
 \end{aligned}$$

Wlog suppose that the radius of D is ≤ 1 and that the radius of E is $\leq \frac{1}{4}$. Since $\forall z \in D - \{0\}$ $g_2(z) \notin E$, we conclude that $g = g_1$.

Up to now, we have only provided motivation for studying g ; now we will actually analyze it.

In analogy with the binomial theorem, we have that for functions a, b that are n times differentiable at z ,

$$(ab)^{(n)} = \sum_{j=0}^n \binom{n}{j} a^{(j)} b^{(n-j)}. \quad (5.8)$$

This is easily proved by induction.

We can compute the n th derivative of g at $z = 0$ for $n \geq 1$ by applying (5.8) twice to $g = g^2 z + \frac{1}{4}$:

$$\begin{aligned}
 g^{(n)} &= \left(g^2 z + \frac{1}{4} \right)^{(n)} \\
 &= (g^2)^{(n)} z + n(g^2)^{(n-1)} \\
 &= \sum_{j=0}^n \binom{n}{j} g^{(j)} g^{(n-j)} z + n \sum_{j=0}^{n-1} \binom{n-1}{j} g^{(j)} g^{(n-1-j)}.
 \end{aligned} \quad (5.9)$$

Notice that $g^{(n)}$ appears on both the LHS and the RHS. So this implicitly defines $g^{(n)}(z)$ provided $1 - 2g(z)z \neq 0$, and this happens iff $z \neq 1$. We conclude that g is analytic on $\mathbb{C} - \{1\}$.

Using (5.9), we have

$$g^{(n)}(0) = n \sum_{j=0}^{n-1} \binom{n-1}{j} g^{(j)}(0) g^{(n-1-j)}(0).$$

Dividing by $n!$ gives

$$\frac{g^{(n)}(0)}{n!} = \sum_{j=0}^{n-1} \frac{g^{(j)}(0)}{j!} \frac{g^{(n-1-j)}(0)}{(n-1-j)!},$$

which shows that $\frac{g^{(n)}(0)}{n!}$ solves the recurrence (5.6). So f is the power series representation of g about the point 0 and the radius of convergence is 1. (see e.g. theorem 10.16 of [Rud87]) Notice that g does *not* have a derivative at 1, although g is continuous on all of $\mathbb{C}$.

To prove convergence of f at the radius of convergence, we first prove a technical lemma.

Lemma 5.19. *Let $s_n : (a, b] \rightarrow \mathbb{R}$ be a sequence of functions, each continuous at b , and such that $s_n(x)$ is nondecreasing in both x and n . Also let $g : (a, b] \rightarrow \mathbb{R}$ be continuous at b . Suppose that*

$$\forall x \in (a, b) \quad \lim_{n \rightarrow \infty} s_n(x) = g(x).$$

Then

$$\lim_{n \rightarrow \infty} s_n(b) = g(b).$$

Proof. ($\leq$) Suppose indirectly that $\lim_{n \rightarrow \infty} s_n(b) > g(b)$ or that the LHS does not converge. Since $s_n(x)$ is nondecreasing in n , the LHS converges to a number $> g(b)$ or properly diverges to ∞ . So $\exists \epsilon > 0, n_0$ $s_{n_0}(b) \geq g(b) + \epsilon$.

Since s_{n_0} and g are continuous at b , $\exists b' \in (a, b)$ $(s_{n_0}(b') \geq s_{n_0}(b) - \frac{\epsilon}{4} \wedge g(b') \leq g(b) + \frac{\epsilon}{4})$.

So

$$\begin{aligned} g(b) &\geq g(b') - \frac{\epsilon}{4} \\ &= \lim_{n \rightarrow \infty} s_n(b') - \frac{\epsilon}{4} \\ &\geq s_{n_0}(b') - \frac{\epsilon}{4} && \text{since } s_n(x) \text{ is nondecreasing in } n \\ &\geq s_{n_0}(b) - \frac{\epsilon}{2} \\ &\geq g(b) + \frac{\epsilon}{2}, \end{aligned}$$

a contradiction.

($\geq$) Suppose indirectly that $\lim_{n \rightarrow \infty} s_n(b) < g(b)$. We already showed that the LHS converges. So $\exists \epsilon > 0$ $\lim_{n \rightarrow \infty} s_n(b) + \epsilon \leq g(b)$.

Since g is continuous at b , $\exists b' \in (a, b)$ $g(b') \geq g(b) - \frac{\epsilon}{2}$. So

$$\begin{aligned}
 g(b) &\geq \lim_{n \rightarrow \infty} s_n(b) + \epsilon \\
 &\geq \lim_{n \rightarrow \infty} s_n(b') + \epsilon && \text{since } s_n(x) \text{ is nondecreasing in } x \\
 &= g(b') + \epsilon \\
 &\geq g(b) + \frac{\epsilon}{2},
 \end{aligned}$$

a contradiction. □

Lemma 5.20.

1. $\sum_{n=0}^{\infty} c_n = \frac{1}{2}$
2. $c_n \in o(1)$
3. $\forall \alpha < 1$ $c_n \notin O(\alpha^n)$.

Proof. We apply lemma 5.19 with $s_n(x) = \sum_{j=0}^n c_j x^j$ and $a = 0, b = 1$ to conclude that $f(1)$ converges to $g(1) = \frac{1}{2}$. This also implies that the n th term of the power series converges to 0 as $n \rightarrow \infty$. So $c_n \in o(1)$.

Next, suppose indirectly that $\exists \alpha < 1$ $c_n \in O(\alpha^n)$. Then $\frac{c_n}{\alpha^n} \in O(1)$. So $\forall \beta \in (\alpha, 1)$ $f(\frac{1}{\beta})$ converges, but this contradicts that the radius of convergence of f is 1. □

Theorem 5.21. Fix $a, b > 0$ and define d_n recursively by

$$\begin{aligned}
 d_0 &= a \\
 d_n &= \sum_{j=0}^{n-1} b d_{n-1-j} d_j.
 \end{aligned}$$

Then

1. $\forall n \geq 0$ $\sum_{j=0}^n d_j \leq 2a(4ab)^n$
2. $d_n \in o((4ab)^n)$
3. $\forall \alpha < 4ab$ $d_n \notin O(\alpha^n)$.

Proof. It is easily seen by induction on n that $d_n = (4a)(4ab)^n c_n$. Now apply lemma 5.20. □

Acknowledgements

Material here is joint work with Russell Impagliazzo and Ramamohan Paturi and appeared in [CIP06].

Chapter 6

k -SAT upper bounds

There are many algorithms for k -SAT, most falling into one of several classes, depending on what problem they are designed to solve, how they work, what they guarantee. For example, some algorithms are designed to work best on instances drawn randomly from a particular distribution, while others are tuned to a particular 'real world' problem domain, such as bounded model checking, circuit fault analysis, or planning. Also, some algorithms employ heuristics not proven to have a particular performance benefit, but which incorporate features of other algorithms with rigorously proven upper bounds (e.g. zChaff, UnitWalk, WalkSAT, survey propagation). See [Cal04] for a high level discussion of such algorithms.

In this chapter we will concern ourselves only with algorithms that (1) attempt to find some solution to a k -CNF, as opposed to, say, counting or enumerating all of them, or discovering some other property of the solution space; and (2) have rigorously proven worst-case upper bounds and (3) are the current best, given some requirement, such as that the algorithm be deterministic. They fall into 2 categories.

- **Incremental assignment algorithms** maintain a partial solution at each step, initially empty. At each step, they choose a variable and deduce its value or branch on it, until a contradiction is reached or no more variables are left. Branching can take the form of randomized decision making or backtracking. The rules used to make deductions, choose the next variable, and backtrack all greatly impact performance.
- **Local search algorithms** maintain a full solution at each step, initially some guess. At each step, if not at a solution, they change the solution to some neighbor assignment (in the Hamming cube) based on some rule. They may restart occasionally if they guess that they started too far from a solution.

There is a way to see these 2 as minor variants of the same idea: consider an incremental assignment algorithm that, under some condition, guesses the value of a variable x_i randomly.

If we consider this randomness to come from the i th bit of an initially random string y , then the algorithm is actually performing a local search with y as its initial guess. We will take this perspective when describing the PPSZ algorithm later.

6.1 DPLL method

The original DPLL incremental assignment method

(Davis, Putnam, Logemann, Loveland) [DP60, DLL62] uses the following 2 rules of inference to decrease the amount of backtracking needed:

- **unit clause rule:** A *unit clause* $C \in F$ is a clause of size 1. If such a clause is discovered, we set its literal to 1 and simplify the formula by eliminating true clauses and false literals.
- **pure literal rule:** A literal l is *pure* in F iff $\bar{l}$ does not appear in F . In this case, F is satisfiable iff $F|l$ is satisfiable, so we set l to 1 and simplify F . Notice that we are not *forced* to set l to 1: $F|\bar{l}$ may also be satisfiable. So, e.g., application of this rule is a bad idea if one wanted to enumerate all solutions.

Applying either of these rules may generate more unit clauses or pure literals, and so the effects can be propagated. Applying these rules repeatedly to a formula reaches a unique fixed point independent of the order in which the rules were applied. At that point, if the formula is $\emptyset$, it is satisfiable; if it is $\{\emptyset\}$, it is unsatisfiable. But otherwise, it contains at least one variable and no unit clauses nor pure literals, and there is no longer a free lunch. The algorithm then chooses a literal on which to branch. This branching causes the well-known exponential worst-case performance, as the other operations all take polynomial time.

The DPLL method leaves open how to choose the next branching variable, and so is not a complete algorithm, much as the Ford-Fulkerson method for network flow is only a family of algorithms since it does not specify how to choose an augmenting path. Also, it should be noted that the terms DPLL, DLL, DP are sometimes used to represent the entire class of incremental assignment methods, especially those that use either unit clause or pure literal rules.

6.2 PPZ algorithm

[PPZ99] give the DPLL algorithm of figure 6.1 to decide the satisfiability of $F \in k\text{-CNF}$.

PPZ_main assigns variables one at a time in the order determined by σ , applying the unit clause rule iff the current variable $x_{\sigma(i)}$ is in a unit clause, (in which case we say $x_{\sigma(i)}$ is *forced*) and simplifying F as it goes. Since $|F| \leq \text{poly}(n)$, the algorithm takes time $\text{poly}(n)$. By the end, F is either $\emptyset$, in the case that the assignment found is a solution, otherwise $F = \{\emptyset\}$. So it has 0 soundness error. Later we will show that the completeness error is $\leq 1 - 2^{-c_k n}$ for some $c_k < 1$, and so we can reduce the completeness error to e^{-1} by calling PPZ_main $2^{c_k n}$ times

```

PPZ_main( $F \in k\text{-CNF}$ )
  let  $\{x_1, \dots, x_n\} = \text{var}(F)$ 
  choose a random permutation  $\sigma$  of  $[n]$ 
  for  $i \leftarrow 1, \dots, n$ 
    if  $x_{\sigma(i)}$  is in a unit clause  $C \in F$  //  $x_{\sigma(i)}$  is forced
      set  $x_{\sigma(i)}$  so as to satisfy  $C$ 
    else set  $x_{\sigma(i)}$  uniformly at random
  simplify  $F$  by eliminating true clauses and false literals
  if  $F = \emptyset$ , return  $(x_1, \dots, x_n)$  // solution found
  else return “probably not satisfiable”

```

Figure 6.1: PPZ algorithm solving k -SAT

and taking the maximum answer. This iterated use of PPZ_main we will refer to as the PPZ algorithm.

The PPZ algorithm differs from the original DPLL method in that (1) it is randomized and iterative rather than backtracking, (2) does not bother to exploit a unit clause unless its variable is next in the permutation σ , and (3) it does not exploit pure literals at all. Of course, adding features (2), (3) would not hurt, but we don't know how to prove that they strictly help. Also, the randomness can be reduced, as we will show in §6.2.1. If there is an isolated solution a , so that $\forall i \in V$ $a \oplus i$ is not a solution, then the randomness can be completely removed while increasing the upper bound on the running time by only a polynomial factor. Also, the randomness can be completely removed if we are willing to settle for a running time of $\text{poly}(n)2^{(1-\frac{\alpha}{k})n}$ where $\alpha = \frac{1}{2} - \frac{1}{\Theta(\sqrt{k})}$.

We now explain the intuition behind the PPZ algorithm. Let $F \in k\text{-CNF}$, $V = \text{var}(F)$, $S = \text{sol}(F)$. For $a, b \in 2^V$, the *Hamming distance* between a, b is

$$d(a, b) = |\{i \in V \mid a(i) \neq b(i)\}|,$$

and the $|V|$ -dimensional *Boolean cube* is the graph

$$G = (2^V, \{\{a, b\} \mid d(a, b) = 1\}).$$

Let G_S be the subgraph of G induced by S and $\deg(a)$ be the degree of a in G_S . The *degree of isolation* $I(a) = n - \deg(a)$ of a node $a \in S$ is the number of non-solution neighbors of a in G , i.e. there are $I(a)$ variables i , each called a *direction of isolation*, such that $a \oplus i \notin S$. Since a is a solution of F but $a \oplus i$ is not, there must be a clause $C \in F$ whose only true literal at a involves variable i . Such a clause is said to be *critical* at a , and i is its *critical variable*. So at a , there are $I(a)$ critical variables i , each with its own critical clause C . Critical clauses will help to expose a to the algorithm, the more the merrier.

Let $f_\sigma(a)$ be the number of forced variables on a run of PPZ_main that uses the permutation σ and where the random assignments are made in agreement with a . The probability that PPZ_main finds $a \in S$ is then

$$\begin{aligned} E_\sigma(2^{-(n-f_\sigma(a))}) \\ \geq 2^{-n+E_\sigma(f_\sigma(a))} \quad \text{from Jensen's inequality.} \end{aligned} \quad (6.1)$$

To relate $E_\sigma(f_\sigma(a))$ to $I(a)$, suppose i is the critical variable of a critical clause C at a . Given that the random assignments made by the algorithm agree with a , if i comes last in the permutation σ among $\text{var}(C)$ then i will be forced. Since $F \in k\text{-CNF}$, the probability of this is $\geq \frac{1}{k}$, and so by linearity of expectation, $E_\sigma(f_\sigma(a)) \geq \frac{I(a)}{k}$. So the probability that PPZ_main finds a solution is

$$p_{\text{succ}} \geq \sum_{a \in S} 2^{-n + \frac{I(a)}{k}} = 2^{-(1-\frac{1}{k})n} \sum_{a \in S} 2^{-\frac{\deg(a)}{k}}. \quad (6.2)$$

[PPZ99] argued that if $S \neq \emptyset$, then

$$\sum_{a \in S} 2^{-\frac{\deg(a)}{k}} \geq \sum_{a \in S} 2^{-\deg(a)} \geq 1. \quad (6.3)$$

(The last inequality is not so obvious, but even the first inequality gives away more than necessary, as we will see shortly.) This implies that $p_{\text{succ}} \geq 2^{-(1-\frac{1}{k})n}$.

The isolation of a solution a helps to reveal a to the algorithm, but many solutions can be arranged so as to be neighbors of each other, reducing their degrees of isolation. On the other hand, the presence of many solutions reduces the chance of guessing the value of a variable incorrectly. (6.3) helps to sort out these opposing forces, implying that they at least balance out in our favor. [CIKP08] went further, showing that more solutions *strictly* help (the analysis, at least).

Let's pick up the analysis where [PPZ99] left off. From Jensen's inequality,

$$\sum_{a \in S} 2^{-\frac{\deg(a)}{k}} \geq |S| 2^{-\frac{1}{k|S|} \sum_{a \in S} \deg(a)}. \quad (6.4)$$

$\sum_{a \in S} \deg(a) = |E(S, S)|$ is the number of directed edges in G from S to S . To develop some intuition about how big $E(S, S)$ might be, note that each of the n edges in G connected to a node in S either goes into S or $\bar{S}$. So

$$n|S| = |E(S, S)| + |E(S, \bar{S})|. \quad (6.5)$$

We would like to lower bound $|E(S, \bar{S})|$ as a function of $|S|$. Informally speaking, $|E(S, \bar{S})|$ is a measure of the surface area of the volume S , and the configuration that minimizes surface area for a given volume ought to be a ball.

Suppose that $S = B_{pn}(0^n)$ is the ball of radius $pn \in \mathbb{N}$ where $p \in (0, \frac{1}{4})$ is a constant and n is large. The boundary ∂S of S is those assignments with Hamming weight exactly pn .

An edge $(a, b) \in E(S, \bar{S})$ iff $a \in \partial S$ and b is $(a$ but with some 0 bit flipped). So there are $(1-p)n$ edges in $E(S, \bar{S})$ for each element of ∂S . So

$$\frac{|E(S, \bar{S})|}{|S|} = (1-p)n \frac{|\partial S|}{|S|}.$$

$\forall i \in [0, pn]$,

$$\frac{\binom{n}{i-1}}{\binom{n}{i}} = \frac{i}{n-i+1} \leq \frac{p}{1-p}.$$

So

$$\begin{aligned} \frac{|\partial S|}{|S|} &= \frac{\binom{n}{pn}}{\sum_{i=0}^{pn} \binom{n}{i}} \\ &\geq \frac{1}{\sum_{i=0}^{pn} \left(\frac{p}{1-p}\right)^i} \\ &\geq 1 - \frac{p}{1-p} = \frac{1-2p}{1-p}. \end{aligned}$$

So

$$\frac{|E(S, \bar{S})|}{|S|} \geq (1-2p)n.$$

From theorems 2.3, 2.2, and since $p < \frac{1}{4}$, we have $\lg |S| \geq h(p)n - o(n) \geq p \lg \frac{1}{p} n - o(n) \geq 2pn$, for sufficiently large n . So

$$\frac{|E(S, \bar{S})|}{|S|} \geq n - \lg |S|.$$

From (6.5),

$$\frac{|E(S, S)|}{|S|} \leq \lg |S|. \quad (6.6)$$

Substituting this back into (6.4) we get

$$\sum_{a \in S} 2^{-\frac{\deg(a)}{k}} \geq |S|^{1-\frac{1}{k}}.$$

Substituting this back into (6.2) gives $p_{\text{succ}} \geq \left(\frac{|S|}{2^n}\right)^{1-\frac{1}{k}}$.

Of course, everything strictly between (6.5) and (6.6) was not general, but just an intuition builder. To be general, we use Harper's edge isoperimetric inequality [Har67] which says that (6.6) holds for any nonempty set S of nodes of G . (So we don't need to assume that S is a ball, nor that p is a constant $< \frac{1}{4}$, nor that n is large.) Summing up,

Theorem 6.1 ([CIKP08]). *Given $F \in k\text{-SAT}$, $S = \text{sol}(F)$, The probability that PPZ_{main} finds a solution is*

$$p_{\text{succ}} \geq \left(\frac{|S|}{2^n}\right)^{1-\frac{1}{k}}.$$

This is very satisfying because it so closely matches our intuition: the time it takes PPZ to find the unique solution in a space of size a is $\leq a^{1-\frac{1}{k}}$, but if there are s solutions, then the run time is the same as if it were searching for the unique solution in a space of size $\frac{a}{s}$.

6.2.1 Derandomization of PPZ

We show 2 ways to derandomize PPZ. Both will make use of the fact that we only need the permutation σ to be approximately k -wise independent, in the sense that the ordering it induces on each subset of $\leq k$ variables is approximately uniformly random. The 1st way will solve k -SAT with the promise that the input has a fully isolated solution and will give the same performance as randomized PPZ, up to a factor of $n^{O(k)}$. The 2nd way will solve k -SAT in general, but the exponential complexity will increase from $1 - \frac{1}{k}$ to $1 - \frac{p}{k}$ where $p = \frac{1}{2} - \frac{1}{\Theta(\sqrt{k})}$.

We'll give a presentation slightly different from that of [PPZ99]. Let $F \in k$ -CNF, $n = |\text{var}(F)|$. Suppose $N \in \mathbb{N}$ is a power of 2 and $N \geq n$. From theorem 2.8, there is a probability space S of size N^k on which we can define k -wise independent random variables $X_1, \dots, X_n$, each uniform over N . We would like to construct a k -wise independent permutation from the X_i . If $k \geq 2$, then the probability that

$$\exists i, j \in [n], i \neq j \text{ } X_i = X_j \quad (6.7)$$

is $\leq \binom{n}{2} \frac{1}{N}$. So we can usually get a permutation σ from the X_i by defining $\sigma(i) = |\{j \in [n] \mid X_j \leq X_i\}|$, provided $N \gg \binom{n}{2}$. The problem is that conditioning on the event that (6.7) does not occur may disturb the k -wise independence.

We get around this by relaxing the requirement that the ordering induced on each subset of $\leq k$ variables be perfectly uniform. The X_i will determine an ordering that may have ties. Break the ties arbitrarily, say, by having the variable with the smallest index come first, and call this permutation σ . Then the probability that a particular variable in a set $\{x_{i_1}, \dots, x_{i_{k'}}\}$ of size $k' \leq k$ comes last is

$$\geq \frac{1}{k'} \Pr(X_{i_1}, \dots, X_{i_{k'}} \text{ are distinct}) \geq \frac{1}{k} \left(1 - \frac{k^2}{N}\right). \quad (6.8)$$

Let $a \in \text{sol}(F)$ be j -isolated so that $I(a) = j$. By linearity of expectation, $E_\sigma(f_\sigma(a)) \geq \frac{j}{k} (1 - \frac{k^2}{N}) \stackrel{\text{def}}{=} j'$. So there must exist a *good* permutation, one for which $f_\sigma(a) \geq j'$. If σ is good, then we can find a as follows: for each string r of $n - j'$ bits, run PPZ_main using σ and using r for randomness when guessing how to set variables. We are effectively *decoding* a from r (hence the title of [PPZ99]: “Satisfiability Coding Lemma”).

Letting N be the smallest power of 2 that is $\geq n^2$, this procedure will find *all* j -isolated solutions of F by iterating over all (σ, r) , of which there are $\leq (2n^2)^k 2^{n - \frac{j}{k} + 1}$. Decoding each (σ, r) takes time $O(n|F|)$, so we have the following.

Theorem 6.2. *Let $F \in k$ -CNF, $n = |\text{var}(F)|$. The j -isolated solutions of F can be deterministically enumerated in time $O(|F|n^{2k+1}2^{n - \frac{j}{k}})$.*

In particular, if F has a fully isolated solution, it can be found deterministically in time $\text{poly}(n)2^{(1 - \frac{1}{k})n}$. We can also use this idea to fully derandomize PPZ, but instead of saving 1 variable in k over the running time of brute force, we save only about 1 variable in $2k$.

Theorem 6.3. *k -SAT can be solved deterministically in time $O(|F|n^{2k+1}2^{(1-\frac{p}{k})n})$ where $p = \frac{1}{2} - \frac{1}{\Theta(\sqrt{k})}$.*

Proof. Let $F \in k\text{-CNF}$, $V = \text{var}(F)$, $n = |V|$, $S = \text{sol}(F)$. For a constant $p < \frac{1}{2}$ depending only on k to be chosen later, let $B = B_{pn}(c)$ be the ball of radius pn about an arbitrary center $c \in 2^V$. By theorem 2.3, $|B| \leq 2^{h(p)n}$. Either F has a solution in B , in which case we can find it by exhaustively searching B , or it does not, in which case, let a be a solution closest to c . Then for each variable i where a disagrees with c , we have $a \oplus i \notin S$. So $I(a) > pn$. By theorem 6.2, we can find a deterministically in time $O(|F|n^{2k+1}2^{(1-\frac{p}{k})n})$.

To minimize the time taken by this combined algorithm, we choose p to be the unique solution $< \frac{1}{2}$ of $h(p) = 1 - \frac{p}{k}$. Since h is increasing and $1 - \frac{p}{k}$ is decreasing on $[0, \frac{1}{2}]$ we can upper bound p by any $p_1 < \frac{1}{2}$ that satisfies $h(p_1) \geq 1 - \frac{p_1}{k}$ and lower bound p by any $p_0 < \frac{1}{2}$ that satisfies $h(p_0) \leq 1 - \frac{p_0}{k}$ (see figure 2.1). By theorem 2.2, $\forall q < \frac{1}{2}$ $h(q) \in [1 - 4(q - \frac{1}{2})^2, 1 - 2(q - \frac{1}{2})^2]$. So if p_1 satisfies $1 - 4(p_1 - \frac{1}{2})^2 = 1 - \frac{p_1}{k}$, then $h(p_1) \geq 1 - \frac{p_1}{k}$. $p_1 = \frac{1}{2} - \frac{1}{\sqrt{8k+1+1}}$ suffices. Similarly, if p_0 satisfies $1 - 2(p_0 - \frac{1}{2})^2 = 1 - \frac{p_0}{k}$, then $h(p_0) \leq 1 - \frac{p_0}{k}$. $p_0 = \frac{1}{2} - \frac{1}{\sqrt{4k+1+1}}$ suffices. So $p \in [p_0, p_1]$. $\square$

As a potentially interesting application of theorem 6.2, consider the problem Conn- k -SAT of deciding whether the solution space of a k -CNF is connected [GKMP09]. Suppose that for some p depending only on k , the solution space of $F \in k\text{-CNF}$ is connected iff the pn -isolated solutions S of F are connected. (Just to make this seem more plausible, consider that the pn -isolated solutions are in some sense the *surface* of the solution space, and the connectivity of a set of subsets of $\mathbb{R}^n$ each homeomorphic to a ball and none containing another is equivalent to the connectivity of their surfaces.) We could use theorem 6.2 to find S . If the number of edges between nodes in S is small and they can be efficiently enumerated, then we could test connectivity by using DFS. This would show that Conn- k -SAT has upper bounds of the same form as k -SAT currently has, and so in some sense isn't too much harder, despite the fact that k -SAT is in NP while Conn- k -SAT is PSPACE-complete. (To be clear, this is just conjecture, but worthy of some future investigation.)

6.3 PPSZ

[PPSZ05] improved upon the PPZ algorithm by performing a limited amount of resolution as a preprocessing step and then invoking PPZ. This improves the exponential complexity from $1 - \frac{1}{k}$ to $1 - \frac{\mu_k}{k-1}$ where $\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j+\frac{1}{k-1})}$, at least, for $k \geq 5$ or ($k \geq 3$ and the input is promised to be uniquely satisfiable, i.e. has ≤ 1 solution). For $k \in \{3, 4\}$ in the general case, the expressions for the exponential complexities are more complex: approximately .5207 for $k = 3$, and .5625 for $k = 4$.

To make it easier to compare these with other results, we make a few remarks about μ_k : μ_k is an increasing function of k , $\mu_3 = 4 - 4 \ln 2 > 1.226$, $\mu_4 \approx 1.3354$, $\lim_{k \rightarrow \infty} \mu_k = \frac{\pi^2}{6} > 1.644$, to approximate μ_k to within $\frac{1}{l}$ it is sufficient to add the first l terms of the sum since the tail $\sum_{j=l+1}^{\infty} \frac{1}{j(j+\frac{1}{k-1})} \leq \sum_{j=l+1}^{\infty} \frac{1}{j^2} \leq \int_l^{\infty} \frac{1}{x^2} dx = \frac{1}{l}$.

We now describe the intuition behind PPSZ, although in far less detail than for PPZ. Let $F \in k\text{-CNF}$, $V = \text{var}(F)$, $S = \text{sol}(F)$, and suppose that $\text{PPZ_main}(F)$ guesses each variable in agreement with some $a \in S$. Each variable $v \in V$ is forced iff there is some clause $C \in F$ which is critical for v at a and such that v occurs last in the permutation σ among $\text{var}(C)$. The analysis of [PPZ99] only assumed that there was a single critical clause for each direction of isolation of a . [PPSZ05] observed that we can increase the number of critical clauses for v substantially if we add the following assumptions: (1) F is closed under a certain limited form of resolution, and (2) a is isolated in the sense that no other solution has small Hamming distance from a .

To be more precise, say clauses A, B are *s-bounded resolvable* iff $|A|, |B|, |\text{resolve}(A, B)| \leq s$. *s-bounded resolution* is the process of adding clauses to a formula until it is closed under such resolution. Also, say $a \in S$ is *distance d isolated* iff the Hamming ball $B_d(a)$ intersects S only at a . [PPSZ05] show that given $F \in k\text{-CNF}$, if we first perform *s-bounded resolution* and let a be a distance d isolated solution, then the probability that each variable is forced in PPZ_main is $p \geq \frac{\mu_k}{k-1} - o(1)$ (where $o(1)$ represents a function of n) provided $d = \lfloor \log_k s \rfloor$ and $s \leq o(\lg n)$. From (6.1), the probability that PPZ finds a is then $\geq 2^{-(1 - \frac{\mu_k}{k-1} + o(1))n}$. Since s grows so slowly, the time required to perform *s-bounded resolution* is only $2^{o(n)}$, and so the exponential complexity of PPSZ is $1 - \frac{\mu_k}{k-1}$.

The key idea to proving this lower bound on the forcing probability is a structure called a *critical clause tree*. The details of this structure are not so important for this discussion, but to each variable v we associate a set of $\leq s$ variables, and provided that the ordering on these $\leq s$ variables is uniformly random, then the probability that v is forced is $\geq p$.

6.3.1 Derandomization of PPSZ, unique solution case

We can derandomize PPSZ exactly as we did for PPZ by using exactly the same distribution to generate σ as we used when derandomizing PPZ, only now we need *s-wise independence*. I.e. let N be a power of 2 with $N \geq n^2$ and use theorem 2.8 to generate a sequence of *s-wise independent* random variables $X_1, \dots, X_n$ each uniform over N . Let the order of the X_i determine a permutation σ , breaking ties arbitrarily. The sample space of this distribution has size $2^{o(n)}$ and the forcing probability is no longer exactly p , but is $\geq p(1 - \frac{s^2}{N}) = p - o(n^{-1})$ (using the same reasoning as for (6.8)), which is good enough to give the same exponential complexity as the randomized PPSZ. The only rub here is that we needed to assume that there was a distance d isolated solution. Certainly this happens if F is uniquely satisfiable, so we have the following.

Theorem 6.4. $dc_{\text{Unique-}k\text{-SAT}} \leq 1 - \frac{\mu_k}{k-1}$ where $\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j+\frac{1}{k-1})}$.

[Rol05a] used a similar (albeit slightly more complex) idea to derandomize PPSZ for the uniquely satisfiable case as well.

To upper bound the run time of PPSZ without the assumption that there is a distance d isolated solution requires substantially more complex logic, which we only sketch here. A *subcube* $B \subseteq 2^V$ of the Boolean cube is the set of all extensions to V of some partial assignment $a \in 2^{D_B}$ where $D_B \subseteq V$ (and clearly a will be unique). D_B and $N_B = V - D_B$ are called the *defining* and *nondefining* variables of B . Let $F \in k\text{-CNF}$ be the input formula, assumed to be closed under s -bounded resolution, and let $S = \text{sol}(F) \neq \emptyset$. A simple induction on $|V|$ shows that S can be partitioned into a set of subcubes $\mathcal{B}$ of the Boolean cube such that each subcube $B \in \mathcal{B}$ contains exactly 1 solution $z_B \in S$. z_B is distance $|N_B|$ isolated since it is the only solution in B . [PPSZ05] show that by conditioning on the event that the variables of D_B come “early” in the permutation σ , then the probability that each of the variables in N_B is forced is at least that for the unique solution case.

Of course we must pay a price for assuming that the variables of D_B come early, but it turns out that this is more than made up for by the fact that the defining variables in some sense cannot be guessed wrong. To be a little more formal, let $y \in_u 2^V$ and suppose that when guessing variable $i \in V$, PPZ_main uses y_i for randomness. Then

$$\begin{aligned} \Pr(\text{PPZ_main}(F) \in S) &= \sum_{B \in \mathcal{B}} \Pr(\text{PPZ_main}(F) = z_B \mid y \in B) \Pr(y \in B) \\ &\geq \min_{B \in \mathcal{B}} \Pr(\text{PPZ_main}(F) = z_B \mid y \in B). \end{aligned} \tag{6.9}$$

So it suffices to lower bound the probability of successfully finding a solution z_B given that the defining variables for B are set correctly.

To actually lower bound this expression requires a great deal of work and is beyond our scope, but one can already see from the high level logic the problems with derandomizing PPSZ for the general case. First, the analysis lower bounds the RHS of (6.9) by conditioning on the defining variables, of which there could be $\Omega(n)$, “coming early”. If we replaced σ by a distribution with only limited independence, then such a conditioning could disturb the limited independence for the ordering of the remaining variables. But even if that could somehow be overcome and we could significantly reduce the size of the sample space from which σ is drawn, already more than $(1 - \frac{\mu_k}{k-1})n$ many random bits from y were used in what little of the analysis we’ve shown here. Another way of seeing this is that the algorithm has no idea which are the defining variables for the solution it’s heading towards (even though the analysis can make use of this information), so it cannot preferentially use full randomness when a nondefining variable comes up and use less otherwise.

At the moment, there is no known efficient derandomization of PPSZ for the general case.

6.4 Local search

Schöning [Sch99] gave a radically different idea for solving k -SAT, but with performance guarantees similar to that of PPZ. In fact, this algorithm easily generalizes to (k, d) -CSP-SAT.

On input $F \in k$ -CNF, the algorithm begins by choosing an initial assignment a at random. If this is a solution, we are done. Otherwise, there is a false clause C , we flip some variable in C , and check again. This continues for $3n$ steps, where $n = |\text{var}(F)|$ (see figure 6.2). This is similar to walking in the woods with a compass whose reliability is poor, but far better than nothing. It is interesting to note that, as far as the analysis in [Sch99] is concerned, continuing searching after $O(n)$ steps becomes less profitable than simply restarting.

```

LocalSearch_main( $F \in k$ -CNF)
  if  $\emptyset \in F$ , return “not satisfiable”
   $V \leftarrow \text{var}(F)$ 
   $n \leftarrow |V|$ 
  choose  $a \in_u 2^V$ 
  if  $F(a) = 1$ , return  $a$ 
  do  $3n$  times
    choose  $C \in \{C \in F \mid C(a) = 0\}$  arbitrarily
    choose  $i \in_u \text{var}(C)$ 
     $a \leftarrow a \oplus i$ 
  if  $F(a) = 1$ , return  $a$ 
  return “probably not satisfiable”

```

Figure 6.2: Schöning’s local search algorithm

[Sch99] lower bounds the probability that LocalSearch_main finds some solution when $F \in \text{SAT}$ by some p_{succ} , and so we can boost this probability to $\geq e^{-1}$ by repeating p_{succ}^{-1} times, which is sometimes called local search with restarts - we will just call it LocalSearch. The argument proceeds as follows. Let $b \in \text{sol}(F)$ be arbitrary and a_t be the value of a at step t . Then $a_0, \dots, a_{3n}$ is a Markov chain, but one that carries too much irrelevant information. Letting d_t be the Hamming distance between a_t and b , we want to lower bound the probability p_{succ} that for some $t \leq 3n$, $d_t = 0$. The sequence $d_0, \dots, d_{3n}$ is no longer necessarily a Markov chain, but we notice that (1) d_0 is binomial $(n, \frac{1}{2})$ and (2) if $d_t > 0$ and $t < 3n$, then $\Pr(d_{t+1} = d_t - 1) \geq \frac{1}{k}$, so we can lower bound p_{succ} in the following way.

Say $x_0, x_1, \dots$ is a (d, p) random walk iff $x_0 = d$ and each $x_t = x_0 + \sum_{i=1}^t X_i$ where $X_1, X_2, \dots$ are independent and identically distributed random variables with $\Pr(X_i = -1) =$

$p, Pr(X_i = 1) = 1 - p$. Define

$$p_{\text{succ},d,p} = Pr(\exists t \leq 3n \ x_t = 0)$$

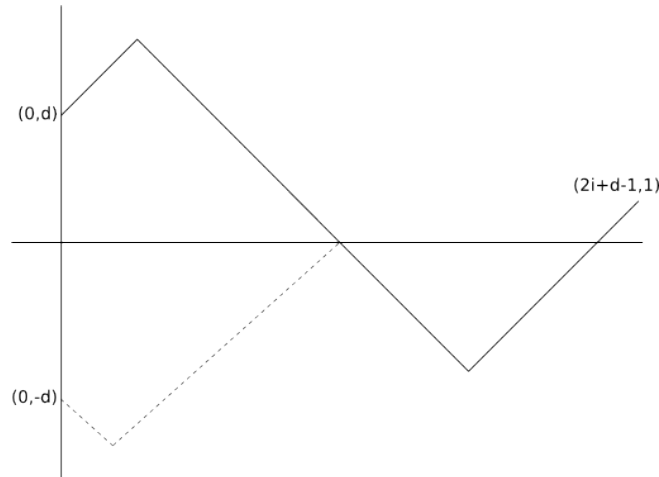
where $x_0, x_1, \dots$ is a (d, p) random walk. Then $p_{\text{succ}} \geq E_d(p_{\text{succ},d,\frac{1}{k}})$ where d is a $(n, \frac{1}{2})$ binomial random variable.

We now lower bound $p_{\text{succ},d,\frac{1}{k}}$. Notice that, trivially, we have $p_{\text{succ},d,\frac{1}{k}} \geq k^{-d}$; but by using the *reflection principle* (a general technique also capable of proving Bertrand's ballot theorem) we have the following improvement.

Lemma 6.5. $\forall k \geq 3, d \geq 0 \ p_{\text{succ},d,\frac{1}{k}} \geq \Omega(n^{-\frac{3}{2}})(k-1)^{-d}$.

Proof. The lemma holds if $d = 0$, so assume $d > 0$. Consider the digraph $G = (\mathbb{Z}^2, \{((t, j), (t+1, j')) \mid |j - j'| = 1\})$ of points on the integer lattice in the plane with edges from each (t, j) to $(t+1, j \pm 1)$. (t, j) represents the walk being in state j at time t . Each “down” (“up”) edge is labeled with $p = \frac{1}{k}$ (or $1 - p$) and the weight of a path is the product of its edge labels. We want the total weight of all paths of length $\leq 3n$ starting from $(0, d)$ that touch the x -axis. Note that a path starting at $(0, d)$, containing i “up” edges, and ending on the x -axis at time t satisfies $t = 2i + d$.

$N \stackrel{\text{def}}{=} (\text{the number of paths from } (0, d) \text{ to } (2i + d, 0) \text{ that touch the } x\text{-axis only at time } 2i + d) = (\text{the number of paths from } (0, d) \text{ to } (2i + d - 1, 1) \text{ that do not touch the } x\text{-axis}) = (\text{the number of paths from } (0, d) \text{ to } (2i + d - 1, 1)) - (\text{the number of paths from } (0, d) \text{ to } (2i + d - 1, 1) \text{ that touch the } x\text{-axis})$. The paths from $(0, d)$ to $(2i + d - 1, 1)$ that touch the x -axis are in bijective correspondence with the paths from $(0, -d)$ to $(2i + d - 1, 1)$ via reflection of (the part of the path from $(0, d)$ to the first place it touches the x -axis) about the x -axis, as the following picture shows.



So

$$\begin{aligned}
N &= (\text{the number of paths from } (0, d) \text{ to } (2i + d - 1, 1)) \\
&\quad - (\text{the number of paths from } (0, -d) \text{ to } (2i + d - 1, 1)) \\
&= \binom{2i + d - 1}{i} - \binom{2i + d - 1}{i - 1} \\
&= \left(\frac{i + d}{2i + d} - \frac{i}{2i + d} \right) \binom{2i + d}{i} \\
&= \frac{d}{2i + d} \binom{2i + d}{i}.
\end{aligned}$$

So

$$p_{\text{succ}, d, \frac{1}{k}} = \sum_{i=0}^{\lfloor \frac{3n-d}{2} \rfloor} \frac{d}{2i + d} \binom{2i + d}{i} \left(\frac{1}{k} \right)^{i+d} \left(1 - \frac{1}{k} \right)^i.$$

We would like to lower bound the sum by the single term where $i = \alpha d, \alpha = \frac{1}{k-2}$. The only problem is that this might not be an integer. To get around that, we will assume d is a multiple of $k - 2$, which may require increasing d by as much as $k - 3$. Now $p_{\text{succ}, d, \frac{1}{k}}$ is an increasing function of d and increasing d by $k - 3$ only decreases $p_{\text{succ}, d, \frac{1}{k}}$ by at most a factor of $k^{-(k-3)}$ (which, in fact, is 1 if $k = 3$) since this is the probability that the first $k - 3$ steps all move towards 0. Assuming that d is a multiple of $k - 2$ and using theorem 2.3 we have

$$\begin{aligned}
p_{\text{succ}, d, \frac{1}{k}} &\geq \frac{1}{3n} \binom{(1 + 2\alpha)d}{\alpha d} \left(\frac{1}{k} \right)^{(1+\alpha)d} \left(1 - \frac{1}{k} \right)^{\alpha d} \\
&\geq \frac{1}{6n^{\frac{3}{2}}} 2^{(h(\frac{\alpha}{1+2\alpha})(1+2\alpha) - (1+\alpha) \lg k + \alpha \lg \frac{k-1}{k})d}.
\end{aligned}$$

The exponent divided by d is

$$\begin{aligned}
&h\left(\frac{1}{k}\right) \frac{k}{k-2} - \frac{k-1}{k-2} \lg k + \frac{1}{k-2} \lg \frac{k-1}{k} \\
&= \left(\frac{1}{k} \lg k + \frac{k-1}{k} \lg \frac{k}{k-1} \right) \frac{k}{k-2} - \frac{k-1}{k-2} \lg k + \frac{1}{k-2} \lg \frac{k-1}{k} \\
&= -\lg(k-1).
\end{aligned}$$

So $p_{\text{succ}, d, \frac{1}{k}} \geq (6n^{\frac{3}{2}})^{-1} (k-1)^{-d}$. If we drop the assumption that d is divisible by $k - 2$, then

$$p_{\text{succ}, d, \frac{1}{k}} \geq (6n^{\frac{3}{2}} k^{k-3})^{-1} (k-1)^{-d}.$$

□

Corollary 6.6. $\forall k \geq 3 \ p_{\text{succ}} \geq \Omega(n^{-\frac{3}{2}}) \left(\frac{1}{2} \left(1 + \frac{1}{k-1} \right) \right)^n$.

Proof. If d is binomial $(n, \frac{1}{2})$, then

$$\begin{aligned} p_{\text{succ}} &\geq E_d(p_{\text{succ}, d, \frac{1}{k}}) \\ &\geq \Omega(n^{-\frac{3}{2}}) \sum_{d=0}^n \binom{n}{d} (k-1)^{-d} 2^{-n} \\ &= \Omega(n^{-\frac{3}{2}}) \left(\frac{1}{2} \left(1 + \frac{1}{k-1} \right) \right)^n. \end{aligned}$$

□

Theorem 6.7. *The exponential complexity of LocalSearch is*

$$c_{\text{LocalSearch}} \leq 1 - \frac{1}{(\ln 2)k}.$$

Proof. From corollary 6.6, we can achieve a completeness error of $\leq e^{-1}$ with a run time of $\leq \text{poly}(n)(2(1 - \frac{1}{k}))^n$. From theorem 2.1 and (2.1), we thus have that $c_{\text{LocalSearch}} \in [1 - \frac{1}{(\ln 2)(k-1)}, 1 - \frac{1}{(\ln 2)k}]$. □

Note carefully that although the analysis fixed $b \in \text{sol}(F)$ and used it to define a random walk, p_{succ} is *not* a lower bound on the probability that $\text{LocalSearch_main}(F) = b$ since the random walk that the algorithm takes may stop before reaching state 0 if the algorithm hits a solution other than b . Here is another way to see this. If a random variable X takes values in some set S and $\forall b \in S \Pr(X = b) \geq p_{\text{succ}}$, then

$$1 = \Pr(X \in S) = \sum_{b \in S} \Pr(X = b) \geq |S| p_{\text{succ}},$$

which implies that $|S| \leq \frac{1}{p_{\text{succ}}}$. The k -CNF formula

$$F = \{\{x_1, \dots, x_{k-1}, x_i\} \mid i \in [k, n]\}$$

has $(2^{k-1} - 1)2^{n-(k-1)} + 1 \geq (1 - 2^{-(k-1)})2^n \geq \Omega(2^n)$ solutions. So $\text{LocalSearch_main}(F)$ cannot possibly find each $b \in S$ with probability $\geq p_{\text{succ}} \geq \omega(2^{-n})$.

6.4.1 Derandomization of LocalSearch

[DGH⁺02] derandomize LocalSearch by using covering codes and treating the local search decisions as deterministic k -way branches. A (n, r, s) *covering code* (of length n , radius r , size s) is a subset C of the Boolean cube $A = 2^{\text{var}(F)}$ (where $|\text{var}(F)| = n$) such that $\bigcup_{c \in C} B_r(c) = A$ and $|C| \leq s$. A *radius- r deterministic local search* starting at $a \in A$ is where we simulate LocalSearch_main on all possible sequences of r -steps starting at a – this finds all solutions in $B_r(a)$ that are closest to a , and possibly others. Instead of initially choosing $a \in A$ at random, for each $c \in C$, we perform a radius- r deterministic local search starting at c . The analysis of this algorithm is easy enough that we will repeat it here.

Theorem 6.8. Let $p \in [0, \frac{1}{2}]$, $n \geq 1$, $r = pn \in \mathbb{N}$.

1. There is no (n, r, s) covering code with $s < 2^{(1-h(p))n}$.
2. There is a (n, r, s) covering code with $s \leq \lceil 2n^{\frac{3}{2}} 2^{(1-h(p))n} \rceil$.

Proof. From theorem 2.3, the number of spheres of radius $r = pn$ required to cover $A = 2^n$ is $\geq \frac{2^n}{|B_r(0^n)|} \geq 2^{(1-h(p))n}$. The probabilistic method can be used to show that we can achieve this bound up to a polynomial factor: let C be a set of s points of A chosen uniformly at random, independently, and with replacement. Then

$$\begin{aligned}
& \Pr(C \text{ is not a covering code of radius } r) \\
&= \Pr(\exists a \in A \forall c \in C a \notin B_r(c)) \\
&\leq 2^n \left(1 - \frac{|B_r(0^n)|}{2^n}\right)^s \quad \text{from the union bound and independence} \\
&\leq 2^n \left(1 - \frac{1}{2\sqrt{n}} 2^{-(1-h(p))n}\right)^s \quad \text{from theorem 2.3} \\
&\leq \left(\frac{2}{e}\right)^n \quad \text{choosing } s = \lceil 2n^{\frac{3}{2}} 2^{(1-h(p))n} \rceil,
\end{aligned}$$

and so there is a (n, r, s) covering code. $\square$

The l th power C^l of a (n, r, s) covering code C is the set of all concatenations of l codewords of C , and C^l is a (ln, lr, s^l) covering code. For an $n' \leq o(n)$ to be determined later, the time required to exhaustively search for a $(n', pn', 2^{(1-h(p))n' + o(n')})$ covering code C' , guaranteed to exist by theorem 6.8, is crudely upper bounded by $2^{2^{n'+1}}$. By taking the $\lceil \frac{n}{n'} \rceil$ th power of C' (and trimming the last $n \bmod n'$ bits from each codeword), we get a $(n, pn + pn', 2^{(1-h(p))n + o(n)})$ covering code C .

The time it takes to perform a radius- r deterministic local search on a k -CNF with m clauses is $\leq \text{poly}(n, m)k^r$, where the polynomial does not depend on k , and so the whole algorithm takes time

$$\begin{aligned}
& \text{time to construct } C + |C| \cdot \text{time for a local search} \\
&\leq 2^{2^{n'+1}} + 2^{(1-h(p))n + o(n) + (pn + pn') \lg k}.
\end{aligned}$$

If $n' \leq \frac{1}{2} \lg n$, then the first term above is $2^{o(n)}$, and so the run time is

$$\begin{aligned}
&\leq 2^{(1-h(p) + p \lg k)n + o(n) + pn' \lg k} \\
&\leq 2^{(1 - \lg(1 + \frac{1}{k}))n + o(n)} \quad \text{setting } p = \frac{1}{k+1}.
\end{aligned}$$

(Note that the hidden constant in the little-Oh does not depend on k . We will use this fact in chapter 10.) So the exponential complexity of the algorithm is at most

$$c = 1 - \lg\left(1 + \frac{1}{k}\right). \quad (6.10)$$

The complete algorithm is in figure 6.3. Note that for all k , by theorem 2.1, $c \leq 1 - \frac{1}{k}$, which makes *deterministic* local search superior to *randomized* PPZ; and for large k , $c \approx 1 - \frac{1}{k}$, which is inferior to PPSZ. Note also that the space used is $\leq \text{poly}(n)$.

6.5 Combine PPSZ with LocalSearch, $k \in \{3, 4\}$

For $k \in \{3, 4\}$, [IT04] showed that PPSZ and LocalSearch could be combined to yield an algorithm superior to both. The key idea is that the analysis of PPSZ seems to get worse as the number of solutions increases while for LocalSearch it seems to improve, so the 2 algorithms cannot achieve their worst-case performances simultaneously.

Suppose $F \in k\text{-CNF}$, $A = 2^{\text{var}(F)}$, $S = \text{sol}(F) \neq \emptyset$, $n = |\text{var}(F)|$. PPSZ_main(F) uses a guess $a \in_u A$ as a source of randomness, so that when it needs to guess the value of variable i , it uses a_i . LocalSearch_main(F) starts at a random location $a \in A$. The Iwama-Tamaki algorithm simply uses the same a for both algorithms.

To analyze this, recall from the discussion of PPSZ that A can be partitioned into a set of subcubes $\mathcal{B}$ so that each $B \in \mathcal{B}$ contains exactly 1 solution $z_B \in S$, and that D_B denotes the defining variables for B , i.e. the set of variables whose values are all in agreement within B . Normalize this to $\Delta_B = \frac{|D_B|}{n}$. (6.9) actually applies to any algorithm, not just PPSZ_main, so

$$\Pr(\text{alg}(F) \in S) \geq \min_{B \in \mathcal{B}} \Pr(\text{alg}(F) \in S \mid a \in B), \quad (6.11)$$

where $\text{alg} \in \{\text{PPSZ_main}, \text{LocalSearch_main}\}$. [PPSZ05], corollary 14, shows that

$$\begin{aligned} & \frac{1}{n} \lg \Pr(\text{PPSZ_main}(F) \in S \mid a \in B) \\ & \geq -\left(1 - \frac{\mu_k}{k-1}\right) + \left(1 - \frac{\mu_k}{k-1} - \lg \frac{k-1}{k-2}\right) \Delta_B - o(1), \end{aligned} \quad (6.12)$$

where μ_k was defined in §6.3 (recall that $\mu_3 = 4 - 4 \ln 2$, $\mu_4 \approx 1.3354$).

Notice that if $a \in B$, then $d = \text{dist}(a, z_B)$ is binomial $((1 - \Delta_B)n, \frac{1}{2})$. Lemma 6.5 shows that $\Pr(\text{LocalSearch_main}(F) \in S \mid \text{dist}(a, z_B) = d) \geq (k-1)^{-d-o(n)}$. Using this to generalize corollary 6.6, we have that

$$\begin{aligned} & \frac{1}{n} \lg \Pr(\text{LocalSearch_main}(F) \in S \mid a \in B) \\ & \geq -(1 - \Delta_B) \lg \frac{2(k-1)}{k} - o(1). \end{aligned} \quad (6.13)$$

Ignoring the $o(1)$ terms, the minimum over all $\Delta_B \in [0, 1]$ of the maximum of (6.12), (6.13) occurs where

$$\Delta_B = \frac{\frac{\mu_k}{k-1} + \lg \frac{k-1}{k}}{\frac{\mu_k}{k-1} + \lg \frac{k-1}{k} + \lg \frac{k-1}{k-2}},$$

which is $\frac{3-2 \ln 2 - \lg 3}{4-2 \ln 2 - \lg 3} \approx .02794$ for $k = 3$, and $\frac{\frac{\mu_4}{3} + \lg \frac{3-2}{3}}{\frac{\mu_4}{3} + 2 \lg \frac{3-2}{3}} \approx .04896$ for $k = 4$. Inserting this into (6.13), we have that the following.

Theorem 6.9. *The exponential complexity of the Iwama-Tamaki algorithm is ¹*

$$c_{\text{Iwama-Tamaki}} \leq \frac{(1 + \lg \frac{k-1}{k}) \lg \frac{k-1}{k-2}}{\frac{\mu_k}{k-1} + \lg \frac{k-1}{k} + \lg \frac{k-1}{k-2}},$$

which is $\frac{2-\lg 3}{4-\lg 3-2\ln 2} \approx .4034$ for $k = 3$ and $\approx .5563$ for $k = 4$.

We can now see why combining PPSZ and LocalSearch only gives improved running times when $k \in \{3, 4\}$: when $k \geq 5$, (6.12) becomes an *increasing* function of Δ_B , and so the PPSZ analysis achieves its worst case when $\Delta_B = 0$, i.e. there is no additional advantage to running LocalSearch. This is the same reason why the PPSZ analysis gives the same exponential complexity for the uniquely satisfiable case and the general case for $k \geq 5$.

[IT04] claim that simply running PPSZ and LocalSearch independently and in parallel does not seem to work, but that they must use the same value of a on each iteration. This, however, is not true, as the following elementary analysis shows. Let $A_1, \dots, A_l$ be events on a sample space S , and $A'_1, \dots, A'_l$ be independent events with the same probabilities: $\forall i \in [l] \Pr(A'_i) = \Pr(A_i)$. Then

$$\begin{aligned} \Pr(A'_1 \cup \dots \cup A'_l) &\geq \frac{1}{l} \left(l \max_{i \in [l]} \Pr(A_i) \right) \\ &\geq \frac{1}{l} \Pr(A_1 \cup \dots \cup A_l) \quad \text{from the union bound.} \end{aligned} \tag{6.14}$$

For us, $l = 2$, A_i is the event that the i th algorithm finds a solution, the RHS of (6.14) is the probability that some algorithm finds a solution given that they all use the same value of a , and the LHS is the probability that some algorithm succeeds given that they use independent values of a . So the probability that independent calls to PPSZ and LocalSearch succeed is at worst $\frac{1}{2}$ that of dependent calls. In all of our analyses, we ignore $\text{poly}(n)^{-1}$ factors, so independent calls, and therefore running the algorithms in parallel, actually yields the same exponential complexity.

6.6 Minor improvements

6.6.1 Deterministic local search, $k = 3$

[DGH⁺02] improve the exponential complexity of deterministic local search to $\approx .5666$ for the case that $k = 3$ by using unit clauses to slightly reduce the branching factor in LocalSearch_det_rec. The logic is fairly complex, so we do not repeat it here, but it makes one wonder whether deeper deductive logic could be used to get further speedups. E.g. consider how a unit clause $C = \{l\}$ is generally used: at some point we spot C and decide to set $l = 1$ because we know that regardless of how we proceed, at some point in the future, we know we must set $l = 1$; so why branch now, especially when setting $l = 1$ greedily now may open up

¹[Rol05b] points out that these constant are slightly better than those of [IT04] because they use an older version of (6.12).

Table 6.1: performance of k -SAT algorithms

algorithm	type	promise	k	exp complexity	reference
PPZ	ran	none	≥ 3	$1 - \frac{1}{k}$	[PPZ99]
		s solutions		$(1 - \frac{1}{k})(1 - \frac{\lg s}{n})$	[CIKP08]
	det	none		$1 - (\frac{1}{2} - \frac{1}{\Theta(\sqrt{k})})^{\frac{1}{k}}$	[PPZ99]
		fully isolated solution		$1 - \frac{1}{k}$	
PPSZ	ran	none	3	.5207	[PPSZ05]
			4	.5626	
			≥ 5	$1 - \frac{\mu_k}{k-1}$	
	det	unique solution	≥ 3	$1 - \frac{\mu_k}{k-1}$	[Rol05a]
LocalSearch	ran	none	≥ 3	$1 - \frac{1}{(\ln 2)^k}$	[Sch99]
	det			$1 - \lg(1 + \frac{1}{k})$	[DGH ⁺ 02]
			3	.5666	
			4	.6781	
IT	ran	none	3	.4034	[IT04]
				.4029	[Rol05b]
			4	.5563	[IT04]

opportunities to apply more branch-reducing logic? Could limited resolution increase this ability to 'look ahead' as it did with PPSZ?

6.6.2 Improved analysis for Iwama-Tamaki, $k = 3$

For $k = 3$, [Rol05b] showed that the bound (6.12) could be tweaked for small Δ_B and so reduce $c_{\text{Iwama-Tamaki}}$ from .4034 to .4029, which is currently the best worst-case upper bound for 3-SAT. The analysis is long and technical, but it suggests 2 ways of gaining future improvement: either tighten the analysis of 1 algorithm around the critical region for Δ_B , or find some new algorithm that outperforms those in your suite around the critical region and add it to the mix.

Of course the improvement in [Rol05b] is so small and comes at such a high cost of intellectual labor that one could also argue that it may be more fruitful to seek alternative algorithmic paradigms. An even more important reason to do this is to gain better insight into the structure of the problem, which likely has more applications than merely better run times. For example, although PPZ is not the fastest k -SAT algorithm, theorem 6.2 gives a powerful structural property, which was used by [PPZ99] to give a tight lower bound on the size of depth 3 circuits computing parity.

6.7 Summary and open problems

In table 6.1 we summarize most of the performance results from the chapter. Any blank entry inherits the value from the entry above.

Is it possible to use unit clauses to speed up randomized local search? It is possible to use resolution, together with PPSZ-like logic, to speed up randomized or deterministic local

search? Does the exponential complexity of PPSZ for the $k \in \{3, 4\}$ case actually increase with increasing numbers of solutions, or is this just a weakness of the analysis? (This author has spent considerable time wrangling with this question. It would have a big payoff since, if false, it would yield the best 3-SAT algorithm by far.) Are there algorithms that outperform the Iwama-Tamaki algorithm around the critical region for Δ_B ?

Notice that each algorithm discussed that works for all k has exponential complexity converging to 1 as $k \rightarrow \infty$, and in fact they all do so at the same asymptotic rate. We do not know of any randomized algorithm that solves k -SAT in time $O(2^{cn})$ for some $c < 1$ *independent* of k . To formalize this a bit, define

$$s_k = c_{k\text{-SAT}} \qquad s_\infty = \lim_{k \rightarrow \infty} s_k. \qquad (6.15)$$

In analogy to the Exponential Time Hypothesis (ETH, namely $s_3 > 0$), [IP01, Sch05] conjecture the following, which we will call the *Strong Exponential Time Hypothesis*.

Conjecture 6.10 (SETH). $s_\infty = 1$.

Acknowledgements

Theorem 6.1 is joint work with Russell Impagliazzo, Ramamohan Paturi, and Valentine Kabanets and appeared in [CIKP08].

```

LocalSearch_det( $F \in k\text{-CNF}$ )
  if  $\emptyset \in F$ , return “not satisfiable”
   $(p, n, r) \leftarrow (\frac{1}{k+1}, |\text{var}(F)|, \lceil pn \rceil)$  // wlog assume  $\text{var}(F) = n$ 
   $(n', r', s', l) \leftarrow (\lfloor \frac{\frac{1}{2} \lg n}{k+1} \rfloor (k+1), pn', \lceil 2n'^{\frac{3}{2}} 2^{(1-h(p))n'} \rceil, \lceil \frac{n}{n'} \rceil)$ 
  for each  $C' \subseteq 2^{n'}$ 
    if is_covering_code( $C', n', r', s'$ ), break
  let  $C' = \{c_1, \dots, c_s\}$  where  $s \leq s'$ 
  for each  $(i_1, \dots, i_l) \in [s]^l$ 
     $a \leftarrow$  first  $n$  bits of  $c_{i_1} \cdots c_{i_l}$ 
     $b \leftarrow \text{LocalSearch\_det\_rec}(F, r, a)$ 
    if  $b \neq \perp$ , return  $b$ 
  return “not satisfiable”

is_covering_code( $C, n, r, s$ )
  if  $|C| > s$ , return 0
  for each  $a \in 2^n$ 
     $d \leftarrow \infty$ 
    for each  $b \in C$ 
       $d \leftarrow \min\{d, \text{dist}(a, b)\}$ 
    //  $d = \text{dist}(a, C)$ 
    if  $d > r$ , return 0
  return 1

LocalSearch_det_rec( $F, r, a$ )
  if  $F(a) = 1$ , return  $a$ 
  if  $r = 0$ , return  $\perp$ 
  let  $C \in \{C \in F \mid C(a) = 0\}$  be arbitrary // e.g. the first element
  for each  $i \in \text{var}(C)$ 
     $b \leftarrow \text{LocalSearch\_det\_rec}(F, r-1, a \oplus i)$ 
    if  $b \neq \perp$ , return  $b$ 
  return  $\perp$ 

```

Figure 6.3: deterministic local search

Chapter 7

Quantified k -SAT

In this chapter, we consider the complexity of evaluating a quantified k -CNF. For $k = 2$, this is in linear time [APT79], but for $k \geq 3$, this is PSPACE-complete. Note that the exponential complexity of evaluating arbitrary quantified circuits, which is at least as hard, is ≤ 1 : evaluation of a circuit can be carried out in time polynomial in its size, and backtracking need explore $\leq 2^n$ paths, each of length $\leq n$, where n is the number of variables of the input (and this only requires polynomial space as well). So we will declare that an algorithm is *nontrivial* iff its exponential complexity is < 1 .

As with CNF-SAT, we will consider various syntactic restrictions of quantified k -SAT to give ourselves some chance of finding nontrivial algorithms – merely bounding k does not seem to be enough. Our main restriction is to bound the number of alternations of quantifier type. Define Π_i k -SAT (Σ_i k -SAT) as the problem of evaluating a quantified k -CNF $Q_1x_1 \cdots Q_ix_i \phi$ where $\phi \in k$ -CNF, each x_j is a tuple of Boolean variables, and each $Q_j \in \{\forall, \exists\}$ is $\forall$ iff j is odd (even). Π_i k -SAT (Σ_i k -SAT), for $k \geq 3$, is a canonical complete problem for the i th level of the polynomial hierarchy, i.e. for Π_i^P (Σ_i^P).

We will show that $s_\infty \leq c_{\Pi_2 \text{ 3-SAT}}$. Thus the Strong Exponential Time Hypothesis (SETH, conjecture 6.10) would imply that for Π_2 3-SAT, virtually no improvement over the 2^n exhaustive search algorithm is possible. Since Π_i k -SAT is at least as hard when $i \geq 2, k \geq 3$, the complexity of problems in the polynomial hierarchy would seem to top off rather early. Of course, SETH is a very strong assumption, so this evidence should be considered weak. Conversely, a single nontrivial algorithm for Π_2 3-SAT would provide a k -SAT solver better than all current algorithms for sufficiently large k .

We also show several syntactic restrictions of Π_2 3-SAT that have nontrivial algorithms, giving strong evidence that the hardest cases of Π_2 3-SAT must have a mixture of clauses of 2 types: 1 universal literal and 2 existential literals, or only existential literals. Algorithmic design may benefit by concentrating on the hard cases, in much the same way that much of the progress

in k -SAT solvers has been driven by focusing on the hard cases – when there are a linear number of clauses, only one solution, expanding variable-dependence graph, etc.

Lastly, we relate the exponential complexities of Π_i k -SAT (Σ_i k -SAT) for various i, k when the clause density is not too high, although it may well be that these are all simply 1.

[Wil02] explored similar problems, namely exponential upper bounds for solving quantified constraint satisfaction problems (QCSP) with domain size d and cn constraints, each of width $\leq k$. When $d = 2$ and the formula is in k -CNF, we'll call this k -QCNF. [Wil02] solves QCSP for $d = 3$ in time $O(2^{.669cn})$, QCNF in $O(2^{.536cn})$ time and space, 3-QCNF in time $O(2^{.610cn})$, and the special case of 3-QCNF with $c = 1$ in time $O(2^{.482n})$. Each of these techniques outperforms exhaustive search only when c is very small, certainly no more than 2. This is probably not an accident: our results suggest that finding a substantially improved algorithm for moderate c , even for low quantifier depth, is unlikely.

The results here are currently unpublished joint work with the author, Russell Impagliazzo, and Ramamohan Paturi.

7.1 Π_2 3-SAT lower bound

Our proof uses a construction similar to [Che05], proposition 3.2, that allows us to reduce the clause width of a k -CNF using a small number of new quantified variables. For this, we need a kind of minimally unsatisfiable k -CNF with a large number of clauses. Our construction differs from [Che05] primarily in its efficiency, since we are more concerned with exact complexity rather than hardness for a class closed under polynomial time reductions, such as NP or coNP.

Say a pair (F, G) of formulas in CNF is *minimally unsatisfiable* iff $\forall F' \subseteq F ((F' \wedge G) \notin \text{SAT} \leftrightarrow F' = F)$. Say that $\phi \in \text{CNF}$ is *combinable* with (F, G) iff $|\phi| \leq |F|$ and $\text{var}(\phi) \cap \text{var}(F \wedge G) = \emptyset$. In this case, letting $f : \phi \rightarrow F$ be an arbitrary injection, we define

$$\text{combine}(\phi, F, G) = G \cup \{\{\bar{l}\} \cup f(C) \mid l \in C \in \phi\} \cup (F - \text{ran } f),$$

i.e., starting with G , for each clause C of ϕ and literal l of C , we add a clause meaning “ l implies the clause of F corresponding to C ”; and any clauses of F that are left over get added as-is, without adjoining a literal.

Lemma 7.1. *Let ϕ be combinable with minimally unsatisfiable (F, G) and $\phi' = \text{combine}(\phi, F, G)$. Then $\forall a \in 2^{\text{var}(\phi)} (\phi(a) = 1 \leftrightarrow \phi'|a \notin \text{SAT})$.*

Proof. This is simply because in ϕ' , after assigning to $\text{var}(\phi)$, each clause $f(C)$ survives iff some literal $l \in C$ is assigned true. Note that this does not depend on the specific injection $f : \phi \rightarrow F$ chosen. $\square$

Lemma 7.2. *$\forall k \geq 3$, given $m \in \mathbb{N}$, one can construct in time $\text{poly}(m)$, $F \in (k-1)$ -CNF and $G \in k$ -CNF such that*

- (F, G) is minimally unsatisfiable
- $|F| \geq m$
- $|\text{var}(F \wedge G)| \leq \frac{(k-1)^2}{k-2} \lceil m^{\frac{1}{k-1}} \rceil$.

Proof. Let $l = \lceil m^{\frac{1}{k-1}} \rceil$. Consider a $(k-1) \times l$ matrix A of Boolean variables $x_{i,j}$ and the contradictory statements

1. “every row of A contains a 1”,
2. “every choice of one entry from each row contains a 0”.

We can express 1. by using $\leq \frac{k-1}{k-2}l$ new variables $y_{i,j}$ and k -clauses G as follows. For each i , partition $x_{i,1}, \dots, x_{i,l}$ into $b = \lceil \frac{l}{k-2} \rceil$ blocks of size $\leq k-2$. Add to G k -clauses expressing that $y_{i,1}$ is the OR of the variables of the first block; $y_{i,j}$ is the OR of $y_{i,j-1}$ and the variables of the j th block, for $j \in [2, b-1]$; and that the OR of $y_{i,b-1}$ and the variables of the b th block is true. We can express 2. by using $l^{k-1} \geq m$ $(k-1)$ -clauses F , which are disjunctions of the negations of the variables in the matrix where each disjunction contains exactly one variable from each row. F and G have the desired properties. $\square$

Corollary 7.3. $\forall k, k' \geq 3$, given $\phi \in k$ -CNF with n variables x and m clauses, one can construct in time $\text{poly}(n)$, $\phi' \in k'$ -CNF with n variables x and $\frac{(k'-1)^2}{k'-2} \lceil m^{\frac{1}{k'-1}} \rceil$ variables y such that $\forall a \in 2^{\text{var}(\phi)}$ ($\phi(a) = 1 \leftrightarrow \phi'|a \notin \text{SAT}$). In particular, $\phi \in \text{SAT}$ iff $\psi = \forall x \exists y \phi'$ is false.

Proof. Use lemma 7.2 to construct minimally unsatisfiable (F, G) with $F \in (k'-1)$ -CNF, $G \in k'$ -CNF, $|F| = m$, $|\text{var}(F \wedge G)| \leq \frac{(k'-1)^2}{k'-2} \lceil m^{\frac{1}{k'-1}} \rceil$, and $\text{var}(\phi) \cap \text{var}(F \wedge G) = \emptyset$. Let $\phi' = \text{combine}(\phi, F, G)$ and apply lemma 7.1. $\square$

Theorem 7.4. $s_\infty \leq c_{\Pi_2 \text{ 3-SAT}}$.

Proof. Fix k and $\epsilon > 0$. We will show that $s_k \leq c_{\Pi_2 \text{ 3-SAT}} + 4\epsilon$. Let ϕ be a k -CNF with n variables. Use theorem 5.15 to sparsify ϕ into at most $2^{\epsilon n}$ many subformulas ϕ_i so that each has $\leq m = \lfloor \frac{\epsilon n}{4} \rfloor^2$ clauses for all sufficiently large n . By corollary 7.3, with $k' = 3$, for each ϕ_i , we can construct in time $\text{poly}(n)$ a $\psi_i \in \Pi_2 \text{ 3-CNF}$ with $(1+\epsilon)n$ variables such that $\phi_i \in \text{SAT}$ iff ψ_i is false. So $\phi \in \text{SAT}$ iff ψ_i is false for some i . Evaluating each ψ_i with some $\Pi_2 \text{ 3-SAT}$ solver with exponential complexity $\leq c_{\Pi_2 \text{ 3-SAT}} + \epsilon$, we see that

$$s_k \leq (c_{\Pi_2 \text{ 3-SAT}} + \epsilon)(1 + \epsilon) + \epsilon \leq c_{\Pi_2 \text{ 3-SAT}} + 4\epsilon.$$

$\square$

The previous result puts us in an interesting situation. [IP01] showed that ETH, namely that $s_3 > 0$, implies s_k increases infinitely often as a function of k . Consider the following much weaker analogue.

Conjecture 7.5. $\text{ETH} \Rightarrow c_{\Pi_i \text{ } k\text{-SAT}}$ increases for some i or $k \geq 3$.

Conjecture 7.5, together with theorem 7.4, would imply that $s_\infty < 1$.

7.2 Algorithms for 2 special cases

In this section, we show that Quantified CNF-SAT has a poly time algorithm if each clause has ≤ 1 existential literal and that Π_2 3-SAT has a nontrivial algorithm if each clause has ≥ 1 universal literal. The purpose of such theorems is to find where the hard cases of Π_i k -SAT lie.

We begin with a simple classification theorem. Recall from chapter 4 that 2 clauses A, B *disagree* on variable x iff one of A, B contains x and the other contains $\bar{x}$. A, B are *resolvable* iff they differ in exactly 1 variable x , and the *resolvent* is $\text{resolve}(A, B) = A \cup B - \{x, \bar{x}\}$, which is logically implied by $A \wedge B$. Given a CNF ϕ , the result of one round of resolution is $\phi \cup \{\text{resolve}(A, B) \mid A, B \in \phi \text{ are resolvable}\}$. Repeatedly applying resolution eventually yields a CNF ϕ' that is closed under resolution.

A *resolution tree* for ϕ is a binary tree where each leaf is labeled with a clause of ϕ , and each interior node is labeled with the resolvent of the labels of its children. We can view such a tree as a proof of the label of the root, and the resolution proof system is sound and complete in the sense that a clause C is logically implied by ϕ iff some $C' \subseteq C$ has a resolution tree proof iff some $C' \subseteq C$ is in the closure of ϕ under resolution. E.g. $\phi \notin \text{SAT}$ iff $\emptyset$ is derivable via resolution. We can extend this logic to Π_2 CNFs.

Lemma 7.6. Let $\phi \in \text{CNF}$. $(\neg \forall x \exists y \phi)$ iff some purely universal clause can be derived from ϕ via resolution.

Proof. $(\Leftarrow)$ Obvious.

$(\Rightarrow)$ Suppose $\exists x_0 \forall y \neg \phi$. Let $\psi = \phi|(x = x_0)$, so that $\forall y \neg \psi$. Then $\emptyset$ is derivable from ψ via resolution. Add the universal literals back into the nodes of a resolution tree for $\emptyset$. By induction, working from the leaves of the tree back to the root, one can see that each such literal must be false at x_0 . So each pair that was resolvable in the original tree still is resolvable. $\square$

Theorem 7.7. Let $\psi = \forall x_1 \exists x_2 \cdots \forall x_n \phi$ be a quantified Boolean formula where each clause of $\phi \in \text{CNF}$ has ≤ 1 existential literal. Then ψ can be evaluated in poly time.

Proof. A variable x_i is quantified *before* x_j iff $i < j$. Let C be a clause containing an existential literal z and let x (y) be the set of universal literals in C that are quantified before (after) z is quantified. If each literal in x is assigned false, then player 'exists' must assign z to true, otherwise player 'forall' could assign each literal in y to false, making C false. If some literal in x is assigned true, then C is eliminated. Either way, y can be removed from C without changing

the truth value of ψ . So wlog we assume the universal literals of each clause are quantified *before* the existential literal, if there is one.

Perform 1 round of resolution on ϕ and call the result ϕ' . If there is some pure universal clause in ϕ' , then ψ is false, so reject. Else we claim ψ is true. To see this, consider the following strategy for player 'exists': For each existential literal z that must be assigned this turn, if there is some clause $C \in \phi$ containing z and only false universal literals, then assign z to 1. At the end, assign leftover existential variables arbitrarily.

Obviously this satisfies ϕ provided it is a consistent assignment. If we try to assign existential literals $z, \bar{z}$ both to 1, then there must be a clause $C \in \phi$ containing only z and false universal literals and a clause $D \in \phi$ containing only $\bar{z}$ and false universal literals. But then C, D are resolvable and ϕ' would contain a pure universal clause. So the assignment is consistent. $\square$

Theorem 7.8. *Let $\psi = \forall x \exists y \phi$, where $\phi \in 3\text{-CNF}$. If each clause of ϕ has ≥ 1 universal literal, then ψ can be evaluated in time 2^{cn} for some $c < 1$.*

Proof. Let $G = (V, E)$ where

$$\begin{aligned} V &= \{\text{existential literals of } \phi\} \\ E &= \{(a, b) \text{ labeled with } c \mid (\bar{a} \vee b \vee \bar{c}) \text{ is a clause of } \phi \\ &\quad \text{and } a, b \text{ are existential and } c \text{ is universal}\}. \end{aligned}$$

A *consistent path* in G is a path such that no 2 edge labels disagree, a *consistent cycle* is a consistent path that is a cycle. Then ψ is false iff there is an existential variable z s.t. there is a consistent cycle in G containing $z, \bar{z}$.

Notice that a simple cycle containing $z, \bar{z}$ cannot use more edge labels than $4\epsilon n$, where ϵn is the number of existential variables. So we can test whether there is a consistent cycle in G containing $z, \bar{z}$ in time $\text{poly}(|\psi|)$ times $\leq \binom{2n}{4\epsilon n} \leq 2^{2h(2\epsilon)n}$.

If ϵ is large, we can exhaustively search over all settings of the universal variables and then use a 3-SAT algorithm on the rest. If our 3-SAT algorithm runs in time $2^{s_3 n}$, then the combined algorithm runs in time at most $\min\{2^{(1-\epsilon)n + \epsilon s_3 n}, 2^{2h(2\epsilon)n}\}$, which is maximized at about $\epsilon = .05$ for $s_3 = .403$ and yields a run time of at most $2^{.97n}$. If s_3 is smaller, this value would be even smaller. Also, there is already an algorithm for 3-SAT with $s_3 \leq .403$ [Rol05b]. $\square$

It is unlikely that we can find a poly time algorithm for the variant where each clause has at least one universal literal because of the following.

Theorem 7.9. *The language of true Π_2 3-CNF formulas where each clause has ≥ 1 universal literal is coNP-complete.*

Proof. The language is in coNP because a witness for falsehood is a consistent cycle containing some existential variable z and its negation $\bar{z}$. To show coNP-hardness, we reduce from 3-UNSAT.

Let $\phi \in 3\text{-CNF}$ have n variables x and m clauses. Assume wlog that m is even, otherwise, just repeat some clause twice. Let $n' = \frac{m}{2}$, $y = (y_1, \dots, y_{n'})$ be new variables, and consider the following contradictory 2-CNF:

$$F = \{(y_i \rightarrow y_{i+1}), (y_{i+1} \rightarrow y_i) \mid i \in [n' - 1]\} \cup \{(y_{n'} \rightarrow \bar{y}_1), (\bar{y}_1 \rightarrow y_{n'})\}.$$

Then $(F, \emptyset)$ is minimally unsatisfiable and $|F| = m$. By lemma 7.1, $\psi = \forall x \exists y \text{ combine}(\phi, F, \emptyset)$ is a Π_2 3-CNF with ≥ 1 universal variable in each clause and such that $\phi \in \text{SAT}$ iff ψ is false. $\square$

The coNP-hardness of theorem 7.9 also follows from [Che05], proposition 3.2, if one observes that the number of universal variables in each constraint constructed there is 1.

7.3 Reduction to a canonical form

Theorems 7.4 and 7.8 together suggest that pure existential clauses (those with only existentially quantified literals) and clauses with a mixture of universal and existential literals are both essential to make an instance of Π_2 3-SAT hard. Below we present further evidence of this.

Let Π_2 3-SAT' be the special case of Π_2 3-SAT where each clause is one of just two types:

- (1) one universal literal and two existential literals
- (2) pure existential.

Theorem 7.10. $c_{\Pi_2 \text{ 3-SAT}'} < 1 \Rightarrow c_{\Pi_2 \text{ 3-SAT}} < 1$.

Proof. Let A be an algorithm with run time $\leq O(2^{cn})$ with $c < 1$ for Π_2 3-SAT'. Given a Π_2 3-CNF, do this: while there is a clause other than types (1) or (2), i.e., a clause C with at least one universal variable and at most one existential variable, reject if there is no existential variable in C , but otherwise branch on the universal variables. One of the branches will force the existential variable. Each formula at a leaf has only clauses of type (1) or (2), so apply algorithm A . This solves the general case in time at most $O(2^{dn})$ where $d = \max\{c, \frac{\lg 7}{3}\} < 1$. $\square$

We may also assume, wlog, that in a Π_2 k -CNF the number of existential variables is $o(n)$, since otherwise we could branch on every possible setting of the universal variables and then invoke a k -SAT solver to obtain a nontrivial algorithm.

7.4 Parameter trade-off at higher levels of the hierarchy

In the next theorem, we show that when confronted with a Π_i k -SAT instance, one may be able to reduce k to $k' < k$ if one is willing to increase i by 1, and the input clause density

is not too high. If m is a function, let $\Sigma_i k\text{-SAT}_m$ ($\Pi_i k\text{-SAT}_m$) be $\Sigma_i k\text{-SAT}$ ($\Pi_i k\text{-SAT}$) but where the number of clauses is promised to be at most $m(n)$.

Theorem 7.11. *Let $k, k' \geq 3, m \in o(n^{k'-1})$. $\forall$ odd $i \geq 1$,*

$$c_{\Sigma_i k\text{-SAT}_m} \leq c_{\Pi_{i+1} k'\text{-SAT}}$$

$$c_{\Pi_i k\text{-SAT}_m} \leq c_{\Sigma_i k'\text{-SAT}}$$

(which is uninteresting for $i = 1$ since $\Pi_1 k\text{-SAT}$ is in P). $\forall$ even $i \geq 2$,

$$c_{\Sigma_i k\text{-SAT}_m} \leq c_{\Pi_i k'\text{-SAT}}$$

$$c_{\Pi_i k\text{-SAT}_m} \leq c_{\Sigma_{i+1} k'\text{-SAT}}.$$

Proof. Consider a quantified formula $Q_1 x_1 \cdots Q_i x_i \phi$ where each Q_j is a quantifier, each x_j is a tuple of Boolean variables, and ϕ is a k -CNF with n variables x and $\leq m(n)$ clauses. Using corollary 7.3, $\forall \epsilon > 0$ and for sufficiently large n , we can construct in poly time a k' -CNF ϕ' such that

- ϕ' has $(1 + \epsilon)n$ vars: x and ϵn new ones y
- for each assignment a to x , $\phi(a) = 1$ iff $\phi'|a \notin \text{SAT}$.

So

$$\begin{aligned} & Q_1 x_1 \cdots Q_i x_i \phi \\ \Leftrightarrow & Q_1 x_1 \cdots Q_i x_i \neg \exists y \phi' \\ \Leftrightarrow & \neg Q'_1 x_1 \cdots Q'_i \exists y \phi' \end{aligned}$$

where Q'_j is existential iff Q_j is universal. □

It would be nice to eliminate the requirement that ϕ have at most $o(n^{k'-1})$ clauses. One might try to do this by first sparsifying ϕ , but the possibly complex quantification of the variables prevents this.

Acknowledgements

Material here is unpublished joint work with Russell Impagliazzo and Ramamohan Paturi.

Chapter 8

Circuit-SAT upper bound

In this chapter, we consider the exponential complexity of the satisfiability problem for circuits of limited size and depth, which, as in the last chapter, seems significantly harder than k -SAT. $\forall d, c > 0$, we give a randomized algorithm solving the satisfiability problem for depth d circuits with n variables and $\leq cn$ gates with exponential complexity $1 - \alpha$ where $\alpha \geq 1/O(c^{2^{d-2}-1} \lg^{3 \cdot 2^{d-2}-2} c)$, and the constant in the big-Oh depends only on d . The algorithm can be adjusted for use with Grover's algorithm to achieve an exponential complexity of $\frac{1-\alpha}{2}$ on a quantum computer.

For $d = 2$, our algorithm becomes deterministic and matches the current best bound ([CIP06] and here in §10.2), since our algorithm and analysis are generalizations of the ones there. However, a randomization procedure similar to that used in theorem 5.16 also yields the best quantum algorithm for this case, with exponential complexity $\frac{1-1/O(\lg c)}{2}$. For $d = 3$, this gives $\alpha \geq 1/O(c \lg^4 c)$, which, as far as the author knows, is the first nontrivial algorithm shown for this problem, or any other AC^0 type problem with $d > 2$. (There is some work that improves the performance of circuit-SAT algorithms in terms of the circuit size parameter, but these algorithms are no better than exhaustive search once the circuit size gets larger than around $4n$ or so.)

The results here are joint work with the author, Russell Impagliazzo, and Ramamohan Paturi, to appear in IWPEC 2009.

8.1 Motivation

There are a few motivations to consider linear-size, constant depth circuits. One is the question of ideal block cipher design. Block ciphers are carefully constructed to maximize efficiency for a given level of security. Particularly, since we want ciphers to be usable by low-power devices, and to be implemented at the network level, it is often very important to have efficient hardware implementations that make maximum use of parallelism. A typical cipher

computes for a small number of “rounds”, where in each round, very simple operations are performed (e.g., substitutions from a look-up table called an S box, permutations of the bit positions, or bit-wise $\oplus$ operations). These operations are almost always AC^0 type or even simpler. It is also considered vital to have key sizes that are as small as possible, and an algorithm that breaks the cryptosystem in significantly less time than exhaustive search over keys is considered worrisome. So this raises the question: Can we have an ideal block cipher family (one per key size), i.e. so that the number of rounds remains constant, each round being implementable in constant depth with a linear number of gates, and security is almost that of exhaustive search over keys? The results here rule out such ideal block ciphers, and so give a partial explanation for why the number of rounds and/or key size needs to increase in new generations of block ciphers. (Block ciphers require average-case security, not worst-case, but worst-case algorithms obviously also rule out average-case security. Our values of α are vanishingly small for the sizes and depths of real cryptosystems, so our results cannot be used for cryptanalysis of existing block ciphers.)

Another motivation is that linear-size, constant depth circuits are perhaps the most general class of circuits for which we can expect to show nontrivial upper bounds. Since one can embed k -CNFs for any k into circuits of any super-linear size (in particular, super-constant density CNF; see[CIP06], also to be discussed later) nontrivial upper bounds for the satisfiability problem for nonlinear-size circuits would imply $s_\infty < 1$, contradicting SETH.

Yet another motivation is that improved algorithms for SAT for a circuit model $\mathcal{C}$ may reveal structural properties of the solution space of circuits in $\mathcal{C}$. These structural properties may in turn be helpful in proving stronger lower bounds on the size of circuits which are disjunctions of circuits in $\mathcal{C}$. In fact, [PSZ00, PPZ99, PPSZ05, IPZ01] exploit this connection to provide the best known lower bounds, of the form $2^{\Delta(k)n/k}$ where $\Delta(k) > 1$, for depth-3, unbounded fan-in circuits with bottom fan-in $\leq k$. This connection between the hardness of the satisfiability problem for a circuit model and lower bounds of a related circuit model is not surprising since, as a more general circuit can compute more complicated functions, it may be more difficult to invert, i.e. check the satisfiability of, these functions.

8.2 The algorithm

We only describe a version of the algorithm that takes poly time but has exponentially small success probability. To convert this into a backtracking algorithm, simply try both branches deterministically.

The analysis for the backtracking version of the algorithm is somewhat more complicated, mostly due to a complex interaction between passed parameters and actual guarantees on the input. Also, the subroutine `find_restriction` would still use randomness, so the extra effort would not even compensate us by yielding a fully deterministic algorithm, although the author

strongly suspects that this problem could be overcome. Nonetheless, for completeness, we include the backtracking version (without analysis, which would be essentially the same, just more verbose) at the end of the chapter in figure 8.2.

8.2.1 Definitions

The *inputs* (*outputs*) of a dag are those nodes with indegree 0 (outdegree 0). A *circuit* F is a dag where each input is labeled with a literal, each non-input (called a *gate*) is labeled AND or OR, and there is exactly 1 output. A *subgate* of size k of a gate g is a gate h (not necessarily in F) with the same label as g and with k of g 's inputs. The *depth* d of F is the number of edges in a longest path in F . The i th *level* of F is the set of gates a distance of i from the output, e.g. the output is at level 0 and the bottom gates are at level $d - 1$.

8.2.2 High level description

Let F be a circuit of depth d , $\text{var}(F) \subseteq V$ where $|V| = n$, and the size of F in gates is $|F| \leq cn$. We use V to keep track of a particular superset of the remaining variables $\text{var}(F)$. In particular, each time we assign a group of k variables, we will remove them from V . Subsequently simplifying the circuit may result in the removal of other variables from F so that $\text{var}(F) \subseteq V$ may become a proper containment. Because of these simplifying steps, $|V|$ is a measure of the algorithm's progress that will be easier to keep track of than $|\text{var}(F)|$.

To test the satisfiability of F , we first reduce the fan-in of each bottom gate to some k by repeatedly selecting a k -subset h of the edges coming into any bottom gate g with fan-in $> k$ and setting h to either true or false. One of these settings will eliminate k variables from V and the other will eliminate a gate. The one that eliminates k variables we choose with probability q and the other with probability $1 - q$.

We continue guessing until either we've halved the original number of variables, in which case we simply guess an assignment to the remaining variables, or each bottom gate has fan-in $\leq k$, in which case we choose a random restriction to all but pn variables of V for some p . This may leave some bottom level gates with > 1 free variable. We clean up these gates by randomly setting all the variables in them. By choosing k, p appropriately, with probability $\geq \frac{1}{2}$ there will still be $\geq \frac{1}{2}pn$ free variables in V , but the bottom level gates will each have ≤ 1 free variable. So we can collapse the circuit to depth $d - 1$ and recurse.

This procedure takes polynomial time and has exponentially small probability s of finding a satisfying assignment given that there is one. By iterating s^{-1} times, we increase the probability of success to a constant.

8.2.3 Detailed description

We describe our algorithm in several subroutines:

- $A_{d,c}(F, V)$ seeks a solution of F when F has depth d , $\text{var}(F) \subseteq V$, $|V| = n$, and $|F| \leq cn$. Although initially $\frac{|F|}{n} \leq c$, the algorithm may set variables, increasing the ratio. If it ever exceeds $2c$, $A_{d,c}$ will simply guess an assignment to the remaining variables since this means at least half of the variables must have been set.
- $A_{d,c,k}(F, V)$ seeks a solution of F when F has depth d , $\text{var}(F) \subseteq V$, $|V| = n$, $|F| \leq cn$, and F has bottom fan-in $\leq k$.
- $\text{find_restriction}(F, V, p)$ finds, with probability $\geq \frac{1}{2}$, a set of variables $W \subseteq V$ whose complement has size in the interval $[\frac{1}{2}pn, pn]$ and such that if the variables of W are assigned, then each bottom level gate has ≤ 1 free variable.
- PPZ_main is 1 iteration of the k -SAT solver (which is the same as a depth 2 circuit solver) from figure 6.1 – i.e., PPZ_main takes poly time and has success probability $\geq 2^{-(1-\frac{1}{k})n}$. Note that this algorithm solves depth 1 circuits in poly time and with success probability 1.

Our algorithm description is not the most succinct, compromises were made to simplify the proof. For example, it is easy to construct an equivalent algorithm containing only 1 subroutine by eliminating tail recursion, but the analysis would be obtuse. Below, the choices of k, p, q, c' are unspecified, and are left for the analysis section.

If h is an AND of literals, then $F|(h = 1)$ sets those literals to true and simplifies the circuit by removing true children of AND gates, false children of OR gates, replacing empty AND gates by true, replacing empty OR gates by false; unless h contains contradictory literals, in which case $F|(h = 1)$ is simply false. If h is an OR of literals, $F|(h = 1)$ removes any gate of which h is a subgate and then performs a similar simplification. Also if h is an AND (OR) of literals, then $F|(h = 0)$ can be treated as $F|(h' = 1)$ where h' is the OR (AND) of the negations of those literals. The purpose of all these definitions is so that below, in $A_{d,c}$, the line $F \leftarrow F|(h = b \text{ XOR } b')$ sets h in the way that eliminates k variables with probability q and the other way with probability $1 - q$. See figure 8.1.

8.3 Run time analysis

Suppose $A_{d,c}, A_{d,c,k}$ succeed with probability $\geq 2^{-(1-a_{d,c})n}, 2^{-(1-a_{d,c,k})n}$, respectively, given that find_restriction succeeds on each call to it – we will eliminate this assumption later. We assume $c \geq 2$ and $\forall d \geq 2, k \geq 4$ $a_{d,c}, a_{d,c,k} \leq \frac{1}{4}$.

Lemma 8.1. $\forall d, c \geq 2$ if $k \geq 4 \lg \frac{4c}{a_{d,2c,k}}$, then $a_{d,c} \geq \frac{1}{2}a_{d,2c,k}$.

Proof. Each iteration of the while loop of $A_{d,c}(F, V)$ eliminates (1) a gate or (2) k variables. (1) occurs $\leq cn$ times and (2) occurs $r \leq \frac{n}{k}$ times. Let a be a solution to F . Exactly one sequence of

random choices, say with r choices of type (2), can lead $A_{d,c}(F, V)$ to find a . So the probability that $A_{d,c}(F, V)$ finds a solution given that each call to `find_restriction` succeeds, is

$$\geq q^r (1 - q)^{cn} \min\{2^{-(1-a_{d,2c,k})(n-kr)}, 2^{-\frac{n}{2}}\}.$$

(Note carefully that we are *not* asserting that with at least this probability a is found. This is because `PPZ_main` may return another solution.)

To lower bound $q^r (1 - q)^{cn} 2^{-(1-a_{d,2c,k})(n-kr)}$, take the logarithm, divide by n , and set $r' = \frac{r}{n} \in [0, \frac{1}{k}]$ to get

$$\begin{aligned} & r' \lg q + c \lg(1 - q) - (1 - a_{d,2c,k}) + (1 - a_{d,2c,k})kr' \\ &= - (1 - a_{d,2c,k}) + c \lg(1 - q) + r'(\lg q + (1 - a_{d,2c,k})k) \\ &\geq - (1 - a_{d,2c,k}) - 2cq + r'(\lg q + (1 - a_{d,2c,k})k) \quad \text{by theorem 2.1 with } n = 4 \\ &\geq - (1 - a_{d,2c,k}) - 2cq + r' \left(\lg q + \frac{1}{2}k \right) \quad \text{since } a_{d,2c,k} \leq \frac{1}{2}, \end{aligned}$$

which is $\geq -(1 - \frac{1}{2}a_{d,2c,k})$ if we set $q = \frac{a_{d,2c,k}}{4c}$, $k \geq -2 \lg q$.

To lower bound $q^r (1 - q)^{cn} 2^{-\frac{n}{2}}$, note that if we choose $k \geq -4 \lg q$, then

$$\begin{aligned} & r' \lg q + c \lg(1 - q) - \frac{1}{2} \\ &\geq \frac{1}{k} \lg q - 2cq - \frac{1}{2} \\ &\geq -\frac{1}{4} - \frac{1}{2}a_{d,2c,k} - \frac{1}{2} \\ &\geq - \left(1 - \frac{1}{2}a_{d,2c,k} \right) \quad \text{since } a_{d,2c,k} \leq \frac{1}{4}. \end{aligned}$$

□

Lemma 8.2. *If $|V| = n$, $|F| \leq cn$, F has bottom fan-in $\leq k$, and we choose $p = \frac{1}{2ck^3}$, then the probability that `find_restriction`(F, V, p) dies is $\leq \frac{1}{2}$.*

Proof. Let $g \in B$ and $X = |\text{var}(g) - U|$. g has a fan-in $k' \leq k$ and so X is hypergeometric with parameters n, k', pn . We claim that $\Pr(g \in G) = \Pr(X \geq 2) \leq \binom{k'}{2} p^2$. To see this, note that the sample points where $X \geq 2$ can be partitioned according to the positions among the k' variables of g of the first 2 free variables (i.e., not in U), and the probability of any of these $\binom{k'}{2}$ events is $\leq p^2$. So

$$E(|U'|) \leq E(|G|)k \leq \binom{k}{2} p^2 |B|k \leq \frac{1}{2} k^3 p^2 cn \leq \frac{1}{4} pn.$$

By Markov's inequality,

$$\Pr(|U'| > \frac{1}{2}pn) \leq \frac{E(|U'|)}{\frac{1}{2}pn} \leq \frac{1}{2}.$$

So the probability that `find_restriction`(F, V, p) dies is $\leq \frac{1}{2}$. □

Lemma 8.3. *If we set $p = \frac{1}{2ck^3}$, $c' = 4c^2k^3$, then $a_{d,c,k} \geq \frac{1}{4ck^3} a_{d-1,4c^2k^3}$.*

Proof. $A_{d,c,k}(F, V)$ leaves $f \in [\frac{1}{2}pn, pn]$ variables of V free and sets the rest. So $\frac{|F'|}{|V-W|} \leq \frac{cn}{\frac{1}{2}pn} = 4c^2k^3 = c'$. (The proof of this lemma would have been confounded if we had used $\text{var}(F)$ to keep track of variables instead of V .) The probability that $A_{d,c,k}(F, V)$ finds a solution, assuming each call to `find_restriction` succeeds, is then

$$2^{-(n-f+(1-a_{d-1,c'})f)} = 2^{-(n-a_{d-1,c'}f)} \geq 2^{-(1-\frac{1}{2}pa_{d-1,c'})n},$$

and the lemma follows. $\square$

Lemma 8.4. $\forall d, c \geq 2$,

$$a_{d,c} \geq 1/O(c^{2^{d-2}-1} \lg^{3 \cdot 2^{d-2}-2} c),$$

where the constant in the big-Oh depends only on d .

Proof. We use induction to show that for each d , k can be chosen to be $O(\lg c)$ so as to satisfy the hypothesis of lemma 8.1 and $a_{d,c} \geq 1/O(c^{f(d)} \lg^{g(d)} c)$ for some functions f, g where the constants in the big-Ohs depend only on d .

$a_{2,c,k} = \frac{1}{k}$. From lemma 8.1, we need to choose k such that $k \geq O(\lg \frac{c}{a_{2,2c,k}}) = O(\lg c + \lg k)$. So $k = O(\lg c)$ suffices, and we conclude that $a_{2,c} \geq 1/O(\lg c)$. So $f(2) = 0, g(2) = 1$. This completes the base case.

From the inductive hypothesis and lemma 8.3,

$$\begin{aligned} a_{d,c,k} &\geq 1/O(ck^3(c^2k^3)^{f(d-1)} \lg^{g(d-1)}(c^2k^3)) \\ &= 1/O(c^{2f(d-1)+1} k^{3f(d-1)+3} \lg^{g(d-1)}(ck)). \end{aligned}$$

To use lemma 8.1, we need to choose k such that

$$\begin{aligned} k &\geq O(\lg(c^{2f(d-1)+2} k^{3f(d-1)+3} \lg^{g(d-1)}(ck))) \\ &= O(\lg c + \lg k). \end{aligned}$$

So $k = O(\lg c)$ suffices, and we conclude that

$$a_{d,c} \geq 1/O(c^{2f(d-1)+1} \lg^{3f(d-1)+3+g(d-1)} c).$$

So we have the recurrence

$$\begin{aligned} f(d) &= 2f(d-1) + 1 & f(2) &= 0 \\ g(d) &= g(d-1) + 3f(d-1) + 3 & g(2) &= 1 \end{aligned}$$

which has solution $f(d) = 2^{d-2} - 1, g(d) = 3 \cdot 2^{d-2} - 2$. $\square$

Theorem 8.5. $\forall d, c \geq 2$, the probability that $A_{d,c}(F, V)$ finds some solution is $\geq 2^{-(1-\alpha)n}$ where

$$\alpha \geq 1/O(c^{2^{d-2}-1} \lg^{3 \cdot 2^{d-2}-2} c),$$

where the constant in the big-Oh depends only on d .

Proof. This is a corollary from the previous lemma together with the following: `find_restriction` is called $\leq d$ times, each with success probability $\geq \frac{1}{2}$, so the probability that in every call it succeeds is $\geq 2^{-d}$, a penalty that can be absorbed into the big-Oh in the theorem statement. $\square$

Again, we are not asserting that every solution is found with this probability, just that some is, and this asymmetric property is inherited from the PPZ algorithm.

8.4 Open problems

Can `find_restriction` be derandomized without too much performance penalty? Can we find a nontrivial algorithm for the case where d grows very slowly with n ?

Acknowledgements

Material here is joint work with Russell Impagliazzo and Ramamohan Paturi and will appear in [CPI09].

```

 $A_{d,c}(F, V)$  //  $F$  has depth  $d$ ,  $\text{var}(F) \subseteq V$ ,  $|V| = n$ ,  $|F| \leq cn$ 
  choose  $k, q$  // as some function of  $d, c$ 
  while  $\exists$  bottom gate  $g$  in  $F$  of fan-in  $> k$ 
    let  $h$  be a subgate of  $g$  of size  $k$ 
     $b \leftarrow \begin{cases} 1 & \text{with probability } 1 - q \\ 0 & \text{with probability } q \end{cases}$ 
     $b' \leftarrow \begin{cases} 1 & \text{if } h \text{ is an AND gate} \\ 0 & \text{if } h \text{ is an OR gate} \end{cases}$ 
     $F \leftarrow F|(h = b \text{ XOR } b')$ 
    if  $b = 0$ ,  $V \leftarrow V - \text{var}(h)$ 
    if  $|F| > 2c|V|$  // guess assignment
      choose  $a \in_u 2^{\text{var}(F)}$ 
      if  $F(a) = 1$ , return  $a$ 
      return “probably not satisfiable”
  return  $A_{d,2c,k}(F, V)$ 

```

```

 $A_{d,c,k}(F, V)$  //  $F$  also has bottom fan-in  $\leq k$ 
  if  $d \leq 2$ , return  $\text{PPZ\_main}(F)$ 
  choose  $p, c'$  // as functions of  $d, c, k$ 
   $W \leftarrow \text{find\_restriction}(F, V, p)$ 
  choose  $a \in_u 2^W$ 
  // the bottom level gates of  $F|a$  are trivial
   $F' \leftarrow F|a$  but collapsing the bottom level
  return  $A_{d-1,c'}(F', V - W)$ 

```

```

 $\text{find\_restriction}(F, V, p)$  //  $|V| = n$ 
   $B \leftarrow \{\text{bottom gates of } F\}$ 
   $U \leftarrow \text{random subset of } V \text{ of size } (1 - p)n$ 
   $G \leftarrow \{g \in B \mid |\text{var}(g) - U| > 1\}$ 
   $U' \leftarrow \text{var}(G)$ 
  if  $|U'| \leq \frac{1}{2}pn$ , return  $U \cup U'$ 
  else die // algorithm fails

```

Figure 8.1: linear size, constant depth circuit solver

```

 $A_{d,c}(F, V)$ 
  if  $|F| \geq 2c|V|$  // solve by brute force
    for each  $a \in 2^{\text{var}(F)}$ 
      if  $F(a)$ , return 1
    return 0
  choose  $k$  // as some function of  $d, c$ 
  if  $\exists$  bottom gate  $g$  in  $F$  of fan-in  $> k$  // branch
    let  $h$  be a subgate of  $g$  of size  $k$ 
     $(V_0, V_1) \leftarrow \begin{cases} (V, V - \text{var}(h)) & \text{if } h \text{ is an AND gate} \\ (V - \text{var}(h), V) & \text{if } h \text{ is an OR gate} \end{cases}$ 
    if  $A_{d,c}(F|(h=1), V_1)$ , return 1
    if  $A_{d,c}(F|(h=0), V_0)$ , return 1
    return 0
  return  $A_{d,2c,k}(F, V)$ 

 $A_{d,c,k}(F, V)$ 
  if  $d \leq 2$ , return PPZ( $F$ )
  choose  $p, c'$  // as functions of  $d, c, k$ 
   $W \leftarrow \text{find\_restriction}(F, V, p)$  // same subroutine as in figure 8.1
  for each  $a \in 2^W$ 
    // the bottom level gates of  $F|a$  are trivial
     $F' \leftarrow F|a$  but collapsing the bottom level
    if  $A_{d-1,c'}(F', V - W)$ , return 1
  return 0

```

Figure 8.2: linear size, constant depth circuit solver, backtracking version

Chapter 9

Unique satisfiability

We saw that the algorithms PPZ and PPSZ could exploit a lack of solutions: PPZ found a solution with higher probability when it had many directions of isolation, and PPSZ found a solution with higher probability when it was very far from any other solutions. This makes it seem like the fewer solutions, the more exposed they are and the faster we can find them. But contradictorily, LocalSearch seemed to benefit from more solutions, and we showed in theorem 6.1 that PPZ actually does benefit from more solutions. All of this leaves us in a lurch: do more solutions help or not? It is hard to believe that the answer could possibly be 'no', that all algorithms will, for some strange configuration of solutions, get stuck between equally attractive ones, and yet that there are so few solutions that the algorithm cannot be sped up by random guessing.

We suspect that the hardest case for k -SAT is when an algorithm is presented with a needle in a haystack, i.e. there is ≤ 1 solution and therefore that the solution space has essentially no structure at all.

To prove such a statement, the most obvious method of attack is to efficiently reduce from the general case to the unique solution case by restricting the solution space. As we do not have deep insight into the structure of the problem, we will form such restrictions blindly. The next section explains the first powerful such technique, the Valiant-Vazirani reduction. In §9.2, we extend this idea to get stronger results for k -SAT.

9.1 Valiant-Vazirani reduction

$\mathcal{B}$ -Formula-SAT (*Circuit-SAT*) is the problem of deciding whether a Boolean formula (circuit) constructed over the basis $\mathcal{B}$, not required to be in CNF, has a satisfying assignment. If $\mathcal{B}$ is not specified, it is assumed to be $\{\wedge, \vee, \neg\}$. Valiant and Vazirani in [VV86] gave a randomized reduction from Formula-SAT to *Unique-Formula-SAT*, the promise version, restricted to inputs

that have ≤ 1 solution. We make this more formal below in corollaries 9.4, 9.5.

The main tool we will need is the following notion, introduced by [CW77]. A family of hash functions $H \subseteq B^A$ is *2-universal* iff $\forall x, y \in A, x \neq y$

$$Pr_{h \in_u H}(h(x) = h(y)) \leq \frac{1}{|B|}.$$

Although the definition allows it, it is not possible to make this worst-case collision probability too much smaller than $\frac{1}{|B|}$ when $|A| \gg |B|$.

Lemma 9.1. *Let $H \subseteq B^A$ be a family of hash functions. Then*

$$\max_{x \neq y} Pr_{h \in_u H}(h(x) = h(y)) \geq \frac{1}{|B|} - \frac{1}{|A|}.$$

Proof. Let $h \in H$ and $x_i = |\{x \in A \mid h(x) = i\}|$. Then $\sum_{i \in B} x_i = |A|$ and $\sum_{i \in B} \binom{x_i}{2} / \binom{|A|}{2}$ is the probability of a collision between random distinct x, y . The method of Lagrange multipliers can be used to show that this is minimized (over $\mathbb{R}^B$) when each $x_i = \frac{|A|}{|B|}$. So if $h \in_u H$, then

$$\begin{aligned} \max_{x \neq y} Pr_h(h(x) = h(y)) &\geq E_{x \neq y} Pr_h(h(x) = h(y)) \\ &= E_h Pr_{x \neq y}(h(x) = h(y)) \\ &\geq \frac{|B| \binom{\frac{|A|}{|B|}}{2}}{\binom{|A|}{2}} \geq \frac{1}{|B|} - \frac{1}{|A|}. \end{aligned}$$

□

H is *strongly 2-universal* iff $\forall x_1, x_2 \in A, x_1 \leq x_2$ and $y_1, y_2 \in B$,

$$Pr_{h \in_u H}(h(x_1) = y_1 \wedge h(x_2) = y_2) = \frac{1}{|B|^2}.$$

Of course, H is strongly 2-universal implies H is 2-universal, but not conversely.

Theorem 9.2 (Valiant-Vazirani). *For each $n, m \in \mathbb{N}$, let $H_{n,m}$ be a 2-universal family of hash functions mapping $2^n \rightarrow 2^m$. By selecting $m \in_u [2, n+1]$, $h \in_u H_{n,m}$, $b \in_u 2^m$, we have that $\forall \emptyset \neq S \subseteq 2^n$,*

$$Pr_{h,b}(|\{x \in S \mid h(x) = b\}| = 1) \geq \frac{1}{O(n)}.$$

Proof. We will first guess the size of S to within a factor of 2, which can be done with probability about $\frac{1}{n}$, and then reduce the size of S down to just 1 with probability $\geq \Omega(1)$ by hashing down to $m \approx \lg |S|$ bits. Details follow.

Since we choose $m \in_u [2, n+1]$, with probability $\geq \frac{1}{n}$,

$$\frac{1}{4} \leq |S|2^{-m} \leq \frac{1}{2}. \tag{9.1}$$

From now on in this proof, all probabilities and expectations will be implicitly conditioned on (9.1) occurring.

Let $B = \{b' \in 2^m \mid \exists! x \in S \ h(x) = b'\}$. Then

$$\begin{aligned} E(|B|) &= \sum_{x \in S} Pr(\forall y \in S - \{x\} \ h(x) \neq h(y)) \quad \text{by linearity of expectation} \\ &= \sum_{x \in S} (1 - Pr(\exists y \in S - \{x\} \ h(x) = h(y))) \\ &\geq |S|(1 - |S|2^{-m}) \quad \text{by the union bound and 2-universality.} \end{aligned}$$

So

$$\begin{aligned} Pr(b \in B) &= E_B(Pr_b(b \in B)) \\ &= 2^{-m} E(|B|) \\ &\geq |S|2^{-m}(1 - |S|2^{-m}). \end{aligned}$$

Letting $\delta = |S|2^{-m}$, we want to maximize $\delta(1 - \delta)$ subject to $\delta > 0$. This happens when $\delta = \frac{1}{2}$. We may not be able to choose δ to be exactly $\frac{1}{2}$, but by (9.1),

$$Pr(b \in B) = \delta(1 - \delta) \geq \frac{1}{4} \left(1 - \frac{1}{4}\right) = \frac{3}{16}.$$

□

If we have a family $H_{n,m}$ mapping $2^n \rightarrow 2^m$ for each $n, m \in \mathbb{N}$, say that the entire family is *computationally efficient* iff there are algorithms S, E such that on input n, m , S can sample $h \in_u H_{n,m}$ in time $\text{poly}(n, m)$ and on input h, x , E can evaluate $h(x)$ in time $\text{poly}(n, m)$.

To apply theorem 9.2 we need to construct a computationally efficient 2-universal hash family $H_{n,m}$. Interpreting Ax as a matrix product over $\mathbb{Z}_2$,

$$H_{n,m} = \{(x \mapsto Ax) \mid A \in 2^{m \times n}\} \quad (9.2)$$

is computationally efficient and 2-universal since $\forall x, y \in 2^n, x \neq y$,

$$Pr(Ax = Ay) = Pr(A(x - y) = 0) = 2^{-m}.$$

Theorem 9.2 selects $m \in_u [2, n + 1]$, so this choice of hash family requires $\Omega(n^2)$ random bits, much more than necessary, as we show below. In exchange, it is conceptually simple.

Another reasonable choice is the convolution (over $\mathbb{Z}_2$) with a random bit vector of length $n + m - 1$. That is, if $x = (x_0, \dots, x_{n-1})$, $a = (a_0, \dots, a_{n+m-2})$ and $a \otimes x = (\sum_{j=0}^{n-1} a_{i+n-1-j} x_j)_{i=0}^{m-1}$, then we claim the following.

Theorem 9.3. *The following hash family is computationally efficient and 2-universal.*

$$H_{n,m} = \{(x \mapsto a \otimes x) \mid a \in 2^{n+m-1}\} \quad (9.3)$$

Proof. To see 2-universality, note that convolution is linear, so it is sufficient to show that $\forall x \neq 0$ $Pr(a \otimes x = 0) = 2^{-m}$. Fixing $x \neq 0$, we show by induction on m that the bits $(b_0, \dots, b_{m-1}) = a \otimes x$ are uniform and independent. The first $m-1$ bits $(b_0, \dots, b_{m-2})$ is the convolution of the first $n+m-2$ bits of a with x and so, by the induction hypothesis, are uniform and independent. The last bit $b_{m-1} = \sum_{j=0}^{n-1} a_{n+m-2-j}x_j$. Let $j_0 = \min\{j \mid x_j = 1\}$. Then $a_{n-m-2-j_0}$ is uniform and independent of $(b_0, \dots, b_{m-2})$, and so b_{m-1} is uniform and independent of $(b_0, \dots, b_{m-2})$. $\square$

Corollary 9.4. $\text{Unique-3-SAT} \in \text{Promise-BPP} \Rightarrow \text{NP} = \text{RP}$.

Proof. ($\supseteq$) Follows by definition of NP, RP.

($\subseteq$) Let U solve Unique-3-SAT in time $\text{poly}(n)$ with 2-sided error $\leq \frac{1}{3}$. By using self-reducibility and repetition, we may assume wlog that U has 0 soundness error (even when the promise is violated) and completeness error $\leq \frac{1}{2}$.

Upon input F with $n = |\text{var}(F)|$, generate $m \in_u [2, n+1]$, $b \in_u 2^m$. Let $H_{n,m}$ be a computationally efficient hash family mapping $2^n \rightarrow 2^m$, e.g. (9.2). Computational efficiency allows us to sample and evaluate $h \in_u H_{n,m}$ in time $\text{poly}(n)$. Using Cook's computation tableau reduction, we can represent $h(x) = b$ by a 3-CNF G with $\text{poly}(n)$ extra variables such that $F \wedge h(x) = b$ is uniquely satisfiable iff $F \wedge G$ is. By theorem 9.2, if F is satisfiable, then $F \wedge G$ is uniquely satisfiable with probability $\geq \frac{1}{O(n)}$; and so if F is satisfiable, then $U(F \wedge G) = 1$ with probability $\geq \frac{1}{O(n)}$. By repeating $O(n)$ times, we can decrease the completeness error to $O(1)$. The total running time is $\text{poly}(n)$. So 3-SAT $\in$ RP. Since 3-SAT is NP-complete, we have the result. $\square$

Corollary 9.5. Let $\mathcal{B} = \{\wedge, \vee, \neg, \oplus\}$. Then

$$\begin{aligned} C_{\text{Circuit-SAT}} &= C_{\text{Unique-Circuit-SAT}} \\ C_{\mathcal{B}\text{-Formula-SAT}} &= C_{\mathcal{B}\text{-Unique-Formula-SAT}}. \end{aligned}$$

Proof. ($\geq$) Obvious.

($\leq$) Let U solve Unique-Circuit-SAT in time $\text{poly}(|G|)2^{cn}$ on input G . By using self-reducibility and repetition, we may assume wlog that U has 0 soundness error (even when the promise is violated) and exponentially small completeness error. Upon input F with $n = |\text{var}(F)|$, generate $m \in_u [2, n+1]$. Let $H_{n,m}$ be a computationally efficient 2-universal hash family, $h \in_u H_{n,m}$, $b \in_u 2^m$. Represent $F \wedge h(x) = b$ by a formula G , and return $U(G)$. The soundness error is 0 and by repeating $O(n)$ times, we can decrease the completeness error to $O(1)$. Since we can represent $h(x) = b$ with a poly sized circuit with no new variables, the total running time is $\leq \text{poly}(|F|)2^{cn}$.

The same proof works for the second statement if we choose $H_{n,m}$ to specifically be one of the two 2-universal hash families (9.2),(9.3). In either case, we can represent $h(x) = b$ with a poly sized formula over the basis $\{\wedge, \vee, \neg, \oplus\}$ with no new variables. $\square$

9.1.1 Hash function representation

Let P be a Boolean satisfiability problem and Q be a predicate on Boolean variables X . We can say that instance F of P *represents* Q iff F has variables $X \cup Y$ (the Y can be thought of as auxiliary) and $\forall a \in 2^X$ ($Q(a)$ iff a can be extended to a solution of F). (This is similar to saying that Q is first-order definable, but we only allow a single existential quantifier and expressiveness of the “formula” is limited by P .) We can make this notion a bit stricter: say F *parsimoniously represents* Q iff F represents Q and $\forall a \in 2^X$ $|\text{sol}(F|a)| \leq 1$. The efficiency of the representation can be measured by how small Y is and how quickly we can compute F given some other representation of Q .

In the proof of corollary 9.5, we were able to use theorem 9.2 to relate $c_{\text{Circuit-SAT}}$, $c_{\text{Unique-Circuit-SAT}}$ because we could efficiently represent as a circuit constraints of the form $h(x) = b$, where $h \in H_{n,m}$ and $H_{n,m}$ is some 2-universal hash family. ‘Efficient’ here meant that the representation was of size $\text{poly}(n)$ and there were no new variables.

Corollary 9.4 used a much weaker notion of efficiency, only requiring that the representation take $\text{poly}(n)$ time to compute. But this notion of efficiency tells us little about the relationship between $c_{3\text{-SAT}}$ and $c_{\text{Unique-3-SAT}}$.

If P is a Boolean satisfiability problem, in order to establish a result like

$$c_P \leq (1 + \alpha)c_{\text{Unique-}P} \quad (9.4)$$

using theorem 9.2, we would need to be able to take an arbitrary instance F of P with n variables and parsimoniously represent $F \wedge h(x) = b$ as an instance G of P , using $\leq \alpha n$ auxiliary variables, and taking only $2^{o(n)}$ time to compute G . If we could make α arbitrarily close to 0, we could establish $c_P = c_{\text{Unique-}P}$. Other tradeoffs between auxiliary variables and time to compute the representation that could still lead to (9.4) are possible; e.g. if we used a P or Unique- P solver in our construction of the representation itself.

In this section, we discuss some of the difficulties in constructing such a representation for problems more restrictive than those in corollary 9.5. Later, in §9.3, we will give a more compelling reason to abandon such strategies: (1) we will show that for $P = k\text{-SAT}$, α cannot be made any smaller than $\frac{1}{80k}$ using any *oblivious* isolation technique, i.e. one that attempts to isolate a solution while ignoring the input formula except to compute the number of input variables; and (2) [Tra08] comes within $O(\lg k)$ of this lower bound anyway.

One way to represent the hash function (9.2) $Ax = b$ by a 3-CNF is by introducing $O(n^2)$ new variables $y_{i,j}, i \in [m], j \in [2, n]$ and representing the following constraints with 3-clauses:

$$\begin{aligned} y_{i,2} &= A_{i,1}x_1 \oplus A_{i,2}x_2 \\ y_{i,j} &= y_{i,j-1} \oplus A_{i,j}x_j \\ y_{i,n} &= b_i \quad \text{for } i \in [m], j \in [3, n]. \end{aligned}$$

The resulting formula G is uniquely satisfiable iff F is, but since we needed $\Omega(n^2)$ new variables, we cannot conclude (9.4) for $P = 3\text{-SAT}$ for any constant factor α .

The convolution hash family (9.3) can be computed using a Fast Fourier Transform circuit of size $O(n \lg^3 n)$, (To see this, suppose $w \log n$ is a power of 2. The circuit can hardcode a generator of the multiplicative group $\mathbb{F}_n^\times$, and use this to carry out FFT in $O(n \lg n)$ field operations, each of which can be done with $O(\lg^2 n)$ bit operations. Recursion can reduce the complexity somewhat, but not below $O(n \lg^2 n)$.) which implies that it can be represented by a 3-CNF with only $O(n \lg^3 n)$ extra variables. Although better than before, we still cannot conclude (9.4) for any constant α . In fact, we know of no even approximately 2-universal hash family mapping $2^n \rightarrow 2^m$ with $m \approx n$ representable by a CNF of size $2^{o(n)}$ and with only $O(n)$ variables.

We now give a sequence of lower bounds on the complexity of hash function representation with increasingly sophisticated arguments, beginning with a trivial exponential lower bound on the size of a CNF F that (not necessarily parsimoniously) represents that the parity of the first n variables is $b \in 2$.

Lemma 9.6 ([FSS84]). *$\forall b \in 2$, there is a unique n variable CNF F that represents that the parity of the variables is b using no auxiliary variables, and it contains exactly 2^{n-1} clauses.*

Proof. To show the existence of F , for each assignment to the variables that makes the parity $1-b$, create a clause that forbids this assignment. To show uniqueness, suppose that the solutions to some $G \in \text{CNF}$ are the assignments with parity b and let C be a clause in G . Choose any assignment a that makes C false. Since flipping any variable makes C true, C must contain every variable, and so it excludes exactly 1 solution. This shows that $G = F$. $\square$

Corollary 9.7. *If $F \in \text{CNF}$ represents that the parity of $x_1, \dots, x_n$ is $b \in 2$ and uses auxiliary variables $y_1, \dots, y_l$, then $|F| \geq 2^{\frac{n-1}{2^l}}$.*

Proof. We can eliminate the y_i from F by applying resolution repeatedly: define

$$\begin{aligned} F_0 &= F \\ F_i &= F_{i-1} - \{C \mid y_i \in \text{var}(C)\} \cup \{\text{resolve}(C, D) \mid y_i \in C, \bar{y}_i \in D \\ &\quad \text{and } C, D \in F \text{ are resolvable}\} \quad \text{for } i \in [l]. \end{aligned}$$

Then $\text{var}(F_i) = \{x_1, \dots, x_n, y_{i+1}, \dots, y_l\}$, $|F_i| \leq |F_{i-1}|^2$, and a is a solution to F_i iff a can be extended to a solution to F_{i-1} . Letting $m = |F| = |F_0|$, we have $|F_l| \leq m^{2^l}$. By lemma 9.6, $m \geq |F_l|^{\frac{1}{2^l}} \geq 2^{\frac{n-1}{2^l}}$. $\square$

If we attempt to represent $Ax = b$ as the conjunction F of independent CNF representations of each of its m linear equations, i.e. each equation representation as a CNF will use its own separate set of auxiliary variables, and we use only $O(n)$ auxiliary variables in total,

then some equation must be represented with only $O(1)$ auxiliary variables. Since A is dense, by corollary 9.7, $|F| \geq 2^{\Omega(n)}$.

Of course, we are not limited to representing $Ax = b$ as the conjunction of *independent* representations of each of its m equations. If, for example, there were a permutation of the rows of A such that the Hamming distance between adjacent rows were $O(1)$, then we could represent $Ax = b$ using only $O(n)$ extra variables. The Chernoff bound (theorem 2.7) can be used to show, however, that it is exponentially unlikely for any 2 rows to be closer than $\frac{n}{4}$.

It should also be noted that we may not need to compute Ax at all with a CNF, since we are only trying to *verify* that $Ax = b$. By *compute a function h with CNFs*, we mean that for each output bit h_i of h , there is a CNF F_i representing $h_i(x) = 1$. However, there are functions whose output may indeed be easier to verify than compute, e.g. if $h(x)$ is the factorization of x , treated as an integer. And so the lower bound in corollary 9.10 below makes one wonder whether there might not be a hash function h with high sensitivity (defined below) so that for each x, b , $h(x) = b$ is efficiently representable as a small CNF with few auxiliary variables even though h itself cannot be computed by small CNFs with few auxiliary variables.

We can use the notion of function sensitivity to get a stronger lower bound than corollary 9.7 provides. Let $f : 2^n \rightarrow 2$. For $x \in 2^n, i \in n$, let $x \oplus i$ denote x but with bit i flipped. Let $\delta(f, x, i)$ denote the predicate $f(x \oplus i) \neq f(x)$. The *sensitivity* of f at $x \in 2^n$ is

$$s(f, x) = |\{i \in n \mid \delta(f, x, i)\}|,$$

and the *sensitivity* of f is

$$s(f) = E_{x \in_u 2^n}(s(f, x)). \quad (9.5)$$

There are several other characterizations of sensitivity, some highly nontrivial (e.g. [LMN93]).

We can define the sensitivity of a function $h : 2^n \rightarrow 2^m$ with multiple output bits as $s(h) = \max_{i \in m} s(h_i)$ where h_i is the i th output bit of h . Note that if h is strongly 2-universal, then each h_i is strongly 2-universal as well.

Lemma 9.8 ([MNT93]). *Let H be a strongly 2-universal family of functions mapping $2^n \rightarrow 2^m$. Then $\exists h \in H$ so that $\geq \frac{m}{3}$ of its output bits have sensitivity $\geq \frac{n}{4}$. In particular, $s(h) \geq \frac{n}{4}$.*

Proof. Choosing $h \in_u H, x \in_u 2^n, j \in_u m$, we have that $\forall i \in m$,

$$\begin{aligned} E_h(s(h_i)) &= E_{h,x}(nPr_j(\delta(h_i, x, j))) \\ &= nE_{x,j}(Pr_h(\delta(h_i, x, j))) = \frac{n}{2}. \end{aligned}$$

So

$$\begin{aligned} Pr_h\left(s(h_i) \geq \frac{1}{4}n\right) &= 1 - Pr_h\left(n - s(h_i) > \frac{3}{4}n\right) \\ &\geq 1 - \frac{2}{3} \quad \text{by Markov's inequality} \\ &= \frac{1}{3}. \end{aligned}$$

So $E_h(\Pr_i(s(h_i) \geq \frac{n}{4})) = E_i(\Pr_h(s(h_i) \geq \frac{n}{4})) \geq \frac{1}{3}$. Some $h \in H$ achieves this average, and for this h , $\Pr_i(s(h_i) \geq \frac{n}{4}) \geq \frac{1}{3}$. $\square$

Results from [LMN93] show that a depth d circuit computing $f : 2^n \rightarrow 2$ requires $2^{\Omega(s(f)^{1/d})}$ gates. [MNT93] use this and lemma 9.8 to conclude that if H is a strongly 2-universal family mapping $2^n \rightarrow 2^m$, then $\exists h \in H$ so that a depth d circuit computing h requires $2^{\Omega(n^{1/d})}$ gates. [LMN93] uses a very complex argument involving Fourier transforms of Boolean functions, and this only gives a lower bound of $\Omega(n^{1/3})$ on the number of auxiliary variables required to compute h by a k -CNF. Though large, this is still sublinear, and therefore not very useful to us. We can use the following much sharper theorem of Boppana to increase this to $\Omega(\frac{n}{k})$.

Theorem 9.9 ([Bop97]). *If $f : 2^n \rightarrow 2$ is computable by a circuit of depth d , width w , and bottom fan-in k , then $s(f) \leq 2k(10 \lg(4w))^{d-2}$.*

Corollary 9.10. *If H is a family of strongly 2-universal hash functions mapping $2^n \rightarrow 2^m$, and every $h \in H$ is computable by k -CNFs with $\leq l$ auxiliary variables, then $l \geq \frac{n}{80k} - O(k \lg n)$.*

Proof. By lemma 9.8, $\exists h \in H, i \in m$ $s(h_i) \geq \frac{n}{4}$. By assumption, there is a k -CNF ϕ representing h_i using $\leq l$ auxiliary variables Y . Let $C = \bigvee_{a \in 2^Y} \phi|a$. Then C is a circuit of depth 3, width $w \leq 2^l n^{O(k)}$, and bottom fan-in k computing h_i . By theorem 9.9, $\frac{n}{4} \leq s(h_i) \leq 20k \lg(4w)$, which implies $\frac{n}{80k} \leq \lg(4w) \leq l + O(k \lg n)$. $\square$

However, we are not satisfied because when using CNFs in the Valiant-Vazirani reduction,

- the hash family need not be strongly 2-universal, maybe it is only approximately so. We only need small collision probability, not perfection. Approximately strongly 2-universal does *not* seem to provide any useful lower bound on the sensitivity of the output bits of some member of the family.
- as we have already noted, the CNF need not compute the h_i , it only needs to *verify* that $h(x) = b$.
- the reduction does not necessarily fail just because one hash function in the family fails to be efficiently representable. Maybe the reduction can detect such a condition and simply fail in those cases.

It is interesting to note that theorem 9.9 is proven with a simple switching lemma argument. By using a more direct such argument, in §9.3 we will get essentially the same bound as corollary 9.10 gives, but overcome all the deficiencies stated above.

9.1.2 Derandomization of the Valiant Vazirani reduction for $2^{o(n)}$ solutions

If we restrict to Boolean satisfiability problems promised to have only $2^{o(n)}$ solutions, then we can derandomize the Valiant-Vazirani reduction in the following sense. Suppose P is a Boolean satisfiability promise problem where instances with n variables are promised to have $\leq 2^{o(n)}$ solutions. Also assume that P allows efficient parsimonious representation (defined in §9.1.1) of $F \wedge h(x) = b$ where F is an instance of I and h is an arbitrary $O(n) \times n$ linear hash function, so that the representation is computable in time $2^{o(n)}$ and has only $o(n)$ auxiliary variables. Then

$$dc_P = dc_{\text{Unique-}P}.$$

In reality we need not require being able to represent an *arbitrary* linear hash function, since we will choose ones from a particular distribution, but anyway these hypotheses are sufficient.

We will make use of a hash function, constructed from the techniques of [NN93], that is almost 2-universal, in that the collision probability is $\leq \epsilon$, and that uses only $O(\lg n + \lg \frac{1}{\epsilon})$ random bits. We will briefly sketch how such a construction is possible.

Suppose for simplicity that $n = 2^i$ is a power of 2. Using theorem 2.8, we can construct l random variables that are (1) k -wise independent and (2) each uniform on n (these 2 conditions together we call *k-universal*), by using $O(k(\lg n + \lg l))$ random bits. The construction there chooses a random polynomial in $\mathbb{F}_{n'}[x]$ of degree $< k$, where $n' = 2^{i'}$, $i' = \max\{i, \lceil \lg l \rceil\}$, evaluates it on l distinct points, then divides each random variable (over $\mathbb{Z}$, not $\mathbb{F}_{n'}$) by $\frac{n'}{n}$ and takes the floor to get the correct range. This can be accomplished with a bit shift. Other constructions are possible, e.g. [CG89].

```

choose random variables
 $z_1, \dots, z_n \in [n]$  to be 2-universal
 $u_1, \dots, u_n \in 2$  to be 7-universal
 $c_1, \dots, c_{\lg n} \in 2$  to be fully independent
for  $j \leftarrow 1, \dots, n$ 
   $i_j \leftarrow \min\{i \geq 1 \mid z_j \leq 2^i\}$ 
   $r_j \leftarrow u_j c_{i_j}$ 
return  $r = (r_1, \dots, r_n)$ 

```

Figure 9.1: generating r

We use the (admittedly obscure) algorithm in figure 9.1, adapted from [NN93], to generate a vector $r \in 2^n$. This requires $O(n)$ operations (such as multiplication in $\mathbb{F}_n$) on words

of size $O(\lg n)$ and using only $O(\lg n)$ random bits. [NN93] show that this r has the following property: $\forall x \neq 0 \Pr(r \cdot x = 0) \leq \frac{1}{8}$. For any constant $\alpha \in (0, 1)$, by gathering $O(1)$ independent copies of such vectors, we can construct a $O(1) \times n$ random matrix A still using only $O(\lg n)$ random bits such that $\forall x \neq 0 \Pr(Ax = 0) \leq \alpha$.

Since A is linear, $\max_{x \neq 0} \Pr(Ax = 0) \leq \alpha$ is the collision probability if we used A as a hash function. We would like to reduce this collision probability further to only exponentially small in some parameter l . But using independent copies of the above construction would be wasteful, requiring $\Omega(l \lg n)$ random bits. Instead, letting S be the sample space in the construction of A above (so that $|S| \leq \text{poly}(n)$), [NN93] construct a $O(1)$ -regular expander G with node set S and let a random walk of length $l = O(\lg \frac{1}{\epsilon})$ on G define a sequence of samples from S . These are used to construct (dependent) $O(1) \times n$ matrices $A_1, \dots, A_l$. Stacking the A_i vertically gives a new matrix A with the following properties.

Theorem 9.11. *Let $n \in \mathbb{N}, \epsilon \in (0, 1)$. We can construct a matrix $A \in 2^{m \times n}$ such that*

- $\forall x \neq 0 \Pr(Ax = 0) \leq \epsilon$
- $m = O(\lg \frac{1}{\epsilon})$
- *A can be constructed using $O(n \lg \frac{1}{\epsilon})$ operations on words of size $O(\lg n)$ (or in time $n \lg \frac{1}{\epsilon} \text{poly}(\lg n)$ using $O(1)$ -bit operations; see [GK06], §4.1, for details on the computational complexity of expander graph constructions) and using $O(\lg n + \lg \frac{1}{\epsilon})$ random bits.*

For a Boolean satisfiability problem P and function $f : \mathbb{N} \rightarrow \mathbb{N}$, let $P_{f \text{ sols}}$ be the promise version where instances with n variables are promised to have $\leq f(n)$ solutions. So, e.g., $\text{Unique-}P = P_{1 \text{ sols}}$.

Theorem 9.12 (Derandomization of Valiant-Vazirani). *Let $\mathcal{B} = \{\wedge, \vee, \neg, \oplus\}$. For any $f \in 2^{o(n)}, f \geq 1$, computable in time $2^{o(n)}$,*

$$\begin{aligned} dc_{\text{Circuit-SAT}_{f \text{ sols}}} &= dc_{\text{Unique-Circuit-SAT}} \\ dc_{\mathcal{B}\text{-Formula-SAT}_{f \text{ sols}}} &= dc_{\text{Unique-}\mathcal{B}\text{-Formula-SAT}}. \end{aligned}$$

Proof. ($\geq$) Obvious.

($\leq$) Let F be a circuit with $S = \text{sol}(F)$, $|S| \leq f(n)$, $n = |\text{var}(F)|$. For an ϵ to be determined later, construct $m \times n$ matrix A as in theorem 9.11. Let

$$B = \{b' \in 2^m \mid \exists! x \in S \ Ax = b'\}.$$

Then

$$\begin{aligned}
E(|B|) &= \sum_{x \in S} \Pr(\forall y \in S - \{x\} \ Ax \neq Ay) \\
&= \sum_{x \in S} (1 - \Pr(\exists y \in S - \{x\} \ Ax = Ay)) \\
&\geq |S|(1 - |S|\epsilon).
\end{aligned}$$

Choosing $\epsilon = \frac{1}{2f(n)}$, we have

$$\Pr_{b \in {}_u 2^m}(b \in B) \geq 2^{-m} E(|B|) > 0,$$

and so the probability that $F \wedge Ax = b$ is uniquely satisfiable is > 0 . The number of random bits needed to construct A, b is $O(\lg n + \lg f(n)) = o(n)$. We complete the proof by calling an oracle for Unique-Circuit-SAT for every possible setting of these random bits. The second statement follows in the same way since we can efficiently parsimoniously represent linear hash functions with $\mathcal{B}$ -Formulas. $\square$

Using an idea from Russell Impagliazzo, one can show that it is not possible to reduce the randomness used in the Valiant-Vazirani reduction to $< n$ bits in the general case. In the most abstract setting, the Valiant-Vazirani reduction uses r random bits somehow to select a set $T \subseteq 2^n$ and then provides a lower bound $p > 0$ on $\Pr(|S \cap T| = 1)$ independent of $\emptyset \neq S \subseteq 2^n$.

Theorem 9.13. *Suppose $T \subseteq 2^n$ is chosen according to some distribution on a sample space of size Q and $\forall \emptyset \neq S \subseteq 2^n \ \Pr(|S \cap T| = 1) > 0$. Then $Q \geq 2^n$.*

Proof. Considering S and T to be vectors in the vector space 2^{2^n} over the field $\mathbb{F}_2$, the hypothesis implies $\forall S \neq 0 \ \Pr(S \cdot T = 1) > 0$. Suppose indirectly that $Q < 2^n$ and let $\mathcal{T}$ be set of all possible values of T . Then $|\mathcal{T}| \leq Q$, which implies that $\dim \text{span}(\mathcal{T}) \leq Q$. So $\dim \text{span}(\mathcal{T})^\perp > 0$ and so has an element $S \neq 0$. But then $\Pr(S \cdot T = 1) = 0$, a contradiction. $\square$

9.2 Lower bound on the complexity of Unique- k -SAT

Recall our definition from (5.1):

$$\begin{aligned}
s_k &= c_{k\text{-SAT}} & s_\infty &= \lim_{k \rightarrow \infty} s_k \\
\sigma_k &= c_{\text{Unique-}k\text{-SAT}} & \sigma_\infty &= \lim_{k \rightarrow \infty} \sigma_k.
\end{aligned}$$

We suspect that Unique- k -SAT is as hard as k -SAT, namely $\forall k \geq 3 \ s_k = \sigma_k$. At the moment we do not know how to show this, but we can show that it is approximately true for large k . In particular, in this section we show that $s_\infty = \sigma_\infty$ (theorem 9.16). We will also show that the exponential time hypothesis (ETH), namely $s_3 > 0$, is equivalent to the exponential time hypothesis for Unique- k -SAT, namely $\sigma_3 > 0$ (theorem 9.17). Both results appeared in [CIKP08].

The Valiant-Vazirani reduction relied upon a hash function with low collision probability, and in §9.1.1, we noted the challenges of representing such a hash function with a k -CNF. We get around this by greatly relaxing our requirements on how low the collision probability needs to be. Instead of using the dense linear hash function (9.2), we will use a *sparse* linear hash function and show that with at least subexponential probability, this reduces the solution space to a ball of small radius. Randomly assigning a small number of variables further reduces this to just 1 solution with subexponential success probability. Whereas the Valiant-Vazirani reduction successfully isolates a solution with inverse linear probability, we will succeed with only subexponentially small probability, but this will be good enough for our purposes.

Unfortunately, this technique requires encoding the hash function as a k' -CNF for some k' somewhat larger than k , so we do not quite achieve $s_k = \sigma_k$. In §9.3, we will see that this is no accident, but a necessity of any technique that attempts to isolate solutions obliviously.

Lemma 9.14 (Isolation Lemma). $\forall k \in \mathbb{N}, \epsilon \in (0, \frac{1}{16}] \exists$ a randomized poly time algorithm $I_{k,\epsilon}$ that, given an n -variable k -CNF ϕ , outputs an n -variable k' -CNF ϕ' , with $k' = \max\{k, \lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \rceil\}$, such that

1. $\text{sol}(\phi') \subseteq \text{sol}(\phi)$, and
2. $\Pr(|\text{sol}(\phi')| \leq 1) \geq \frac{1}{32n} 2^{-4\epsilon \lg \frac{1}{\epsilon} n}$.

Proof. ϕ' will be the conjunction of ϕ and other clauses, so it is sufficient to prove the second claim in the case that ϕ is satisfiable.

First, we describe how to choose our hash function that concentrates solutions into a ball. Let $\{x_1, \dots, x_n\} = V = \text{var}(\phi)$, $S = \text{sol}(\phi) \neq \emptyset$. With probability $\geq \frac{1}{n}$, we can guess an integer $s \in [\lg |S|, \lg |S| + 1]$. For the rest of this proof, we assume that our guess is correct. For m to be determined later, construct the sparse $m \times n$ linear system $Ax = b$ over the field $\mathbb{F}_2$ as follows.

for $i \in [m]$
 choose $V_i \subseteq V$ of size k' uniformly at random
 choose $b_i \in_u 2$
 for $j \in V_i$, choose $A_{i,j} \in_u 2$
 for $j \in V - V_i$, $A_{i,j} \leftarrow 0$

Note that $Ax = b$ can easily be encoded as a k' -CNF. While A is unlikely to distinguish two vectors $x, y \in 2^V$ that are close to each other, it has a reasonable chance if they are distance $\text{dist}(x, y) \geq r \stackrel{\text{def}}{=} \lceil \epsilon n \rceil$ apart:

Lemma 9.15. $\forall x, y \in 2^V$ with $\text{dist}(x, y) \geq r$,

$$\Pr_A(Ax = Ay) \leq \left(\frac{e^\epsilon}{2}\right)^m.$$

Proof. For each $i \in [m]$,

$$\begin{aligned}
Pr((Ax)_i \neq (Ay)_i) &= Pr((A(x-y))_i \neq 0) \\
&= Pr((A(x-y))_i \neq 0 \mid (x-y)_{V_i} \neq 0) \\
&\quad \cdot Pr((x-y)_{V_i} \neq 0) \\
&= \frac{1}{2} Pr((x-y)_{V_i} \neq 0),
\end{aligned} \tag{9.6}$$

where $(x-y)_{V_i}$ is the vector $x-y$ restricted to the coordinates from V_i .

$$\begin{aligned}
Pr((x-y)_{V_i} = 0) &\leq \frac{\binom{n-r}{k'}}{\binom{n}{k'}} = \prod_{j=0}^{k'-1} \frac{n-r-j}{n-j} = \prod_{j=0}^{k'-1} \left(1 - \frac{r}{n-j}\right) \\
&\leq \left(1 - \frac{r}{n}\right)^{k'} \leq e^{-\frac{r}{n}k'} \leq e^{-\epsilon \frac{1}{\epsilon} \ln \frac{1}{\epsilon}} = \epsilon.
\end{aligned} \tag{9.7}$$

Combining (9.6), (9.7),

$$\begin{aligned}
Pr((Ax)_i = (Ay)_i) &\leq 1 - \frac{1}{2}(1 - \epsilon) \\
&= \frac{1}{2}(1 + \epsilon) \leq \frac{e^\epsilon}{2},
\end{aligned}$$

and so $Pr(Ax = Ay) \leq (\frac{e^\epsilon}{2})^m$. □

Define the set of *semi-isolated* solutions as

$$SI = \{x \in S \mid \forall y \in S \text{ with } \text{dist}(x, y) \geq r \ Ax \neq Ay\}.$$

Then for each $x \in S$,

$$\begin{aligned}
Pr(x \notin SI) &\leq \sum_{y \in S, \text{dist}(x, y) \geq r} Pr(Ax = Ay) \quad \text{from the union bound} \\
&\leq |S| \left(\frac{e^\epsilon}{2}\right)^m \quad \text{by lemma 9.15} \\
&\leq 2^{s-m(1-\frac{\epsilon}{\ln 2})} \\
&\leq \frac{1}{2} \quad \text{choosing } m = s + \lceil 2\epsilon s \rceil + 2.
\end{aligned} \tag{9.8}$$

So

$$\begin{aligned}
&Pr_{x \in {}_u S, A, b}(x \in SI \wedge Ax = b) \\
&= E_{x \in {}_u S}(Pr_{A, b}(Ax = b \mid x \in SI) Pr_A(x \in SI)) \\
&\geq 2^{-m-1} \quad \text{by (9.8) and independence of } b.
\end{aligned} \tag{9.9}$$

On the other hand, for fixed A, b , the number of semi-isolated solutions x such that $Ax = b$ is $\leq |B_{\epsilon n}(0^n)|$ since every pair of such solutions is distance $< r$ apart. So $X \stackrel{\text{def}}{=} |\{x \in SI \mid Ax = b\}| \leq 2^{h(\epsilon)n}$, by theorem 2.3. For any random variable X with an upper bound B ,

$E(X) \leq B \cdot \Pr(X > 0)$, so

$$\begin{aligned} & \Pr_{x \in {}_u S, A, b}(x \in \text{SI} \wedge Ax = b) \\ &= \frac{E(X)}{|S|} \leq \frac{2^{h(\epsilon)n}}{|S|} \Pr(X > 0) \\ &\leq 2^{h(\epsilon)n-s+1} \Pr(X > 0). \end{aligned} \tag{9.10}$$

Combining (9.9), (9.10),

$$\begin{aligned} \Pr(\exists x \in \text{SI } Ax = b) &= \Pr(X > 0) \\ &\geq 2^{s-h(\epsilon)n-m-2} \\ &\geq 2^{-(h(\epsilon)+2\epsilon)n-5} \quad \text{by our choice of } m. \end{aligned} \tag{9.11}$$

At this point, we have shown that with reasonable probability, we can concentrate the solutions into a ball of radius $< r$ without eliminating all of them. So suppose now that $\phi \wedge Ax = b$ is satisfiable and its solutions are in a ball of radius $< r$, and therefore diameter $d \leq 2\lceil \epsilon n \rceil$.

For assignments uv to disjoint sets of variables, let uv denote the union of these assignments. Let $uv, u\bar{v}$ be distinct solutions to $\phi \wedge Ax = b$ that are furthest apart, and let C be the set of variables on which they differ, so that $|C| \leq d$.

If we can guess a superset D of C and guess the partial assignment $\alpha_D = (uv)_D$, then we claim that uv is the unique solution to $\phi \wedge Ax = b \wedge x_D = \alpha_D$. To see this, suppose wv is another. Then $\text{dist}(wv, u\bar{v}) > \text{dist}(uv, u\bar{v})$, a contradiction. The probability of correctly guessing a set D of size d that contains C and correctly guessing α_D is

$$\geq \frac{1}{\binom{n}{d} 2^d} \geq 2^{-(h(2\epsilon)+2\epsilon)n}. \tag{9.12}$$

Combining (9.11), (9.12), we conclude that the probability p of correctly guessing s, A, b, D, α_D so that $\phi' \stackrel{\text{def}}{=} \phi \wedge Ax = b \wedge \alpha_D$ is uniquely satisfiable is $\geq \frac{1}{n} 2^{-(h(\epsilon)+h(2\epsilon)+4\epsilon)n-5}$. Now

$$\begin{aligned} h(\epsilon) + h(2\epsilon) + 4\epsilon &\leq \epsilon \lg \frac{4}{\epsilon} + 2\epsilon \lg \frac{2}{\epsilon} + 4\epsilon \quad \text{by theorem 2.2} \\ &= 2\epsilon \lg \frac{1}{\epsilon} + 8\epsilon \\ &\leq 4\epsilon \lg \frac{1}{\epsilon} \quad \text{since } \epsilon \leq 2^{-4}, \end{aligned}$$

so $p \geq \frac{1}{32n} 2^{-4\epsilon \lg \frac{1}{\epsilon}}$, as claimed. $\square$

We now use the isolation lemma to prove the 2 main results of this section.

Theorem 9.16. $s_k \leq \sigma_k + O(\frac{\ln^2 k}{k})$, and so $s_\infty = \sigma_\infty$.

Proof. Let $\epsilon = \frac{\ln k}{k}$. Then $k \geq \frac{1}{\epsilon} \ln \frac{1}{\epsilon}$. For each $k, \gamma > 0$, there is a randomized algorithm $U_{k,\gamma}$ for Unique- k -SAT with error $\leq \frac{1}{3}$ and exponential complexity $\leq \sigma_k + \gamma$. By the isolation lemma, there is a poly time randomized algorithm $I_{k,\epsilon}$ that can reduce a $\phi \in k$ -CNF to a uniquely

satisfiable $\phi' \in k\text{-CNF}$ with success probability $p \geq 2^{-O(\epsilon \lg \frac{1}{\epsilon})n}$. By running $U_{k,\gamma} \circ I_{k,\epsilon} O(p^{-1})$ times, we get a randomized algorithm for $k\text{-SAT}$ with error $\leq \frac{1}{3}$ and exponential complexity $\leq \sigma_k + \gamma + O(\epsilon \lg \frac{1}{\epsilon}) \leq \sigma_k + \gamma + O(\frac{\ln^2 k}{k})$. Since γ was arbitrary, $s_k \leq \sigma_k + O(\frac{\ln^2 k}{k})$. $\square$

It should be noted that theorem 9.16 is only an asymptotic result, we do not get anything useful for small k .

Theorem 9.17. *The following are all equivalent to the exponential time hypothesis (ETH):*

1. $s_3 > 0$
2. $\sigma_3 > 0$
3. $s_\infty > 0$
4. $\sigma_\infty > 0$.

Proof. $(\neg 1 \Rightarrow \neg 2) \sigma_3 \leq s_3 = 0$.

$(\neg 2 \Rightarrow \neg 3)$ It is sufficient to show that $\forall k \ s_k = 0$. We will use a simple clause-width reducing algorithm R_3 which takes as input a $k\text{-CNF}$ ϕ in n variables and m clauses and returns a 3-CNF ψ in $n + (k-3)m$ variables and $m(4(k-3) + 1)$ clauses such that $\text{sol}(\psi)$ has a trivial bijective correspondence with $\text{sol}(\phi)$. R_3 does the following: for each clause $\{l_1, \dots, l_k\}$ of ϕ , introduce $k-3$ new variables $y_2, \dots, y_{k-2}$ and add to ψ 3-clauses representing

$$\begin{aligned} (l_1 \vee l_2) &\leftrightarrow y_2 \\ (y_2 \vee l_3) &\leftrightarrow y_3 \\ &\vdots \\ (y_{k-3} \vee l_{k-2}) &\leftrightarrow y_{k-2} \\ (y_{k-2} \vee l_{k-1} \vee l_k). \end{aligned}$$

For each solution x of ϕ , there is exactly one solution y of ψ , and we can compute x from y simply by discarding the new variables.

For any set of parameters p and function f , let $O_p(f)$ denote the big-Oh class for f where the parameters in p are considered constant. The randomized sparsification theorem 5.17 showed that $\forall k, \epsilon > 0 \exists$ a $\text{poly}(n)$ time algorithm $S_{k,\epsilon}$ that, given an n -variable $k\text{-CNF}$ ϕ , returns an n -variable $k\text{-CNF}$ ψ with $\leq O_{k,\epsilon}(n)$ clauses such that $\text{sol}(\psi) \subseteq \text{sol}(\phi)$ and if ϕ is satisfiable, then so is ψ , with probability $\geq 2^{-\epsilon n}$.

We combine sparsification, isolation, and clause-width reduction. Suppose that $\forall \epsilon > 0 \exists$ a randomized algorithm $U_{3,\epsilon}$ solving Unique-3-SAT with error $\leq \frac{1}{3}$ with exponential complexity $\leq \epsilon$.

Let $\epsilon \in (0, \frac{1}{16}]$ and consider the action of the algorithm $T = R_3 \circ I_{k,\epsilon} \circ S_{k,\epsilon}$ on $\phi \in k\text{-CNF}$. If ϕ is not satisfiable, then neither is the output, but otherwise we trace the properties of formulas at intermediate steps in the following table.

formula	clause width	vars	clauses	sols	success prob
ϕ	k	n		≥ 1	1
$S_{k,\epsilon}(\phi)$	k	n	$O_{k,\epsilon}(n)$	≥ 1	$2^{-\epsilon n}$
$I_{k,\epsilon} \circ S_{k,\epsilon}(\phi)$	$O_{k,\epsilon}(1)$	n	$O_{k,\epsilon}(n)$	1	$2^{-O_k(h(\epsilon)n)}$
$T(\phi)$	3	$O_{k,\epsilon}(n)$	$O_{k,\epsilon}(n)$	1	$2^{-O_k(h(\epsilon)n)}$

So $U_{3,\epsilon} \circ T$ is a randomized algorithm for k -SAT with exponential complexity $\leq \epsilon$ and success probability $p \geq 2^{-O_k(h(\epsilon)n)}$. Running this $O(p^{-1})$ times decreases the error to $\leq \frac{1}{3}$ and increases the exponential complexity to $O_k(h(\epsilon))$, which converges to 0 as $\epsilon \rightarrow 0$. So $s_k = 0$.

$$(\neg 3 \Rightarrow \neg 4) \sigma_\infty \leq s_\infty = 0.$$

$$(\neg 4 \Rightarrow \neg 1) \sigma_3 \leq \sigma_\infty = 0 \Rightarrow s_3 \leq s_\infty = 0.$$

□

9.3 Limitations of oblivious isolation

Define the *unique k -SAT gap* as

$$\delta_k = c_{k\text{-SAT}} - c_{\text{Unique-}k\text{-SAT}}. \quad (9.13)$$

Of course $\delta_k \geq 0$. Our inability to prove $\delta_k = 0$ using some variant of the the Valiant-Vazirani reduction is not due to choosing too stringent a notion of 'good' hash family, nor a lack of imagination in constructing and representing one as a k -CNF, but is impossible simply because these techniques attempt to isolate a solution *obliviously* – they ignore the input formula F except for computing the number of variables n and passing F along to the oracle.

In this section, based on an idea of Russell Impagliazzo, we use Håstad's switching lemma to show that

$$\delta_k < \frac{1}{80k} c_{\text{Unique-}k\text{-SAT}} \quad (9.14)$$

cannot be shown using any oblivious technique. In theorem 9.16 ([CIKP08]), we showed that $\delta_k \leq O(\frac{\lg^2 k}{k})$. This was later improved by [Tra08], corollary 3, to

$$\delta_k \leq O\left(\frac{\lg k}{k}\right), \quad (9.15)$$

which is within a factor of $\lg k$ of optimal for oblivious techniques, which both results use.

(9.14) gives a method to quickly refute a claimed proof that $\delta_k = 0$. E.g. [Gho06] claims that $\delta_k = 0$, but only uses oblivious methods (and without separately disproving the exponential time hypothesis (ETH)), so this gives a quick way to verify that the proof is incorrect without going into the details. The subsequent revision still uses oblivious methods and so is also incorrect. But more importantly, (9.14) shows that we must focus future efforts on finding non-oblivious methods of isolating solutions.

Our proof proceeds as follows. Consider an oblivious technique that, on an input $F \in k$ -CNF with $V = \text{var}(F)$, chooses a formula $G \in k$ -CNF with $V \cup U = \text{var}(G)$, from some

distribution depending only on V , in the hopes that F restricted by G has a unique solution with noticeable probability. Let $n = |V|$ and suppose that the number of auxiliary variables $|U|$ is $\leq \alpha n$.

First a word of caution: in §9.1.1 it was shown that $O(n^2)$ auxiliary variables are sufficient to represent functions from a 2-universal hash family on n variables as a 3-CNF, which would allow oblivious isolation with probability $\geq \text{poly}(n)^{-1}$ via the Valiant-Vazirani reduction (theorem 9.2). So to show that (9.14) cannot be proven with oblivious techniques, we must pay attention to α .

To thwart the oblivious technique, let us give it as an input formula a random restriction F , that simply fixes a random subset of V . (A random restriction is easily encoded as a k -CNF.) Håstad's switching lemma implies that w.h.p. G restricted by F will have a representation by a low-width DNF and so cannot have a unique solution. Notice that previously we were thinking of G as restricting F , but now we are taking the opposite perspective, thinking of F as a restriction of G ; formally, both are just $F \wedge G$. How restrictive F needs to be, how high the probability is, and how wide the DNF is will be worked out in the following theorem.

Theorem 9.18. *Let $k \geq 2, \alpha = \frac{1}{80k}$, V be a set of variables, $n = |V|$, $\mathcal{G}$ be a distribution on k -CNF formulas on variables $V \cup U, |U| \leq \alpha n$. Then $\exists F \in k\text{-SAT}$ with $\text{var}(F) = V$ s.t.*

$$\Pr_{G \in \mathcal{G}}(|\text{sol}(F \wedge G)| = 1) \leq 2^{-\alpha n + 1}.$$

Proof. This will be fairly technical and depend on precise definitions, so let's review a few. For a formula F , $\text{var}(F)$ is the set of variables that actually appear in F , not an implicit superset of it. $\text{sol}(F) = \{a \in 2^{\text{var}(F)} \mid F(a) = 1\}$; thus, the unit clause $\{x\}$ that neglects to talk about some other variable y that we may have in mind is not deemed to have ≥ 2 solutions – it has 1. A clause is *proper* iff no 2 of its literals disagree. A CNF is a set of proper clauses. A k -CNF is a set of proper clauses each of size $\leq k$. For a CNF G and restriction $\rho : \text{var}(G) \rightarrow \{0, 1, *\}$, $G \upharpoonright \rho$ is the CNF obtained from G by eliminating true clauses and false literals. Thus $\text{var}(G \upharpoonright \rho) \subseteq \text{var}(G) \cap \rho^{-1}(*)$ and the subset may be proper. All this fussiness is necessary: to increase the relevance of the theorem, we want to give a Turing reduction from k -SAT to Unique- k -SAT the benefit of the doubt by having the loosest possible notion of the uniqueness promise and the tightest possible restrictions on the input F .

Let $G \in \mathcal{G}, p = \frac{1}{10k}$. Let $\rho \in \mathcal{D}'_{V \cup U, p}$ be a random p -restriction (defined in §2.6). Also, let $\rho_V = \rho|_V$. We represent ρ_V as $F \in 2\text{-CNF}$ with $\text{var}(F) = V$ by creating clauses $\{x = \rho_V(x), y\}, \{x = \rho_V(x), \bar{y}\}$ for each $x \in V - \rho_V^{-1}(*)$, $y \in \rho_V^{-1}(*)$. (We could represent ρ_V as a 1-CNF, but then it would not have n variables. We could fix that problem by adding extra variables in the form of tautologies such as $x \vee \bar{x}$, but then one could question whether the result holds when such degenerate clauses are forbidden. Notice that the result does *not* hold when $k = 1$ since a 1-CNF always has ≤ 1 solution, and so we could choose G to always be the empty 1-CNF.)

Let $k' = \beta pn$ for some $\beta \in (0, 1)$ to be chosen later. Then

$$\begin{aligned}
& Pr(|\text{sol}(F \wedge G)| = 1) \\
&= Pr(|\text{sol}(F \wedge G)| = 1 \wedge |\rho_V^{-1}(*)| < k') \\
&+ Pr(|\text{sol}(F \wedge G)| = 1 \wedge |\rho_V^{-1}(*)| \geq k').
\end{aligned} \tag{9.16}$$

$X = |\rho_V^{-1}(*)|$ is binomial with parameters (n, p) , so the first term on the RHS of (9.16) is

$$\begin{aligned}
&\leq Pr(X < \beta pn) \\
&\leq \left(\frac{e^{-(1-\beta)}}{\beta}\right)^{pn} \quad \text{from the Chernoff bound (theorem 2.7)} \\
&\leq 2^{-(1-\beta+\beta \lg \beta)pn} \\
&= 2^{-\beta pn} \quad \text{choosing } \beta = \frac{1}{4}.
\end{aligned} \tag{9.17}$$

To lower bound the second term on the RHS of (9.16) will be harder. Suppose that

$$\text{sol}(F \wedge G) = \{a\} \text{ and } a|U \text{ agrees with } \rho|U. \tag{9.18}$$

In other words, $F \wedge G$ has a unique solution a and $\forall x \in \text{dom}(a) \cap U - \rho^{-1}(*)$ $a(x) = \rho(x)$. We claim that there is a unique DNF representation H of $G \upharpoonright \rho$, and it has exactly 1 term T and $\text{var}(T) = \text{var}(G \upharpoonright \rho)$. To see this, note that H must contain at least 1 term since a satisfies it. For each term $T \in H$, a satisfies T . If $x \in \text{var}(G \upharpoonright \rho) - \text{var}(T)$, then $a, a \oplus x \in \text{sol}(F \wedge G)$, a contradiction. So $\text{var}(T) = \text{var}(G \upharpoonright \rho)$. But since a satisfies T , there can be only 1 such T .

So (9.18) implies $\text{DNF}(G \upharpoonright \rho) = |\text{var}(G \upharpoonright \rho)|$. Similarly, assuming (9.18), if $x \in \rho_V^{-1}(*) - \text{var}(G \upharpoonright \rho)$, then we have the contradiction $a, a \oplus x \in \text{sol}(F \wedge G)$ and $x \in \text{var}(F \wedge G)$, and so $\rho_V^{-1}(*) \subseteq \text{var}(G \upharpoonright \rho)$. So

$$(9.18) \Rightarrow \text{DNF}(G \upharpoonright \rho) \geq |\rho_V^{-1}(*)|. \tag{9.19}$$

Since $|U| \leq \alpha n$, we have $\forall V' \subseteq V \cup U, a \in 2^{V'}$,

$$Pr(a|U \text{ agrees with } \rho|U) \geq 2^{-\alpha n}. \tag{9.20}$$

So

$$\begin{aligned}
& Pr(|\text{sol}(F \wedge G)| = 1 \wedge |\rho_V^{-1}(*)| \geq k') 2^{-\alpha n} \\
&= \sum_a Pr(\text{sol}(F \wedge G) = \{a\} \wedge |\rho_V^{-1}(*)| \geq k') 2^{-\alpha n} \\
&\leq \sum_a Pr(\text{sol}(F \wedge G) = \{a\} \wedge |\rho_V^{-1}(*)| \geq k') \\
&\quad \cdot Pr(a|U \text{ agrees with } \rho|U) \quad \text{by (9.20)} \\
&= \sum_a Pr(\text{sol}(F \wedge G) = \{a\} \wedge |\rho_V^{-1}(*)| \geq k') \\
&\quad \wedge a|U \text{ agrees with } \rho|U) \quad \text{by independence} \\
&= Pr(|\text{sol}(F \wedge G)| = 1 \wedge |\rho_V^{-1}(*)| \geq k') \\
&\quad \wedge \text{unique solution of } (F \wedge G) \text{ agrees with } \rho|U) \\
&\leq Pr(\text{DNF}(G \upharpoonright \rho) \geq k') \quad \text{by (9.19)} \\
&\leq (5pk)^{k'} \quad \text{by theorems 2.13, 2.14, 2.17} \\
&= 2^{-\beta p n}.
\end{aligned} \tag{9.21}$$

Combining (9.16), (9.17), (9.21) gives

$$Pr(|\text{sol}(F \wedge G)| = 1) \leq 2^{-\beta p n} + 2^{-(\alpha - \beta p)n} \leq 2^{-(\alpha - \beta p)n + 1}.$$

Choosing $\alpha = \frac{1}{2}\beta p = \frac{1}{80k}$ shows

$$Pr(|\text{sol}(F \wedge G)| = 1) \leq 2^{-\alpha n + 1}.$$

F was chosen randomly, so this inequality must also hold for some fixed F . $\square$

Of course an oblivious randomized Turing reduction f from k -SAT to Unique- k -SAT with $\Omega(1)$ success probability could call the oracle on a formula with more than αn auxiliary variables, but then it cannot be guaranteed to take time $< 2^{c_{\text{Unique-}k\text{-SAT}}(1+\alpha)n}$ when a blackbox Unique- k -SAT solver is substituted for the oracle. So regardless of the number of auxiliary variables used, f cannot show $c_{k\text{-SAT}} < (1 + \alpha)c_{\text{Unique-}k\text{-SAT}}$.

This also shows that for each collection $\{H_{n,m} \mid m \in [2, n+1]\}$ of families of (not necessarily strongly) 2-universal hash functions mapping n bits to m bits, we cannot parsimoniously represent in k -CNF (an arbitrary equation $h(x) = b$ where h is drawn from one of the $H_{n,m}$) in time $2^{o(n)}$ and using $\leq \alpha' n$ auxiliary variables where $\alpha' < \alpha = \frac{1}{80k}$. This is a fairly powerful application of theorem 9.18. To prove a linear lower bound on the required number of auxiliary variables directly would be extremely difficult.

9.3.1 Conclusions

We actually showed slightly more than we set out for. It is not possible to obviously isolate the solutions to any family of formulas (of any kind, CNF or not) that contain arbitrary

restrictions if the isolators are k -CNFs using $\leq \alpha'n$ auxiliary variables where $\alpha' < \frac{1}{80k}$. E.g. it is not possible, with $\leq \alpha'n$ auxiliary variables and $2^{-o(n)}$ probability, to isolate the solutions to an arbitrary system of linear equations over $\mathbb{Z}_2$ by intersection with an obliviously chosen k -CNF.

9.4 Deciding unique satisfiability

So far we have only investigated k -SAT given the promise that there is ≤ 1 solution, and although we suspect that the unique satisfiability gap from (9.13) is 0, we had to settle for only bounding it by $O(\frac{\lg k}{k})$. Here we consider the problem of *deciding* whether a given k -CNF has exactly 1 solution, obtaining much stronger results.

In this section, we resolve an open question by [GB97]: the exponential complexity of deciding whether a k -CNF has a solution is the same as that of deciding whether it has exactly one solution, both when it is promised and when it is not promised that the input formula has a solution. We also show that this has the same exponential complexity as deciding whether a given variable is backbone (i.e. forced to a particular value), given the promise that there is a solution. We show similar results for True Quantified Boolean Formulas in k -CNF, k -Hitting Set (and therefore Vertex Cover), k -Hypergraph Independent Set (and therefore Independent Set), Max- k -SAT, Min- k -SAT, and 0-1 Integer Programming with inequalities and k -wide constraints. All these results will appear in [CP09].

Let us state the results more precisely. Recall that for a problem $L = \{x \mid \exists y R(x, y)\}$ where R is some relation, we define the *solutions* of x as $\text{sol}(x) = \{y \mid R(x, y)\}$. Define Decision-Unique- L (or DU- L) to be the problem of deciding whether the input x has $|\text{sol}(x)| = 1$; Unique- L (or U- L) to be DU- L but with the promise that $|\text{sol}(x)| \leq 1$; and Satisfiable-Unique- L (or SU- L) to be DU- L but with the promise that $|\text{sol}(x)| \geq 1$.

Note that these definitions depend on R being understood from context, since alternative formulations of a problem could lead to alternative definitions of DU- L , U- L , SU- L , but this will rarely cause ambiguity. E.g. DU- k -SAT will be the problem of deciding whether a given k -CNF has a unique satisfying assignment and DU-IS will be the problem of deciding whether a given graph has a unique independent set of size at most a given integer.

While $c_{k\text{-SAT}} \leq c_{\text{DU-}(k+1)\text{-SAT}}$ is obvious (add a new variable z to each clause and n new clauses ($z \rightarrow x_i$) for each i), the inequality is probably not tight since, assuming ETH, $c_{k\text{-SAT}}$ increases infinitely often as a function of k ([IP01], and here in theorem 10.4), and it would be surprising if it did not increase with each k . [GB97] conjectured that the deterministic exponential complexity of k -SAT and DU- k -SAT are the same, i.e. that

$$dc_{k\text{-SAT}} = dc_{\text{DU-}k\text{-SAT}}. \quad (9.22)$$

We prove (9.22) by giving a deterministic poly time Turing reduction from k -SAT to SU- k -SAT such that on an input with n variables, the oracle is called only on formulas with $\leq n + O(1)$

variables. Since $dc_{\text{SU-}k\text{-SAT}} \leq dc_{\text{DU-}k\text{-SAT}}$, and self-reducibility implies $dc_{\text{DU-}k\text{-SAT}} \leq dc_{k\text{-SAT}}$, we have

$$dc_{k\text{-SAT}} = dc_{\text{SU-}k\text{-SAT}} = dc_{\text{DU-}k\text{-SAT}} .$$

Via standard error reduction techniques (lemma 9.19), such a reduction also shows

$$c_{k\text{-SAT}} = c_{\text{SU-}k\text{-SAT}} = c_{\text{DU-}k\text{-SAT}} . \quad (9.23)$$

We use the same technique to show similar results for a variety of other problems.

It is not known whether $c_{k\text{-SAT}} = dc_{k\text{-SAT}}$, but it seems like the answer is 'no'. E.g. [IT04] gives a nice table contrasting the history of the development of deterministic vs. randomized k -SAT algorithms, with the more recent randomized algorithms with significantly less exponential complexity than their deterministic counterparts. (The table in §6.7 corroborates this.)

The main contribution here is the following technique for efficiently reducing a problem P to $\text{SU-}P$: view an instance ϕ of P as a set of constraints $C_1, \dots, C_m$ over a set of variables $x_1, \dots, x_n$. Starting with the empty set of constraints, add one constraint of ϕ at a time to a list and maintain the invariant that we know a solution to the conjunction of the constraints in the list. To add a new constraint C_i to the list, construct some gadget that conditionally turns on C_i in the case that the variables are assigned differently than the solution already known for the $i - 1$ case. Then, using this gadget, call the oracle repeatedly to learn a solution for the i case. Knowing a solution to the $i - 1$ case allows us to encode such gadgets significantly more efficiently than we would otherwise know how to do. Although this idea is the basis for each reduction we consider, there are significant problem-specific details that prevent us from factoring out large common parts of the proofs.

9.4.1 Previous work

[GB97] show that $dc_{k(r)\text{-SAT}} \leq dc_{\text{DU-}k\text{-SAT}}$ where $k(r)$ -SAT is k -SAT but where the input is restricted to have $\leq r$ false clauses at one or both of the assignments $1^n, 0^n$. Since there is nothing special about these 2 assignments, we might as well think of this variant as requiring that the input include some assignment at which the formula has $\leq r$ false clauses - i.e. that not only does the formula have a small satisfiability gap, but that the input include a witness to such a small gap.

But it's not immediately clear whether $k(r)$ -SAT is a very important restriction of SAT. On the one hand, the standard Cook reduction from an arbitrary problem in NP to SAT via computation tableaux generates a k -CNF with a gap of 1, and one can even generate a witness for this gap in poly time. This is because the formula generated essentially encodes, "There is a y such that after computing the predicate $R(x, y)$, the result is true," and only a single

clause actually encodes the part that says “the result is true”. So $k(1)$ -SAT is NP-complete, but the number of variables needed in a reduction to $k(r)$ -SAT seems to be large, even when reducing from k -SAT, and so it doesn’t seem to be very useful for upper bounding the exponential complexity of k -SAT.

On the other hand, the expected fraction of clauses satisfied in a k -CNF under a random assignment is $1 - 2^{-k}$, and so an assignment satisfying at least that many clauses can be found in poly time with high probability. But Håstad showed, using a proof based on PCPs [Hås01], that with no extra restriction on the input formula, no larger a fraction can be guaranteed, unless $P = NP$. This leaves unclear just how much smaller is the exponential complexity of $k(r)$ -SAT than that of k -SAT. We will show that they are equal.

9.4.2 Efficient reductions

The following notion will simplify the theorems in this section considerably. Let us say that a parameterized problem A *efficiently reduces* to parameterized problem B , and write $A \preceq B$, iff $\exists$ a poly time Turing reduction f from A to B so that for each instance x of A of parameter n and oracle call y that $f^B(x)$ makes, the parameter of y is $\leq n + O(1)$. Obviously, $\preceq$ is reflexive and transitive. We will also write $A \simeq B$ iff $A \preceq B$ and $B \preceq A$.

Lemma 9.19. *If $A \preceq B$ then $dc_A \leq dc_B$ and $c_A \leq c_B$.*

Proof. The first inequality is obvious. For the second, let M_B be a randomized $\text{poly}(|x|)2^{cn}$ -time algorithm solving B with error $\leq \frac{1}{16} = p$ (this can be constructed by one with error $\leq \frac{1}{3}$ by taking the majority answer from 21 independent calls). Suppose $f^B(x)$ runs in time $\leq t$. Define M_A as f , but replacing each call to the oracle by $r = 2 \lg t$ calls to M_B and take the majority answer. From the union bound, the probability that M_A errs is $\leq t$ times the probability that a binomial random variable with parameters (r, p) is $\geq \frac{1}{2}r$, and this is $\leq \sum_{i=\lceil \frac{r}{2} \rceil}^r \binom{r}{i} p^i (1-p)^{r-i} \leq p^{\frac{r}{2}} 2^r = 2^{-r}$. So M_A solves A , takes time $\leq \text{poly}(|x|)2t \lg t 2^{c(n+O(1))}$, and errs with probability $\leq t2^{-r} = \frac{1}{t}$. $\square$

Note that lemma 9.19 would hold even if we significantly loosened the notion of an efficient reduction by allowing subexponential time, randomness with one-sided error, and oracle calls to problems with parameter as much as $n(1 + o(1))$, but we will actually be demonstrating this stronger notion here. Also it should be noted that this form of reduction is more strict than the similar SERF reductions of [IPZ01], which allow a linear increase in the complexity parameter since they want the loosest possible notion of reduction under which SUBEXP is closed.

9.4.3 Constraint satisfaction problems

k -SAT

Theorem 9.20.

$$k\text{-SAT} \simeq \text{SU-}k\text{-SAT} \simeq \text{DU-}k\text{-SAT} .$$

(9.22) and (9.23) then follow from lemma 9.19. Note that all problems discussed in this section with a parameter k are solvable in poly time for $k < 2$, so we will assume throughout that $k \geq 2$. For each problem P , $\text{SU-}P \preceq \text{DU-}P$ via the identity map, so we won't bother to state this in the proofs. Also, for each problem P that we consider in this section, $\text{DU-}P \preceq P$ by using self-reducibility to find a solution and then making n more queries to decide whether there is another, so we won't bother to formally state this in the proofs to follow either.

For example, if $P = k\text{-SAT}$ and the input formula is ϕ , we can set a variable x_i , then ask the oracle whether the formula is still satisfiable to discover a correct setting of x_i , then set x_i correctly and continue similarly with the remaining variables to get a complete solution a . Then for each variable x_i , ask the oracle whether $F|_{x_i \neq a_i}$ is satisfiable.

Proof. ($k\text{-SAT} \preceq \text{SU-}k\text{-SAT}$) Let A be an oracle for $\text{SU-}k\text{-SAT}$. Also, let any predicate on $\leq k$ Boolean variables represent the k -clauses logically equivalent to them; e.g. $(x \rightarrow y = 0)$ will stand for the clause $\{\bar{x}, \bar{y}\}$.

Let $\phi = \{C_1, \dots, C_m\}$ with variables $x_1, \dots, x_n$ be our input formula and let z be a new variable. For i going from 1 to m , we will find a solution a to the first i clauses $\phi_i = \{C_1, \dots, C_i\}$, if there is one. Finding an a that satisfies 0 clauses is trivial. Suppose that we have an a that satisfies ϕ_{i-1} . For each literal l in C_i , we ask A whether $\phi_{i-1} \cup \{(z \rightarrow x_j = a_j) \mid j \in [n]\} \cup \{(\bar{z} \rightarrow l)\}$ (which is satisfiable: set $x = a, z = 1$) is uniquely satisfiable. It answers yes iff there is no solution to ϕ_i with $l = 1$. If each of the $|C_i| \leq k$ queries gives an answer of yes, then ϕ_i , and hence ϕ , is not satisfiable. If the j th query answers no, then we can safely set the first $j - 1$ literals of C_i to 0 and the j th literal to 1 in a partial assignment b . At this point, we know that b can be extended to a solution to ϕ_i , and we want to use similar calls to the oracle to find assignments to the remaining variables to get a new assignment that satisfies ϕ_i .

More specifically, suppose we have a partial assignment b' that we know can be extended to a solution to ϕ_i . Let b be a partial assignment that extends b' by setting a new variable x_r to an arbitrary value b_r . Then we use A and the following lemma to discover whether b can also be extended to a solution of ϕ_i , and if not, we simply flip b_r . We continue in this way until we have a full solution to ϕ_i .

Lemma 9.21. *Let $a \in \text{sol}(\phi_{i-1})$, b be a partial assignment, and $\psi = \phi_{i-1} \cup \{(z \rightarrow x_r = a_r) \mid r \in [n]\} \cup \{(\bar{z} \rightarrow x_r = b_r) \mid r \in \text{domain of } b\}$. Then ψ has a solution, and it is unique iff b cannot be extended to a solution to ϕ_{i-1} .*

Proof. a together with assigning z to 1 is a solution to ψ . There is no other solution to ψ with $z = 1$, and $\psi|_{z=0}$ forces the partial assignment b . $\square$

The reduction uses $\text{poly}(n)$ time and makes $\leq mn$ oracle calls, each with $\leq n + 1$ variables, so it is efficient. $\square$

Integer programming

Let k -BIP be the problem of deciding whether there is a solution to a given system of m linear inequalities in n Boolean variables, where each inequality involves $\leq k$ variables.

Corollary 9.22.

$$k\text{-BIP} \simeq \text{SU-}k\text{-BIP} \simeq \text{DU-}k\text{-BIP} .$$

Proof. Since the construction is almost exactly the same as for k -SAT in theorem 9.20, we only give an outline. To express $\alpha \rightarrow \beta$ in the construction, use the inequality $\alpha \leq \beta$. To express a negated variable $\bar{x}$, use $1 - x$. When adding a new constraint C_i , ask the oracle $\leq 2^k$ questions, one for each setting c of the $\leq k$ variables in C_i that satisfies C_i : is it possible to extend $b \cup c$ to a solution of the first $i - 1$ constraints? This gives a poly time reduction that makes $\leq m(2^k - k + n)$ queries, each with $\leq n + 1$ variables. $\square$

Backbone variables

Backbone variables are a tool from statistical physics for understanding the nature of random k -SAT instances - see e.g. [MZK⁺99]. Dubois and Dequen [DD01] use such variables in a heuristic to refute large unsatisfiable hard random 3-SAT instances.

Given a nonempty set system $S \subseteq \mathcal{P}(U)$, $i \in U$ is a *backbone variable* iff $\forall a \in S \ i \in a \vee \forall a \in S \ i \notin a$. x_i is a backbone variable of formula ϕ iff x_i is a backbone variable of $\text{sol}(\phi)$. Define the k *backbone variable* promise problem (k -BB) to be to decide whether a given variable is backbone in a given k -CNF ϕ with the promise that ϕ is satisfiable.

Corollary 9.23.

$$k\text{-SAT} \simeq k\text{-BB}.$$

Proof. Immediate from theorem 9.20 since a satisfiable k -CNF has exactly 1 solution iff each of its variables is backbone. $\square$

9.4.4 Extending the result up the PH

Define (d, k) -TQBF to be those true quantified Boolean formulas of the form $Q_1 \vec{w}_1 \cdots Q_d \vec{w}_d \phi$ where each $\vec{w}_i$ is a (possibly empty) tuple of Boolean variables, each quantifier

$Q_i \in \{\exists, \forall\}$ is $\exists$ iff i is odd, each variable of $\phi \in k$ -CNF is quantified, and the whole formula is true in the standard sense. The solutions are those assignments to $\vec{w}_1$ that make $Q_2 \vec{w}_2 \cdots Q_d \vec{w}_d \phi$ true. Then DU- (d, k) -TQBF is essentially those (d, k) -TQBFs where the first quantifier is $\exists$! (“there is a unique”) instead of $\exists$. If we parameterize on the total number of variables n , then we have the following.

Theorem 9.24.

$$(d, k)\text{-TQBF} \simeq \text{SU-}(d, k)\text{-TQBF} \simeq \text{DU-}(d, k)\text{-TQBF}.$$

Proof. $((d, k)\text{-TQBF} \preceq \text{SU-}(d, k)\text{-TQBF})$ It was shown in [APT79] that TQBF restricted to quantified 2-CNF formulas is in linear time, so we may assume wlog that $k \geq 3$. Let A be an oracle for SU- (d, k) -TQBF and $F = Q_1 \vec{w}_1 \cdots Q_d \vec{w}_d \phi$ with variables $x_1, \dots, x_n$ and clauses $\phi = \{C_1, \dots, C_m\}$ be our input formula. Let $x_1, \dots, x_q$ be the variables in $\vec{w}_1$, and y, z be variables not in F . If some clause has only $\forall$ quantified literals, then player $\forall$ can easily win and we reject. Otherwise, for i going from 1 to m , we will find a solution a , if there is one, to $F_i = Q_1 \vec{w}_1 \cdots Q_d \vec{w}_d \phi_i$ where $\phi_i = \{C_1, \dots, C_i\}$. Finding an a that satisfies F_0 is trivial. Suppose we have an a that satisfies F_{i-1} . We use A and the following lemma (with $b = \emptyset$) to decide whether F_i is satisfiable.

Lemma 9.25. *Let $a \in \text{sol}(F_{i-1})$, b be a partial assignment to $\vec{w}_1$. Suppose $l_k \in C_i$ is $\exists$ quantified under Q_j . Let $\psi = \phi_{i-1} \cup \{(z \rightarrow x_r = a_r) \mid r \in [q]\} \cup \{(\bar{z} \rightarrow x_r = b_r) \mid r \in \text{domain of } b\} \cup \{(z \rightarrow y), \{l_1, \dots, l_{k-1}, y\}, \{\bar{y}, l_k, z\}\}$, and $G = Q_1 z, \vec{w}_1 \cdots Q_j y, \vec{w}_j \cdots Q_d \vec{w}_d \psi$. Then G has a solution, and it is unique iff b cannot be extended to a solution to F_i .*

Proof. a together with assigning z (and y , if $j = 1$) to 1 is a solution to G . There is no other solution to G with $z = 1$, and $G|_{z=0}$ forces the partial assignment b . A winning strategy for player $\exists$ for $G|_{z=0}$ is also a winning strategy for F_i : just ignore y . Conversely, a winning strategy for F_i is a winning strategy for $G|_{z=0}$ if, in addition, we set $y = l_k$, which is possible since they are both quantified under Q_j . $\square$

If F_i is satisfiable, we find a solution b to F_i by starting with the empty partial assignment. Suppose we have a partial assignment b' that we know can be extended to a solution to F_i . Let b be a partial assignment that extends b' by setting a new variable x_r in $\vec{w}_1$ to an arbitrary value b_r . We use A and lemma 9.25 to discover whether b can be extended to a solution to F_i , and if not, we flip b_r . We continue in this way until we have a full solution to F_i . The reduction uses $\text{poly}(n)$ time and makes $\leq m(q + 1)$ oracle calls, each with $\leq n + 2$ variables, so it is efficient. $\square$

The same result also holds for (∞, k) -TQBF, i.e. without restricting the number of levels of alternation, but it is less exciting since it is obvious: just add 2 empty quantifiers at the beginning.

One could also ask whether the exponential complexity changes when !s are placed on some subset of existential quantifiers other than just the first one. But the above proof technique does not seem to generalize. One problem that arises when trying to prove that the exponential complexity of the unique case (where the ! is on a $\exists$ other than the first) is no more than that of the non-unique case is that, while before it was easy to store a solution (the a variable) in a polynomial amount of space, given a formula such as $\exists x \forall y \exists z \phi$, it now seems like we have to store a whole function that maps each y to an appropriate z . It is not obvious how to store such a function in a subexponential amount of space. But another problem arises when trying to prove even the reverse inequality: it was easy to show that a problem of the form $\exists!x \phi$ Turing reduces to problems of the form $\exists x \phi$ with only $o(n)$ more variables and in a subexponential amount of time, but how can one even Turing reduce a problem of the form $\forall x \exists!y \phi$ to problems of the form $\forall x \exists y \phi$ with only $o(n)$ more variables and in a subexponential amount of time, let alone other, more complex formulas with more quantifiers?

9.4.5 Solution optimization problems

The following problems involve optimizing the size of the solution.

k-hitting set

Define the *k-Hitting Set* problem (*k*-HS) as given a *k*-set system and an integer l , decide whether there is a hitting set of size $\leq l$; i.e. given (U, S, l) such that $S \subseteq \mathcal{P}(U)$ and $\forall s \in S \ |s| \leq k$, if we define the hitting sets as $\{h \subseteq U \mid \forall s \in S \ h \cap s \neq \emptyset\}$, decide whether there is a hitting set of size $\leq l$.

Theorem 9.26.

$$k\text{-HS} \simeq \text{SU-}k\text{-HS} \simeq \text{DU-}k\text{-HS} .$$

In particular, taking $k = 2$ gives the result for vertex cover:

$$\text{VC} \simeq \text{SU-VC} \simeq \text{DU-VC} .$$

Proof. ($k\text{-HS} \preceq \text{SU-}k\text{-HS}$) Let A be an oracle for $\text{SU-}k\text{-HS}$ and let (U, S, l) with nodes $U = \{x_1, \dots, x_n\}$ and sets $S = \{C_1, \dots, C_m\}$ be our input instance. Let $z, \bar{z}$ be 2 nodes not in U . For i going from 1 to m , we will find a smallest hitting set a of $S_i = \{C_1, \dots, C_i\}$. Initially, $a = \emptyset$ is a smallest hitting set of S_0 . Suppose that we have a smallest hitting set a of S_{i-1} .

Let $L = (l_1, \dots, l_n)$ be an ordering of U with the nodes of C_i first. Let sets b, c be initially empty. For j going from 1 to n , we will maintain the invariant that at step j , if S_i has a hitting set of size $\leq |a|$, then the lexicographically largest such (according to the order L , where l_1 is the most significant) contains all of b and none of c , and $b \cup c = \{l_1, \dots, l_j\}$. Suppose the

invariant holds for $j - 1$. We use A and the following lemma (with $b' = b \cup \{l_j\}$) to discover whether $b \cup \{l_j\}$ is contained in a hitting set of S_i of size $\leq |a|$.

Lemma 9.27. *Let a be a smallest hitting set of S_{i-1} , $b' \subseteq U$, and $T = S_{i-1} \cup \{\{z, \bar{z}\}\} \cup \{\{\bar{z}, x\} \mid x \in a\} \cup \{\{z, x\} \mid x \in b'\}$. Then T has a hitting set of size $\leq |a| + 1$, and it is unique iff S_i does not have a hitting set of size $\leq |a|$ that contains b' .*

Proof. $a \cup \{z\}$ is a hitting set of T of size $\leq |a| + 1$. There is no other containing z , and any without z contains b' . $\square$

If the answer is yes (which corresponds to an oracle answer of no), we add l_j to b , otherwise to c . If c ever contains all of C_i then S_i has no hitting set of size $\leq |a|$, in which case, letting l be an arbitrary element of C_i , $a \cup \{l\}$ is a smallest hitting set of S_i . Otherwise, we continue applying the lemma, adding elements to either b or c , until we have a hitting set of S_i of size $\leq |a|$. Once we have a smallest hitting set a for S_m , we simply compare $|a|$ to l . The reduction uses $\text{poly}(n)$ time and makes $\leq mn$ oracle calls, each with $\leq n + 2$ nodes, so it is efficient. $\square$

k -hypergraph independent set

The *k -Hypergraph Independent Set* problem (k -HIS) is, given a k -hypergraph (i.e. where each edge contains $\leq k$ vertices) and an integer l , decide whether there is a set I of vertices of size $\geq l$ such that no edge is contained in I .

Corollary 9.28.

$$k\text{-HIS} \simeq \text{SU-}k\text{-HIS} \simeq \text{DU-}k\text{-HIS} .$$

In particular, taking $k = 2$ gives the result for independent set:

$$\text{IS} \simeq \text{SU-IS} \simeq \text{DU-IS} .$$

Proof. Follows immediately from theorem 9.26 by observing that a set of vertices is independent and of size $\geq l$ iff its complement is a hitting set of the edges and of size $\leq n - l$. $\square$

Limitations

Although Hitting-Set (HS) and Set-Cover (SC) are duals of one another, it is not as obvious how the exponential complexities of k -HS and k -SC are related. The technique used to show theorem 9.26 does not seem to work for SC since constraints are represented by the universe elements and not the sets, so adding linearly many constraints to construct the oracle queries increases the parameter from n to cn for some $c > 1$, causing the reduction to be inefficient.

The situation for k -Coloring is similar.¹ Though we can easily construct an oracle query with the right logical properties, using only the techniques here will cause it to have linearly many more vertices, and thus only show that $c_{k\text{-Coloring}} \leq O(c_{\text{DU-}k\text{-Coloring}})$, where the constant factor in the big-Oh does not depend on k .

9.4.6 Constraint optimization problems

The following problems involve optimizing the number of constraints that are satisfied.

Max- k -SAT (Min- k -SAT) is the problem of deciding whether a given k -CNF with an integer weight for each clause has an assignment that satisfies at least (at most) some given weight l .

Theorem 9.29.

$$\text{Max-}k\text{-SAT} \simeq \text{SU-Max-}k\text{-SAT} \simeq \text{DU-Max-}k\text{-SAT} .$$

Proof. (Max- k -SAT $\preceq$ SU-Max- k -SAT) Let A be an oracle for SU-Max- k -SAT and $\phi \in k\text{-CNF}$, $l \in \mathbb{Z}$ be our input where ϕ has variables $x_1, \dots, x_n$ and clauses $C_1, \dots, C_m$ with (wlog) nonzero weights $w_1, \dots, w_m$. Let z be a variable not in ϕ . For i going from 1 to m , we will find the maximum weight l' that can be satisfied in $\phi_i = \{C_1, \dots, C_m\}$ and an assignment a that satisfies that weight. Initially, $l' = 0$ and a is arbitrary. Suppose we know the maximum weight l' that can be satisfied in ϕ_{i-1} and a satisfies it. We want to find the maximum weight l'' that can be satisfied in ϕ_i . We consider 2 cases: $w_i > 0$ and $w_i < 0$.

If $w_i > 0$, then $l'' \in [l', l' + w_i]$. For each literal $l \in C_i$ and $t' \in (l', l' + w_i]$, we can use A and the following lemma (with b the partial assignment that sets $l = 1$, and $t = t' - w_i$) to discover whether some assignment with $l = 1$ satisfies weight $\geq t'$ in ϕ_i .

Lemma 9.30. *Let l' be the largest weight that can be satisfied in ϕ_{i-1} and let a satisfy that weight. Let $|w_i| > 0$ and $l' - |w_i| < t \leq l'$. Let b be a partial assignment and $\psi = \phi_{i-1} \cup \{|w_i| \cdot (z \rightarrow x_r = a_r) \mid r \in [n]\} \cup \{|w_i| \cdot (\bar{z} \rightarrow x_r = b_r) \mid r \in \text{domain of } b\}$. Then ψ has an assignment that satisfies weight $\geq t + (n + |b|)|w_i|$, and it is unique iff b cannot be extended to an assignment that satisfies weight $\geq t$ in ϕ_{i-1} .*

Proof. a together with $z = 1$ satisfies weight $\geq l' + (n + |b|)|w_i| \geq t + (n + |b|)|w_i|$ in ψ . There is no other such assignment with $z = 1$, and any with $z = 0$ agrees with b . $\square$

So we can use binary search to discover l'' using $\leq |C_i| \lg(w_i + 1)$ queries. If $l'' = l'$, then a satisfies weight l'' in ϕ_i and we are done. Otherwise, as a slight optimization, if the last query set the j th literal of C_i to 1, then we can safely set literal j to 1 and literals 1 through $j - 1$ to 0 in the partial assignment b , and at this point, we know that b can be extended to a

¹Here we take 'uniqueness' of a coloring solution to mean 'unique up to permutations of the colors'.

full assignment satisfying weight l'' in ϕ_i and that b satisfies C_i (since $l'' > l'$). We can continue using the lemma to extend b to such an assignment using $n - j$ more queries to A .

If $w_i < 0$, then $l'' \in [l' + w_i, l']$. Let b be the partial assignment that makes C_i false. For each $t \in (l' + w_i, l']$ we can use 1 query to A and the lemma to discover whether there is an assignment satisfying weight $\geq t$ in ϕ_i . So we can use binary search to discover l'' using $\leq \lg(|w_i| + 1)$ queries. If $l'' = l' + w_i$, then a satisfies weight l'' . Otherwise, we can continue using the lemma to extend b to a full assignment satisfying weight l'' in ϕ_i using $n - |C_i|$ more queries.

At the end, we simply compare l' to l . The reduction uses time polynomial in the size of the input and makes $\leq \sum_i (|C_i| \lg(|w_i| + 1) + n - |C_i|)$ oracle calls, each with $\leq n + 1$ variables, so it is efficient. $\square$

Theorem 9.29 applies to integer weights, but the proof is robust and can easily be modified to accommodate rational weights, e.g. by first multiplying by the LCM of the denominators. Since the weights used in the proof have the same size as those of the input, the same proof works for the problem restricted to unit weights. The same proof also works for nonnegative weights, or any combination of the above.

Negating the input weights together with the algorithm in the proof also gives an efficient reduction from Min- k -SAT to SU-Min- k -SAT. The only rub is that this reduction is not correct if we restrict to nonnegative weights. To handle this case, we use a different reduction below.

Theorem 9.31. *If we restrict to nonnegative integer weights,*

$$\text{Min-}k\text{-SAT} \simeq \text{SU-Min-}k\text{-SAT} \simeq \text{DU-Min-}k\text{-SAT}.$$

Two clauses are in conflict with each other, i.e. cannot be simultaneously unsatisfied, iff they share a variable, but positively in the one and negatively in the other. We want to find a maximum weight independent set in the graph with the clauses as vertices and edges between conflicting clauses. But even if we restricted to unit weights, we cannot simply invoke corollary 9.28 since the parameter here is variables, not clauses. Thus it seems like we need another proof.

It should be pointed out that the proof below does not work if we restrict to unit weights since the construction of ψ uses non-unit weights. There may be some more convoluted construction using the same ideas but that avoids this technical point.

Proof. (Min- k -SAT $\preceq$ SU-Min- k -SAT) Let A be an oracle for SU-Min- k -SAT and $\phi \in k\text{-CNF}$, $l \in \mathbb{Z}$ be our input where ϕ has variables $x_1, \dots, x_n$ and clauses $C_1, \dots, C_m$ with (wlog) positive weights $w_1, \dots, w_m$. Let z be a variable not in ϕ . For i going from 1 to m , we will find the minimum weight l' that can be satisfied in $\phi_i = \{C_1, \dots, C_i\}$ and an assignment a that satisfies that weight. Initially, $l' = 0$ and a is arbitrary. Suppose we know the minimum weight l' that can be satisfied in ϕ_{i-1} and a satisfies it. The minimum weight l'' that can be satisfied in ϕ_i is in the interval $[l', l' + w_i]$. Letting b be the partial assignment that makes C_i false, we can use the following lemma and binary search to discover l'' using $\lg(w_i + 1)$ calls to A .

Lemma 9.32. *Let l' be the smallest weight that can be satisfied in ϕ_{i-1} and let a satisfy that weight. Let $w_i > 0$ and $l' \leq t < l' + w_i$. Let b be a partial assignment and $\psi = \phi_{i-1} \cup \{w_i \cdot (z \rightarrow x_r \neq a_r) \mid r \in [n]\} \cup \{w_i \cdot (\bar{z} \rightarrow x_r \neq b_r) \mid r \in \text{domain of } b\} \cup \{(n - |b|)w_i \cdot (z)\}$. Then ψ has an assignment that satisfies weight $\leq t + nw_i$, and it is unique iff b cannot be extended to an assignment that satisfies weight $\leq t$ in ϕ_{i-1} .*

Proof. a together with $z = 1$ satisfies weight $\leq l' + nw_i \leq t + nw_i$ in ψ . There is no other such assignment with $z = 1$, and any with $z = 0$ agrees with b . $\square$

If $l'' = l' + w_i$, then a satisfies weight l'' in ϕ_i . Otherwise, we use the lemma (with $t = l''$) $n - |C_i|$ times to extend b to a full assignment achieving weight l'' in ϕ_i . At the end, we simply compare l' to l . The reduction uses time polynomial in the size of the input and makes $\leq \sum_i (\lg(w_i + 1) + n - |C_i|)$ oracle calls, each with $\leq n + 1$ variables, so it is efficient. $\square$

9.4.7 Conclusions

We've shown a simple technique to settle conjecture (9.22) as well as many questions relating the exponential complexity of similar parameterized constraint satisfaction problems to their unique-solution counterparts. The problem list here is not intended to be exhaustive but demonstrative. Theorem 9.29 shows the robustness of the technique, allowing considerable variation in problem specification without disturbing the proof.

Relating the exponential complexities of such problems to their unique-solution counterparts under the *promise* of at most 1 solution appears to be harder. Current techniques, such as the Valiant-Vazirani reduction and the Isolation Lemma 9.14 use oblivious hashing, which we showed in §9.3 necessarily fall short of such strong results as are here. It seems like new, non-oblivious techniques are needed to make further progress.

Acknowledgements

Material from §9.2 is joint work with Russell Impagliazzo, Ramamohan Paturi, and Valentine Kabanets and appears in [CIKP08]. That from §9.3 is unpublished joint work with Russell Impagliazzo. That from §9.4 is joint work with Ramamohan Paturi and appears in [CP09].

Chapter 10

Clause width, density duality for SAT

The performance of algorithms is often instance-sensitive, but how does the difficulty vary with the instance? For CNF-SAT, 3 common approaches to parametrically constrain a CNF to upper bound how hard it is are

- bound the maximum clause width to some k ,
- bound the ratio of clauses to variables to some Δ ,
- bound the maximum frequency with which variables appear to some f ,

and we denote the problems with these restricted inputs by, respectively, k -SAT, SAT_Δ , and f -Freq-SAT. All known upper bounds α on the exponential complexities of these problems for particular values of the parameters k, Δ, f converge to 1 as the parameter in question tends to ∞ . This raises the question of whether for any of the 3 restriction types there is an algorithm, or even a non-uniform sequence of algorithms, for which α converges to a number < 1 as the parameter tends to ∞ . For k -SAT, the negation of this we called the Strong Exponential Time Hypothesis (SETH, conjecture 6.10).

Though still not resolved, [CIP06] show, and we show in this chapter, that they are equivalent. Thus SETH can be equivalently defined in terms of any of these restrictions. In fact, [CIP06] show that the sequences of exponential complexities (α) for the 3 restriction types have an intuitive functional relationship and even converge to the same limit (theorems 10.13, 10.14). In particular, c_{SAT_Δ} is not must larger than $c_{k\text{-SAT}}$ when $k \leq O(\lg \Delta)$, and the reverse approximate inequality holds when $\lg \Delta \leq O(k \lg k)$.¹ We will make this more precise in §10.3.1.

¹It was slightly overstated in [CIP06] that $c_{\text{SAT}_\Delta} \approx c_{k\text{-SAT}}$ when $k = \Theta(\lg \Delta)$. This would actually require a further reduction in the sparsification constant.

The intuition for the proofs comes from considering pairs of restrictions. E.g. if we fix k and vary Δ , how does the complexity of $k\text{-SAT}_\Delta = k\text{-SAT} \cap \text{SAT}_\Delta$ vary? If Δ is very small, then the problem will be *underconstrained*, in that there will typically be a very large number of solutions, and we will be able to set many variables arbitrarily without making the problem unsatisfiable. Thus, in some intuitive sense, the problem really involves a much smaller number of variables than it has, and is thus not maximally difficult. On the other hand, an instance where Δ is very big is *overconstrained*, in that setting a few variables usually will force the values of many others. Thus, overconstrained problems are also not the most difficult.

For random formulas, this intuition can be substantiated. If F is chosen as a uniformly random set of $m = \Delta n$ clauses over n variables (et al distributions), it is known that there is a constant *satisfiability threshold* for Δ , which is $2^k \ln 2 - O(k)$ ([AM02, AP03]), below which $F \in \text{SAT}$ w.h.p. and above which $F \notin \text{SAT}$ w.h.p. It is not known whether this threshold is a single value or a small interval ([Fri99]). Poly time algorithms that almost certainly certify satisfiability of random instances with clause density substantially below the threshold are known ([FS96]), and survey propagation is a heuristic that seems to work well even for instances near but below the threshold. There is currently no proven efficient algorithm that can certify unsatisfiability just above the threshold, nor even is there a poly time algorithm when $k = 3$, Δ is above the threshold but $\ll \sqrt{n}$ [Fla09]. However, for resolution-based methods, random formulas have exponential complexity for constant densities above the threshold, but sub-exponential for every increasing density ([BKPS98]). Both experimental and theoretical studies illustrate how the expected difficulty of the problem peaks at the threshold, then falls gradually off ([KS94, ART06]).

The sparsification lemma shows that for any fixed k , there is a density Δ so that $k\text{-CNF}$ formulas of density Δ are almost the hardest instances of $k\text{-SAT}$. (Independently, [JS99] proved a similar result for the maximum independent set problem.) This supports the intuition that the hardest instances have a linear relationship between variables and constraints. However, in [IPZ01], this constant density was doubly exponential in k . Intuitively, however, the hardest density of $k\text{-SAT}$ should be exponential in k , as is the case for random instances. The improved bound from [CIP06], and here in theorem 5.15, brings this down to exponential in $O(k \lg k)$, which is much closer to the intuitive value.

As one application of the new techniques here, we will demonstrate in §10.4 the impossibility of sparsifying CNF formulas of size polynomial in the number of variables.

There seems to be a highly fruitful connection between new approaches to SAT algorithms and reductions between restricted versions of SAT. For example, [IP01], which showed that ETH implies s_k increases infinitely often as a function of k , is largely based on the satisfiability coding lemma of [PPZ99], which was originally used to analyze a new SAT algorithm. The results here are also largely based on a SAT algorithm: Schuler's CNF-SAT algorithm [Sch05]. Schuler's approach leverages the performance of $k\text{-SAT}$ algorithms such as PPZ to solve general SAT through a width reduction technique. We reformulate this technique as a reduction from

SAT_Δ to $k\text{-SAT}$, giving a considerably different presentation from that of [Sch05]. A benefit of this is that while [Sch05] presents as an open question whether there exists a deterministic algorithm solving CNF-SAT on n variables and m clauses with exponential complexity $\leq 1 - \frac{1}{\lg m}$, the presentation here will show clearly that the answer is yes.

10.1 Definitions

The width of F is $\text{width}(F) = \max\{|C| \mid C \in F\}$. The density of F is $\text{density}(F) = \frac{|F|}{|\text{var}(F)|}$. The frequency of F , $\text{freq}(F)$, is the maximum number of times a variable occurs in F . The average frequency of F , $\overline{\text{freq}}(F)$, is the number of times each variable occurs in F , averaged over all variables; i.e. the number of literal occurrences divided by $|\text{var}(F)|$.

$$k\text{-SAT} = \{F \in \text{CNF-SAT} \mid \text{width}(F) \leq k\}$$

$$\text{SAT}_\Delta = \{F \in \text{CNF-SAT} \mid \text{density}(F) \leq \Delta\}$$

$$f\text{-Freq-SAT} = \{F \in \text{CNF-SAT} \mid \text{freq}(F) \leq f\}$$

$$f\text{-}\overline{\text{Freq}}\text{-SAT} = \{F \in \text{CNF-SAT} \mid \overline{\text{freq}}(F) \leq f\}.$$

Then define

$$s_k = c_{k\text{-SAT}}$$

$$d_\Delta = c_{\text{SAT}_\Delta}$$

$$\varphi_f = c_{f\text{-Freq-SAT}}$$

$$\overline{\varphi}_f = c_{f\text{-}\overline{\text{Freq}}\text{-SAT}},$$

and a subscript of ∞ in any of these sequences will represent the limit; e.g. $s_\infty = \lim_{k \rightarrow \infty} s_k$. Occasionally, we will use the same notation when k, Δ are growing functions of the number of variables n ; e.g. $\text{SAT}_\Delta = \{F \in \text{SAT} \mid \text{density}(F) \leq \Delta(|\text{var}(F)|)\}$. Note that for any CNF F ,

$$\text{freq}(F) \leq \overline{\text{freq}}(F) \leq \text{density}(F). \quad (10.1)$$

So $\forall f > 0 \ \varphi_f \leq \overline{\varphi}_f \leq d_f$.

10.2 Previous work

Let $k\text{-SAT}_\Delta = k\text{-SAT} \cap \text{SAT}_\Delta$, and $s_{k,\Delta} = c_{k\text{-SAT}_\Delta}$. Then by composing the sparsification algorithm with any algorithm for $k\text{-SAT}_{O(k/\epsilon)^{3k}}$ we obtain from the new sparsification constants in theorem 5.15:

Lemma 10.1. $\forall k, \epsilon > 0 \ \exists f \leq O(k/\epsilon)^{3k} \ s_k \leq c_{k\text{-SAT} \cap f\text{-Freq-SAT}} + \epsilon$.

In particular, letting $\epsilon = k^{-c}$, we obtain:

Corollary 10.2. $\forall k, c > 0 \exists f \leq 2^{O(ck \lg k)} s_k \leq c_{k\text{-SAT} \cap f\text{-Freq-SAT}} + k^{-c}$.

Thus, roughly exponential density formulas are very close to the hardest k -CNF instances. (10.1) then implies the following.

Corollary 10.3. $\forall k, \epsilon > 0 \exists f > 0 s_k \leq \varphi_f + \epsilon \leq \overline{\varphi}_f + \epsilon \leq d_f + \epsilon$.

The following theorem, proven by demonstrating a deterministic Turing reduction using the sparsification lemma and ideas from the satisfiability coding lemma (theorem 6.2), upper bounds the rate of convergence of s_k .

Theorem 10.4 ([IP01]). $\text{ETH} \Rightarrow \exists d > 0 \forall k \geq 3 s_\infty - s_k \geq \frac{d}{k}$.

[Sch05] demonstrated a randomized algorithm solving CNF-SAT in n variables and m clauses, with no restriction on clause width, in time

$$\text{poly}(n, m) 2^{(1 - \frac{1}{1 + \lg m})n}. \quad (10.2)$$

This algorithm can be viewed as a Turing reduction from CNF-SAT with bounded clause density to CNF-SAT with bounded clause width, and we will now present and analyze it as such.

```

ReduceWidthk( $F \in \text{CNF}$ )
  if  $F$  has no clause of size  $> k$ 
    output  $F$ 
  else
    let  $C$  be a clause of  $F$  of size  $k' > k$ 
    and  $C'$  be the first  $k$  literals of  $C$ 
     $F_1 \leftarrow F - \{C\} \cup \{C'\}$ 
     $F_0 \leftarrow F|(C' = 0)$ 
    /*  $F_0$  is  $F$  but with the literals of  $C'$  set to false,
    and with true clauses and false literals removed */
    ReduceWidthk( $F_1$ ) /* left branch: guess  $C'$  true */
    ReduceWidthk( $F_0$ ) /* right branch: guess  $C'$  false */

```

Figure 10.1: width reduction algorithm

For $k \geq 3$, consider the routine in figure 10.1. The idea is that the algorithm reduces the width of an arbitrary long clause (size $> k$) C by branching on whether the disjunction C' of the first k literals of C is true or not. If it guesses so, then it replaces C by the short clause C' and the number of long clauses is reduced by 1. If it guesses not, then it simplifies the formula

by eliminating the k variables of C' . Clearly the solutions of the input formula is the union of the solutions of the output formulas.

Let p be some path of length t in the tree T of recursive calls of $\text{ReduceWidth}_k(F)$ and let G be the output at the leaf of p . Let l, r be the number of left, right branches in p . Then since each right branch eliminates k variables, we have $|\text{var}(G)| = n - kr$ and $r \leq \frac{n}{k}$. Since each left branch eliminates 1 clause of size $> k$, we have $l \leq m$.

So the number of paths in T with r right branches is $\leq \binom{m+r}{r}$, and each outputs a formula with $n - kr$ variables. Furthermore, each output formula is generated with $\text{poly}(n, m)$ delay.

Lemma 10.5. *If k -SAT can be solved in time $\text{poly}(n, m)2^{sn}$ by some algorithm S , where the polynomial does not depend on k, s, S , then CNF-SAT in n variables and $m \geq \frac{n}{k}$ clauses can be solved in time*

$$\text{poly}(n, m)2^{sn + \frac{4m}{2sk}},$$

the polynomial not depending on k, s, S . (If $m < \frac{n}{k}$, then we get $\text{poly}(n, m)2^{(s + \frac{4}{k2sk})n}$.)

Proof. Combining S with ReduceWidth_k gives an algorithm for CNF-SAT whose running time is $\leq \text{poly}(n, m)$ times

$$\begin{aligned} & \sum_{r=0}^{\lfloor \frac{n}{k} \rfloor} \binom{m+r}{r} 2^{s(n-kr)} \\ & \leq \sum_{r=0}^{m + \lfloor \frac{n}{k} \rfloor} \binom{m + \lfloor \frac{n}{k} \rfloor}{r} 2^{s(n-kr)} \\ & \leq 2^{sn} \left(1 + 2^{-sk}\right)^{m + \frac{n}{k}} \\ & \leq 2^{sn} e^{2^{-sk}(m + \frac{n}{k})} \\ & \leq 2^{sn + \frac{4 \max\{m, \frac{n}{k}\}}{2sk}}. \end{aligned}$$

□

PPZ is an algorithm solving k -SAT in time $\text{poly}(n, m)2^{(1-\frac{1}{k})n}$, where the polynomial does not depend on k , and so Schuler's result (10.2) can now be obtained from lemma 10.5 by taking $s = 1 - \frac{1}{k}, k = 1 + \lg m$. Our derandomization of LocalSearch in §6.4.1 solves k -SAT in time $\text{poly}(n, m)2^{(1-\lg(1+\frac{1}{k}))n + o(n)}$, where the polynomial does not depend on k . So using the derandomized LocalSearch, $s = 1 - \lg(1 + \frac{1}{k}), k = \lg m$, we get a *deterministic* algorithm for CNF-SAT with running time $\leq \text{poly}(n, m)2^{(1-\frac{1}{\lg m})n + o(n)}$, answering one of Schuler's open questions. (Note that for our derandomization of PPZ, the leading polynomial in the running time has an exponent that grows linearly with k , and so will not suffice for this result unless we strengthen the hypothesis by bounding m .) However, we are going to be more interested in using the above as a reduction than to directly design SAT algorithms.

10.3 Relating the SAT constants

We use the reductions given in the previous section to prove the main results of the chapter – that the exponential complexities of k -SAT, SAT_Δ , and f -Freq-SAT are intertwined, and hence their limiting values are identical.

10.3.1 Relating k -SAT to SAT_Δ

The proof of $d_\infty = s_\infty$ proceeds in 2 steps. First, sparsification (corollary 10.3) shows that $s_\infty \leq d_\infty$. To go the other way, we reduce SAT_Δ to k -SAT by using the width-reduction algorithm. The following lemma bounds the penalty paid for this reduction.

Lemma 10.6. $\forall \Delta > 0, k \geq \frac{1}{\Delta}, \epsilon > 0 \ d_\Delta \leq s_k + \epsilon + \frac{4\Delta}{2^{(s_k + \epsilon)k}}.$

Proof. Let S be an algorithm solving k -SAT in time $O(2^{(s_k + \epsilon)n})$. Given $F \in \text{CNF}$ with n variables and $m \leq \Delta n$ clauses, duplicate clauses in F until $m \in [\Delta n, \Delta n + 1)$. Now apply lemma 10.5. $\square$

Taking the limit as $\epsilon \rightarrow 0$ gives the following.

Corollary 10.7. $\forall \Delta > 0, k \geq \frac{1}{\Delta} \ d_\Delta \leq s_k + \frac{4\Delta}{2^{s_k k}}.$

By choosing $k = O(\lg \Delta)$ we can make the penalty $\frac{4\Delta}{2^{s_k k}}$ small. In particular, we will want this to be smaller than the distance between s_k and s_∞ , which we can lower bound by using theorem 10.4. The following technical lemma shows exactly how to choose k .

Lemma 10.8. $\text{ETH} \Rightarrow \forall \Delta > 0 \ \exists k \in O(\lg \Delta), \delta \in \Omega(\frac{1}{\lg \Delta}), l$

$$s_l - s_k \geq \frac{4\Delta}{2^{s_k k}} + \delta.$$

Proof. Let d be as in theorem 10.4. Choose $k = \frac{2}{s_3} \lg(\frac{2}{s_3} \frac{16\Delta}{d}) \in O(\lg \Delta)$, $\delta = \frac{d}{4k} \in \Omega(\frac{1}{\lg \Delta})$, l so large that $s_\infty - s_l \leq \frac{d}{2k}$. Then

$$\begin{aligned} \frac{d}{16\Delta} 2^{s_k k} &\geq \frac{d}{16\Delta} 2^{s_3 k} = \left(\frac{2}{s_3}\right)^2 \frac{16\Delta}{d} \\ &\geq \frac{2}{s_3} \lg\left(\frac{2}{s_3} \frac{16\Delta}{d}\right) = k. \end{aligned} \tag{10.3}$$

So

$$\begin{aligned} s_l - s_k &= (s_\infty - s_k) - (s_\infty - s_l) \\ &\geq \frac{d}{k} - \frac{d}{2k} \\ &= \frac{d}{4k} + \frac{d}{4k} \\ &\geq \frac{4\Delta}{2^{s_k k}} + \delta \quad \text{by (10.3).} \end{aligned}$$

$\square$

Now we can upper bound the complexity of SAT_Δ by that of $k\text{-SAT}$ for some k (and vice versa) and even add an exponential gap.

Corollary 10.9. $\text{ETH} \Rightarrow \forall \Delta > 0 \exists l \ d_\Delta \leq s_l - \Omega(\frac{1}{\lg \Delta})$.

Proof. Combine corollary 10.7 and lemma 10.8. $\square$

Corollary 10.10. $\text{ETH} \Rightarrow \forall k \geq 3 \exists \Delta > 0 \ s_k \leq d_\Delta - \Omega(\frac{1}{k})$.

Proof. From corollary 10.3, $s_\infty \leq d_\infty$. Let d be as in theorem 10.4. Choose Δ so large that $d_\infty - d_\Delta \leq \frac{d}{2k}$. Then

$$\begin{aligned} s_k &\leq s_\infty - \frac{d}{k} \leq (d_\infty - d_\Delta) + (d_\Delta - \frac{d}{k}) \\ &\leq d_\Delta - \frac{d}{2k}. \end{aligned}$$

$\square$

We are now in a position to prove the analog of theorem 10.4.

Theorem 10.11. $\text{ETH} \Rightarrow \forall \Delta > 0 \ d_\infty - d_\Delta \geq \Omega(\frac{1}{\lg \Delta})$.

Proof. Combine corollaries 10.9, 10.10. $\square$

We need the following to eliminate the assumption of ETH.

Lemma 10.12. $\neg \text{ETH} \Rightarrow \forall \Delta > 0, k \ d_\Delta = s_k = 0$.

Proof. Use theorem 9.17 and lemma 10.6 to conclude that $\forall \Delta > 0, k \geq \frac{1}{\Delta}, \epsilon > 0 \ d_\Delta \leq \epsilon + \frac{4\Delta}{2\epsilon k}$. Choose $k = \frac{1}{\epsilon} \lg \frac{4\Delta}{\epsilon}$ to get that $\forall \epsilon > 0 \ d_\Delta \leq 2\epsilon$, which implies $d_\Delta = 0$. $\square$

Theorem 10.13. $d_\infty = s_\infty$.

Proof. Combine corollaries 10.9, 10.10 and lemma 10.12. $\square$

From theorem 9.16, $s_\infty = \sigma_\infty$, where $\sigma_k = c_{\text{Unique-}k\text{-SAT}}$. So now we have $d_\infty = s_\infty = \sigma_\infty$.

Of course the above theorem did not require the gaps in corollaries 10.9, 10.10 that we went through so much trouble to lower bound. §10.4 will make use of these gaps.

10.3.2 Relating $k\text{-SAT}$ to $f\text{-Freq-SAT}$

Analogous to theorem 10.13, we can easily show that $\bar{\varphi}_\infty = \varphi_\infty = d_\infty = s_\infty = \sigma_\infty$.

Theorem 10.14. $d_\infty = \bar{\varphi}_\infty = \varphi_\infty$.

Proof. Recall that $\forall f > 0 \ \varphi_f \leq \bar{\varphi}_f \leq d_f$. So $\varphi_\infty \leq \bar{\varphi}_\infty \leq d_\infty$. If $\neg \text{ETH}$, then by lemma 10.12, $\varphi_\infty \leq \bar{\varphi}_\infty \leq d_\infty = 0$. If ETH holds, then corollaries 10.9 and 10.3 show that $\forall \Delta > 0 \exists k, f, \epsilon > 0 \ d_\Delta \leq s_k - \epsilon \leq \varphi_f - \frac{\epsilon}{2} \leq \bar{\varphi}_f - \frac{\epsilon}{2}$, and so $d_\infty \leq \varphi_\infty \leq \bar{\varphi}_\infty$. $\square$

Combining theorems 10.14, 10.11 gives the following.

Theorem 10.15. $\text{ETH} \Rightarrow \forall f \geq 2 \ \varphi_\infty - \varphi_f \geq \Omega(\frac{1}{\lg f})$.

10.3.3 Putting the bounds together

Theorem 10.16. $\overline{\varphi}_f$ is a continuous function of f on the domain $[1, \infty)$.

Proof. It is easy to see that $\overline{\varphi}_1 = \lim_{f \rightarrow 1+} \overline{\varphi}_f = 0$. Since $\overline{\varphi}_f$ is monotonically nondecreasing in f , it is sufficient to show that

$$\forall f > 1, \epsilon > 0 \ \exists \delta > 0 \ \overline{\varphi}_{f+\delta} - \overline{\varphi}_{f-\delta} \leq \epsilon. \quad (10.4)$$

Let $f > 1, \epsilon > 0$ and choose $\delta = \frac{\epsilon(f-1)}{2}$. Let $F \in \text{CNF}, n = |\text{var}(F)|, \overline{\text{freq}}(F) \leq f + \delta$. Let S be the set of the ϵn variables of F of highest frequency. By branching on S , we obtain $\leq 2^{\epsilon n}$ formulas F_i . For each variable x that is removed by the branching, we can replace it by a unit clause in x . This has the effect of restoring the original number of variables, but reducing the frequency of the ϵn most frequent variables to just 1. Let G_i be the result of adding such unit clauses to F_i . So F is satisfiable iff some G_i is satisfiable, and $\overline{\text{freq}}(G_i) \leq \epsilon + (f + \delta)(1 - \epsilon) \leq f + \delta - (f - 1)\epsilon \leq f - \delta$. So we can decide the satisfiability of F by using a $(f - \delta)$ -Freq-SAT solver on each G_i , and this shows (10.4). $\square$

Theorem 10.17. d_Δ is a continuous function of Δ on the domain $[0, \infty)$.

Proof. Consider the following procedure applied to a CNF F with n variables and $\leq \Delta n$ clauses: while there is a pure literal, set it to true and simplify F by removing true clauses and false literals. The result is a CNF G with no pure literals and $G \in \text{SAT}$ iff $F \in \text{SAT}$. Now branch on some literal and simplify again. Note that both the true and false branches lead to the elimination of some clause since the branch literal is not pure.

If we repeat this procedure ϵn times, we will eliminate $\geq \epsilon n$ clauses. We can add back the eliminated variables in one big clause without altering the satisfiability of the formula. The resulting formula will have density $\leq \frac{(\Delta - \epsilon)n + 1}{n}$, and so $d_\Delta \leq d_{\Delta - \epsilon} + \epsilon$. Taking $\delta = \frac{\epsilon}{2}$ and replacing Δ by $\Delta + \delta$, we have

$$\forall \Delta, \epsilon > 0 \ \exists \delta > 0 \ d_{\Delta + \delta} - d_{\Delta - \delta} \leq \epsilon,$$

which shows that d_Δ is continuous. $\square$

Combining corollaries 10.2, 10.7, theorems 10.16, 10.17, and the intermediate value theorem, we obtain the following.

Theorem 10.18. $\forall k, c > 0 \ \exists f, \Delta \in [2^{\Theta(k)}, 2^{\Theta(ck \lg k)}]$

$$|\overline{\varphi}_f - s_k| \leq k^{-c}$$

$$|d_\Delta - s_k| \leq k^{-c}.$$

10.4 Impossibility of general sparsification

In [Sch05] it was conjectured that $c_{\text{CNF-SAT}}$ might be < 1 , which is slightly stronger than $\neg\text{SETH}$, and it was observed that at least this is true if we restrict to instances to have linearly many clauses [AS03]. (although corollary 10.9 proves this directly). This naturally makes one wonder whether a more general sparsification is possible, where we assume a polynomial bound on the number of clauses m , but no restriction on the maximum clause width k .

Such a general sparsification lemma would be useful if, say, one had a formula with bounded average clause width, in which case the number of clauses would be polynomially bounded.

Conjecture 10.19 (general sparsification). $\exists$ function $g \forall c, \epsilon > 0 \exists$ algorithm A
 $\forall F \in \text{CNF}, |F| \leq n^c, n = |\text{var}(F)| \ A(F)$ outputs $F_1, \dots, F_s \in \text{CNF}$ with $\text{poly}(n)$ delay and where
 $|\text{var}(F_i)| \leq n$ s.t.

1. $s \leq 2^{\epsilon n}$
2. $\text{sol}(F) \neq \emptyset$ iff $\cup_i \text{sol}(F_i) \neq \emptyset$
3. $\text{density}(F_i) \leq g(c, \epsilon)$
4. $\exists c' > 1 \ g(c', \epsilon) \in 2^{o(\frac{1}{\epsilon})}$. (a weak demand on the rate of growth of g)

We can weaken the demands of the conjecture considerably and still show that it is unlikely to be true. The following is implied by conjecture 10.19.

Conjecture 10.20. $\exists$ functions $\alpha(n) \in \omega(1), g(\epsilon) \in 2^{o(\frac{1}{\epsilon})} \forall \epsilon > 0 \exists$ a $O(2^{\epsilon n})$ -time Turing reduction from SAT_α to $\text{SAT}_{g(\epsilon)}$ that preserves the number of variables.

Theorem 10.21. $\text{ETH} \Rightarrow$ not conjecture 10.20.

Proof. Suppose indirectly that conjecture 10.20 holds. Then

$$\forall \epsilon > 0 \exists \Delta \in 2^{o(\frac{1}{\epsilon})} \ d_\alpha \leq d_\Delta + \epsilon,$$

which implies, by corollary 10.9,

$$\forall \epsilon > 0 \exists \delta \in \omega(\epsilon), l \ d_\alpha \leq s_l + \epsilon - \delta.$$

Since $\delta \in \omega(\epsilon)$, we have $\exists \epsilon > 0 \ \epsilon < \delta(\epsilon)$.

So $\exists l \ d_\alpha < s_l$, which implies, by lemma 10.10,

$$\exists \Delta' > 0 \ d_\alpha < d_{\Delta'} \leq d_\alpha,$$

a contradiction. □

Of course, if you are an optimist, then the contrapositive is the more interesting way to look at the above: if $\text{poly}(n)$ -sized formulas can be sparsified, then k -SAT can be solved fast.

10.5 Conclusion

Let us make 2 remarks. First, all of the reductions used in this chapter – sparsification, width reduction, and the variable-reduction technique from [IP01] – are all deterministic, and so all the results in this chapter hold relative to deterministic exponential complexity as well. (E.g. if we define $s'_k = dc_{k\text{-SAT}}$, $d'_\Delta = dc_{\text{SAT}_\Delta}$, then $s'_\infty = d'_\infty$, etc.)

Second, having shown the equivalence of SETH in terms of clause width, clause density, and variable frequency, we may ask whether for one of these restriction types algorithm design might be an intuitively easier process. After all, the power of proving equivalences is that something that might be hard to see from one perspective might be easier from another.

Let's make a metaphor. 3-SAT is, in some sense, “easy” compared to Integer Programming (IP), and this can be seen in (1) the ease with which one reduces 3-SAT to IP and (2) the difficulty in the reverse direction. Nonetheless, they are both NP-complete. The way we use this equivalence is that if we want to prove novel problem A is NP-hard, we may prefer to reduce 3-SAT (the easy problem) to A rather than reducing IP (the hard problem) to A , since we feel that a proof of the later would entail at least all the difficulty of reducing IP to 3-SAT, and perhaps more. Conversely, if we want to prove that some novel problem is in NP, we may prefer to reduce it to IP, since we may be able to leverage the nontrivial proof that IP is in NP, whereas 3-SAT being in NP is so trivial that reduction to 3-SAT is unlikely to save any work. Not surprisingly then, there are more nontrivial algorithms for 3-SAT than for IP – actually, there are none for IP.

So (A “easily” reduces to B but not vice versa) seems to imply that algorithmic design for A should be easier than for B . Given that sparsification is substantially more technically sophisticated than the other reductions we used, we should suspect that f -Freq-SAT may be “easy” compared to k -SAT. Yet, much more effort for algorithmic design has focused on k -SAT. Perhaps this is a mistake. It is true that, starting from a CNF of bounded width, sparsification yields a CNF of bounded width *and* bounded frequency, but we have already shown that the sparsification constant from [IPZ01] was somewhat loose, and it may still be. So it may be worthwhile to spend more effort looking at k -SAT algorithms where the frequency (or density) is closer to the satisfiability threshold, or perhaps just f -Freq-SAT problems more generally.

Acknowledgements

Material here is joint work with Russell Impagliazzo and Ramamohan Paturi and appears in [CIP06].

Chapter 11

Summary and open problems

Below, we summarize the main results of this work.

- We decreased the sparsification constant from $(\frac{8k2^k}{\epsilon})^{2^{k-1}}$ to $O(\frac{k}{\epsilon})^{3k}$.
- We reviewed the currently best worst-case upper bound k -SAT algorithms, and showed an upper bound on the PPZ k -SAT algorithm that decreases as a function of the number of solutions of the input formula.
- We showed $c_{\Pi_2} \text{ 3-SAT} \geq s_\infty$, providing an alibi for the lack of nontrivial algorithms at and beyond the 2nd level of the polynomial hierarchy, gave a canonical syntactic form to the hard instances of Π_2 3-SAT, and related the c_{Π_i} k -SAT constants for other values of i, k .
- We gave what we believe is the first nontrivial algorithm for deciding the satisfiability of circuits of n variables, d depth, cn gates. It has exponential complexity $1 - \alpha$ where $\alpha \geq 1/O(c^{2^{d-2}-1} \lg^{3 \cdot 2^{d-2}-2} c)$, and the constant in the big-Oh depends only on d .
- We thoroughly reviewed the Valiant-Vazirani reduction, derandomizing it when the input has $\leq 2^{o(n)}$ solutions, and discussing its limitations. We extended this reduction through the isolation lemma, upper bounding the unique satisfiability gap $s_k - \sigma_k$ by $O(\frac{\lg^2 k}{k})$, which implies $s_\infty = \sigma_\infty$, and showed that this gap is nearly optimal for oblivious hashing techniques.
- We showed that deciding whether a problem has exactly 1 solution, i.e. defines a function, has the same exponential complexity as deciding whether there is a solution for many standard problems, giving a technique that easily generalizes.
- We showed an intuitive functional relationship between $s_k, d_\Delta, \varphi_\Delta$: that these are approximately the same when $\lg \Delta$ is between $\Omega(k)$ and $O(k \lg k)$, which implies $s_\infty = d_\infty = \varphi_\infty = \sigma_\infty$.

If we could reduce the sparsification constant to $O(\frac{1}{\epsilon})^k$, we could simplify the last bullet point to “ $s_k, d_\Delta, \varphi_\Delta$ are approximately the same when $k = \Theta(\lg \Delta)$ ”. This seems so plausible that perhaps it is worth investigating. Is there a superior CNF-SAT algorithm that can exploit both limited clause width and variable frequency close to that of the satisfiability threshold? What non-oblivious techniques could possibly break the $\Omega(\frac{1}{k})$ barrier for the unique satisfiability gap? We already know a few properties of the structure of k -CNF solution spaces, given by the satisfiability coding lemma and the insensitivity of k -CNFs. Perhaps one of these or some other combinatoric property of the solution space structure will shed the necessary light. The alternative is to look into the syntax of the formulas. Are ETH and SETH equivalent? If not, does ETH imply any explicit lower bound on s_∞ ? If it is too much to hope for a nontrivial algorithm for the satisfiability of arbitrary circuits, what is the largest class for which we can find such algorithms?

11.1 Impact of quantum computers

Lastly, we should say something about quantum computation since it may be in the medium-term future. Peter Shor’s poly time quantum factoring and discrete log algorithms and Lov Grover’s quadratic speedup algorithm [Sho97, Gro96, BBHT96] show great potential for quantum computing. As we have observed before, Grover’s algorithm transforms a time t algorithm with success probability p into a time $O(t/\sqrt{p})$ algorithm with constant success probability. It was observed in [DKW05] that since all of the satisfiability algorithms discussed in this work – PPZ, PPSZ, LocalSearch, the Iwama-Tamaki algorithm, Schuler’s algorithm, the constant depth and linear size circuit solver in chapter 8 – can be expressed as exponential iterations of poly time algorithms, Grover’s algorithm can halve each of their exponential complexities.

This prompts the following question: given an algorithm A with exponential running time t , can we always transform it into an exponential iteration of a poly time algorithm B with success probability approximately $1/t$? Such a transformation would prime A for use in Grover’s algorithm and we could reap the full benefit of its quadratic speedup. Most backtracking algorithms have such a property, but not everything is a backtracking algorithm of polynomial depth, e.g. dynamic programming with an exponentially large DP table. What other algorithmic paradigms might not have such a property? Are there better algorithms for satisfiability lurking within them?

While normally randomness in an algorithm is often seen as a temporarily acceptable curse to be lifted as soon as possible, perhaps when designing algorithms in the future, citing the property that an algorithm can be randomized, in the sense of being transformed into an iteration of a fast algorithm with less success probability, may be seen as a feature.

Bibliography

- [AM02] Dimitris Achlioptas and Cristopher Moore. The asymptotic order of the random k -sat threshold. In *FOCS*, pages 779–788. IEEE Computer Society, 2002.
- [AP03] Dimitris Achlioptas and Yuval Peres. The threshold for random k -sat is $2^k \ln 2 - o(k)$. In *STOC*, pages 223–231. ACM, 2003.
- [APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979.
- [ART06] Dimitris Achlioptas and Federico Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. In Jon M. Kleinberg, editor, *STOC*, pages 130–139. ACM, 2006.
- [AS03] Vikraman Arvind and Rainer Schuler. The quantum query complexity of 0-1 knapsack and associated claw problems. In Toshihide Ibaraki, Naoki Katoh, and Hiroataka Ono, editors, *ISAAC*, volume 2906 of *Lecture Notes in Computer Science*, pages 168–177. Springer, 2003.
- [Aus07] Per Austrin. Balanced max 2-sat might not be the hardest. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 189–197. ACM, 2007.
- [BBHT96] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching, May 1996.
- [BDH93] Amir Ben-Dor and Shai Halevi. Zero-one permanent is $\#P$ -complete, a simpler proof. In *ISTCS*, pages 108–117, 1993.
- [Bea94] Paul Beame. A switching lemma primer. Technical Report TR UW-CSE-95-07-01, Department of Computer Science and Engineering, University of Washington, November 1994.
- [BKPS98] Paul Beame, Richard M. Karp, Toniann Pitassi, and Michael E. Saks. On the complexity of unsatisfiability proofs for random k -cnf formulas. In *STOC*, pages 561–571, 1998.
- [BKS07] Piotr Berman, Marek Karpinski, and Alexander D. Scott. Computational complexity of some restricted instances of 3-sat. *Discrete Applied Mathematics*, 155(5):649–653, 2007.
- [Bop97] Ravi B. Boppana. The average sensitivity of bounded-depth circuits. *Inf. Process. Lett.*, 63(5):257–261, 1997.

- [BW05] Fahiem Bacchus and Toby Walsh, editors. *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*. Springer, 2005.
- [Cal04] Chris Calabro. Upper bounds for k -sat. research exam (available on author's web page), 2004.
- [CG89] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *J. Complexity*, 5(1):96–106, 1989.
- [Che05] Hubie Chen. Existentially restricted quantified constraint satisfaction. *CoRR*, abs/cs/0506059, 2005.
- [CIKP08] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k -sat: An isolation lemma for k -cnfs. *J. Comput. Syst. Sci.*, 74(3):386–393, 2008.
- [CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for sat. In *CCC '06: Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, pages 252–260, Washington, DC, USA, 2006. IEEE Computer Society.
- [Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.
- [CP09] Chris Calabro and Ramamohan Paturi. k -sat is no harder than decision-unique- k -sat. In Anna E. Frid, Andrey Morozov, Andrey Rybalchenko, and Klaus W. Wagner, editors, *CSR*, volume 5675 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2009.
- [CPI09] Chris Calabro, Ramamohan Paturi, and Russell Impagliazzo. The complexity of satisfiability of small depth circuits. In *IWPEC*, 2009. (to appear in IWPEC 2009, LNCS).
- [CW77] Larry Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *STOC*, pages 106–112. ACM, 1977.
- [DD01] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In Bernhard Nebel, editor, *IJCAI*, pages 248–253. Morgan Kaufmann, 2001.
- [DGH⁺02] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2 - 2/(k + 1))^n$ algorithm for k -sat based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002.
- [DKW05] Evgeny Dantsin, Vladik Kreinovich, and Alexander Wolpert. On quantum versions of record-breaking algorithms for sat. *SIGACT News*, 36(4):103–108, 2005.
- [DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
- [DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

- [FK07] Martin Fürer and Shiva Prasad Kasiviswanathan. Algorithms for counting 2-sat solutions and colorings with applications. In Ming-Yang Kao and Xiang-Yang Li, editors, *AAIM*, volume 4508 of *Lecture Notes in Computer Science*, pages 47–57. Springer, 2007.
- [Fla09] Abraham D. Flaxman. *Algorithms for Random 3-SAT*, chapter (preprint to appear in Encyclopedia of Algorithms). Springer, 2009.
- [Fri99] Ehud Friedgut. Necessary and sufficient conditions for sharp thresholds of graph properties, and the k -sat problem. *J. Amer. Math. Soc.*, 12:1017–1054, 1999.
- [FS96] Alan M. Frieze and Stephen Suen. Analysis of two simple heuristics on a random instance of k -sat. *J. Algorithms*, 20(2):312–355, 1996.
- [FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.
- [GB97] Etienne Grandjean and Hans Kleine Büning. Sat-problems and reductions with respect to the number of variables. *J. Log. Comput.*, 7(4):457–471, 1997.
- [Geb09] Heidi Gebauer. Disproof of the neighborhood conjecture with implications to sat. *CoRR*, abs/0904.2541, 2009.
- [Gho06] Subhas Kumar Ghosh. Unique k -sat is as hard as k -sat. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(062), 2006.
- [GK06] Venkatesan Guruswami and Valentine Kabanets. Hardness amplification via space-efficient direct products. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 556–568. Springer, 2006.
- [GKMP09] Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009.
- [Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219, 1996.
- [Har67] L. H. Harper. A necessary condition on minimal cube numberings. *Journal of Applied Probability*, 4(2):397–401, 1967.
- [Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20. ACM, 1986.
- [Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
- [HS06] Shlomo Hoory and Stefan Szeider. A note on unsatisfiable k -cnf formulas with few occurrences per variable. *SIAM J. Discrete Math.*, 20(2):523–528, 2006.
- [IP01] Russel Impagliazzo and Ramamohan Paturi. On the complexity of k -sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
- [IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- [IT04] Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-sat. In J. Ian Munro, editor, *SODA*, page 328. SIAM, 2004.

- [JS99] David S. Johnson and Mario Szegedy. What are the least tractable instances of max independent set? *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 927–928, 1999.
- [JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.
- [KLM89] Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.
- [KS94] Scott Kirkpatrick and Bart Selman. Critical behaviour in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.
- [KST93] Jan Kratochvíl, Petr Savický, and Zsolt Tuza. One more occurrence of variables makes satisfiability jump from trivial to np-complete. *SIAM J. Comput.*, 22(1):203–210, 1993.
- [LLZ02] Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In William Cook and Andreas S. Schulz, editors, *IPCO*, volume 2337 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.
- [LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.
- [MNT93] Yishay Mansour, Noam Nisan, and Praseon Tiwari. The computational complexity of universal hashing. *Theor. Comput. Sci.*, 107(1):121–133, 1993.
- [MZK⁺99] Remi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic ‘phase transitions’. *Nature*, 400:133–137, July 1999.
- [NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.
- [PPSZ05] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k -sat. *J. ACM*, 52(3):337–364, 2005.
- [PPZ99] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999(115), December 1999.
- [PSZ00] Ramamohan Paturi, Michael E. Saks, and Francis Zane. Exponential lower bounds for depth three boolean circuits. *Comput. Complex.*, 9(1):1–15, 2000.
- [RK80] J.M. Buhrman R. Kaas. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, 34(1):13–18, 1980.
- [Rol05a] Daniel Rolf. Derandomization of ppsz for unique- k -sat. In Bacchus and Walsh [BW05], pages 216–225.
- [Rol05b] Daniel Rolf. Improved bound for the ppsz/schöning-algorithm for 3-sat. *Electronic Colloquium on Computational Complexity (ECCC)*, 12(159), 2005.
- [Rot96] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.
- [Rud87] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 3rd edition, 1987.

- [Sch99] Uwe Schöning. A probabilistic algorithm for k -sat and constraint satisfaction problems. In *FOCS*, pages 410–414, 1999.
- [Sch05] Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.
- [Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
- [Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.
- [Tra08] Patrick Traxler. The time complexity of constraint satisfaction. In Martin Grohe and Rolf Niedermeier, editors, *IWPEC*, volume 5018 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2008.
- [Vad01] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.
- [Val79a] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.
- [Val79b] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
- [VV86] Leslie G. Valiant and Vijay V. Vazirani. Np is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.
- [Wah05] Magnus Wahlström. Faster exact solving of sat formulae with a low number of occurrences per variable. In Bacchus and Walsh [BW05], pages 309–323.
- [Wei] Eric W. Weisstein. Stirling’s approximation. From MathWorld—A Wolfram Web Resource
<http://mathworld.wolfram.com/StirlingsApproximation.html>.
- [Wil02] Ryan Williams. Algorithms for quantified boolean formulas. In *SODA*, pages 299–307, 2002.
- [XZ06] Mingji Xia and Wenbo Zhao. #3-regular bipartite planar vertex cover is #p-complete. In Jin yi Cai, S. Barry Cooper, and Angsheng Li, editors, *TAMC*, volume 3959 of *Lecture Notes in Computer Science*, pages 356–364. Springer, 2006.