Algorithms and Extremal Properties of SAT and CSP

A dissertation submitted to
ETH ZÜRICH

for the degree of
DOCTOR OF SCIENCES

presented by
DOMINIK ALBAN SCHEDER
Master of Science, University of Colorado at Boulder
born 28. August 1980
citizen of Germany

accepted on the recommendation of
Prof. Dr. Emo Welzl, examiner
Prof. Dr. Ramamohan Paturi, co-examiner

2011
Contents

Abstract v

Zusammenfassung vii

Acknowledgments ix

1 Introduction 1
   1.1 Why SAT? .................................................. 3
   1.2 Algorithms for SAT ......................................... 4
   1.3 Extremal Combinatorics .................................... 11

2 Notation 17

I Algorithms for SAT and CSP 21

3 Local Search Algorithms for SAT 23
   3.1 Schöning’s Algorithm ..................................... 24
   3.2 The Algorithm by Dantsin et al. .......................... 29
   3.3 A Complete Derandomization of Schöning’s Algorithm . 32

4 Local Search Algorithms for CSP 41
   4.1 Schöning and cover-search for $(d, \leq k)$-CSP formulas ... 41
   4.2 A Better Deterministic Algorithm for $(d, k)$-CSP ............ 44
   4.3 A Deterministic Reduction from $(d, k)$-CSP to $k$-SAT .......... 52

5 Construction of Covering Codes 55
   5.1 Covering $\{0, 1\}^n$ With Hamming Balls ................. 56
   5.2 A General Framework for Covering Codes ................. 58
# CONTENTS

5.3 Application to 2-Boxes .............................................. 62  
5.4 Application to $G$-balls ........................................... 63  

6 PPZ for $(d, k)$-CSP .................................................. 67  
6.1 Introduction .......................................................... 68  
6.2 The Algorithm ....................................................... 72  
6.3 Analyzing the Success Probability ................................. 73  
6.4 A Correlation Inequality ............................................ 82  

II Extremal Combinatorics of CNF Formulas ......................... 87  

7 The Conflict Structure of CNF Formulas .......................... 89  
7.1 Introduction .......................................................... 89  
7.2 1-Conflicts .......................................................... 99  
7.2.1 Combinatorial Properties ...................................... 99  
7.2.2 The Main Result: Trivial – Tractable – Hard .......... 102  
7.2.3 Hardness for $|c_1| \geq k + 1$ ................................. 105  
7.3 Conflicts Generated by an Individual Variable .............. 107  
7.4 Total Number of Conflicts ........................................ 114  

8 Linear Formulas ....................................................... 125  
8.1 Introduction .......................................................... 125  
8.2 Existence and Size .................................................. 129  
8.3 Resolution Complexity ............................................. 132  
8.4 Linear MU(1)-Formulas .......................................... 134  

Bibliography .................................................................. 139
Abstract

This thesis treats algorithmic and combinatorial aspects of Boolean satisfiability and constraint satisfaction. Satisfiability is the problem of deciding whether a given propositional formulas in conjunctive normal form, short CNF formula, is satisfiable. For designing and analyzing algorithms one usually focuses on $k$-SAT, i.e., one requires that every clause of the formula contain at most $k$ literals for some constant number $k$, and measures the time complexity of the algorithm in terms of $n$, the number of variables in the formula. The most thoroughly studied case is 3-SAT, because this is the minimum $k$ for which $k$-SAT is NP-complete.

In the first part of this thesis we investigate algorithms for satisfiability and constraint satisfaction. In 1999, Uwe Schöning presented a simple randomized algorithm for these problems based on random walks. The expected running time is exponential but much faster than older algorithms. Its running time for 3-SAT is $O((4/3)^n poly(n))$. A few years later, Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan, and Schöning gave a deterministic algorithm that was inspired by the randomized one, but its running time is worse by an exponential factor. It solves 3-SAT in $O(1.5^n poly(n))$ time. We show how to improve the deterministic algorithm, ultimately closing the gap to Schöning’s random walk algorithm up to a subexponential factor. This is joint work with Robin Moser.

A different randomized algorithm was discovered by Paturi, Pudlák, and Zane in 1997. We show how the algorithm generalizes to $(d, k)$-CSP, i.e., constraint satisfaction problems where each variable can take on $d$ different values and every constraint consists of at most $k$ literals.

In the second part we study extremal combinatorial properties of satisfiability. A typical question involves some function $\mu$ that measures the structural complexity of CNF formulas. For example, $\mu$ could be the number of clauses in a formula or the number of conflicting clause pairs. Furthermore, we consider a subclass $\mathcal{P}$ of CNF formulas, usually the class of $k$-CNF formu-
las, in which every clause consists of exactly $k$ literals. We then ask “What is the maximal $d \in \mathbb{N}_0$ such that every CNF formula $F$ in $\mathcal{P}$ with $\mu(F) \leq d$ is satisfiable?”, or, equivalently, “What is the minimum $\mu(F)$ for $F$ being an unsatisfiable CNF formula in $\mathcal{P}$?” The answer to the second question equals the answer of the first one, plus 1. We are interested in the asymptotic behavior of this number of large $k$. For example, what is the smallest number of conflicting clause pairs in an unsatisfiable $k$-CNF formula? We show that this number is asymptotically between $2.69^k$ and $3.51^k$. This is joint work with Philipp Zumstein. For a second type of problems, we let $\mathcal{P}$ be the set of linear $k$-CNF formulas, i.e., formulas in which any two clauses have at most one variable in common. We prove close bounds on the number of clauses of unsatisfiable linear $k$-CNF formulas, namely that this number is at least $\Omega(4^k/k^2)$ and at most $O(k^24^k)$ and prove lower bounds on the treelike resolution complexity of such formulas.
Zusammenfassung


Auf einer ganz anderen Idee füsst ein randomisierter Algorithmus, den Paturi, Pudlák und Zane 1997 entdeckten. Wir verallgemeinern diesen für $(d,k)$-CSP, also Constraint-Satisfaction-Probleme, bei dene jede Variable einen von $d$ Werten annehmen kann (im Gegensatz zu lediglich zwei Werten im
booleschen Fall) und jedes Constraint aus höchstens $k$ Literalen besteht.

Im zweiten Teil untersuchen wir extremalkombinatorische Eigenschaften boolescher Erfüllbarkeit. Typischerweise betrachtet man eine Funktion $\mu$, die die strukturelle Komplexität von KNF-Formeln misst. Zum Beispiel könnte $\mu$ die Klauseln in einer Formel zählen oder die Anzahl von Klauselpaaren, die einen Konflikt bilden. Zusätzlich betrachtet man eine Untermenge $P$ aller KNF-Formeln, beispielsweise die Menge aller $k$-KNF-Formeln. Dies sind KNF-Formeln, bei denen jede Klausel aus genau $k$ Literalen besteht. Wir stellen nun folgende Frage: „Was ist die grösste natürliche Zahl $d$ für welche jede $k$-KNF-Formel $F$ mit $\mu(F) \leq d$ erfüllbar ist?“, oder, äquivalent dazu, „Was ist der kleinstmögliche Wert von $\mu(F)$, wenn $F$ eine unerfüllbare Formel aus $P$ ist?“ Die Antwort auf die zweite Frage ist gleich der Antwort auf die erste, plus 1. Uns interessiert das asymptotische Wachstum dieser Zahl in Abhängigkeit von $k$. Was ist zum Beispiel die kleinstmögliche Anzahl von Konfliktpaaren in einer unerfüllbaren $k$-KNF-Formel? Wir zeigen, dass diese Zahl asymptotisch zwischen $2.69^k$ und $3.51^k$ liegt. Dies geschah in Zusammenarbeit mit Philipp Zumstein.

Bei einem weiteren Problem sei $P$ die Menge aller linearen $k$-KNF-Formeln. Dies sind Formeln, in denen zwei verschiedene Klauseln ins höchstens einer Variable überlappen. Wir beweisen recht scharfe Schranken für die Anzahl von Klauseln in unerfüllbaren linearen $k$-KNF-Formeln, nämlich dass diese Zahl mindestens $\Omega(4^k/k^2)$ und höchstens $O(k^24^4)$ ist. Weiterhin zeigen wir untere Schranken für die Baumresolutionskomplexität solcher Formeln.
Acknowledgments

Let me start by expressing my gratitude to my advisor Emo Welzl. My years as his PhD student were enriching both academically and personally. Through his course on satisfiability he introduced me to the subject that finally became the topic of this thesis. He is definitely a researcher, teacher, and person out of whose book one should take a leaf. Furthermore, I am sure the open and collaborative atmosphere in our group owes a lot to him. I am grateful to Ramamohan Paturi for agreeing to be the co-examiner of this thesis. I enjoyed working together with my co-authors Timon Hertli, Claudia Käppeli, Robin Moser, and Philipp Zumstein, and hope there will be more joint work.

I want to thank all current members of the Gremo team: Andrea Francke, Andrea Salow, Anna Gundert, Bernd Gärtner, Emo Welzl, Gabriel Nivasch, Heidi Gebauer, Marek Sulovský, Martin Jaggi, Michael Hoffmann, Robin Moser, Sebastian Stich, Timon Hertli, Tobias Christ, Uli Wagner, and Yves Brise. Also the former members whom I had the pleasure to meet: Robert Berke, Joachim Giesen, Shankar Ram Lakshminarayanan, Dieter Mitsche, Andreas Razen, Leo Rüst, Eva Schuberth, Tibor Szabó, Patrick Traxler, and Philipp Zumstein.

In particular I am grateful to Andrea Salow for her support and the discussions on music; Andrea Francke for various fan facts; Anna for being a great neighbor, for proof-reading, and for her vegan cuisine; Bernd for having returned to the Gremo Sola team; Emo for the Sola afterparties; Gabriel for preventing us from becoming a purely German-speaking group; Heidi for proof-reading and reliably running one of the female-only tracks at Sola; Marek for proof-reading and the trip to Argentina; Martin for giving the Gremo team something to brag about in front of people who do not care about math; Michael for being awesome at the Töggelichaschte; Sebastian and Timon for proudly wearing beards in an otherwise thoroughly shaved group; Robin for broadening my horizon both academically and concerning the Zürich nightlife; Tobias for being my office mate and for explaining me the etymology of Chi-
inese characters; Uli for the coffee, his humor, and his wit; Yves and only Yves for the trip to Helsinki; Philipp for the trips to Lisbon, China, and Obwalden; Andreas for telling me about a certain TV series; Henning for being a great roommate; Florian for occasionally asking “Wer?”; Chrisse for having been the reason hiking up the Säntis; Reto for inviting me to Saarbrücken; Matuš for shaving his upper lip when everybody else was growing a mustache; Yann for showing up at our party at 8:50 pm and thinking he was late; Jan Foniok for the week in the Alps. Also I am deeply grateful to all my friends not already listed above.

Above all I am grateful to my family, to my parents Franz and Charlotte, to my sisters Julia and Dorothea, and to their children Samuel, Viktor, Henry, Theo, and Mathilda. Thank you for being there.
THEORETICAL computer scientists know all too well the moments in which a new acquaintance asks them what their subject is about. There are many possible answers, from serious to cheeky. When I want to give a serious answer, I usually say it is about drawing a line between simple problems and complex problems. Arguably, the notion of P and NP is probably the closest thing to such a line which theoretical computer science can offer.

The class of NP-complete problems is widely believed to lie beyond the boundaries of efficient computation. Although all NP-complete problems are, by definition, polynomially equivalent, some are "more equivalent" than others. A central place among them is held by SAT, the problem of deciding whether a given Boolean formula in conjunctive normal form is satisfiable.

In this thesis we study SAT from two viewpoints. The first one is algorithmic: Although we do not aspire to find a polynomial algorithm for SAT, there is a lot of progress that can be made within the realm of exponential algorithms. The second viewpoint is that of extremal combinatorics: We study extremal instances of SAT, that is, we try to find unsatisfiable formulas that minimize a certain parameter measuring their complexity. The goal is to gain
a better understanding of the line between satisfiability and unsatisfiability. One should keep in mind, however, that NP-completeness means that this border will remain elusive.

The objects we study in this thesis are propositional formulas in conjunctive normal form (short CNF formulas). For example,

$$(x \lor y \lor z) \land (x \lor \bar{y}) \land (\bar{x} \lor \bar{z})$$

is such a CNF formula. When we substitute Boolean constants (true and false, often represented by 1 and 0, respectively) for the variables in a CNF formula, it evaluates to 0 or 1. For example, when substituting $x \mapsto 1, y \mapsto 1, z \mapsto 0$, the above formula evaluates to 1, when substituting $x \mapsto 1, y \mapsto 1, z \mapsto 1$, it evaluates to 0. Such a substitution is called an assignment. A CNF formula is satisfiable if there is an assignment that satisfies $F$, i.e., makes it evaluate to 1. The problem of deciding whether a given CNF formula is satisfiable is called SAT and plays a central role in theoretical computer science.

Of particular interest are formulas in which every clause consists of exactly $k$ literals. These are called $k$-CNF formulas. If every clause consists of at most $k$ literals, we call it a ($\leq k$)-CNF formula. Consequently, $k$-SAT is the problem of deciding whether a given ($\leq k$)-CNF formula is satisfiable. The most intensively studied case is $k = 3$, because this is the smallest $k$ for which $k$-SAT is NP-complete.

A CNF formula is a formula in propositional logic, but it is also a combinatorial object. To underline its combinatorial nature and to simplify notation, we will view a clause as a set of literals and a CNF formula as a set of clauses. This removes duplicate literals and duplicate clauses, but since $\lor$ and $\land$ are idempotent operators, this makes no difference. For example, we will write the above formula as

$$F = \{ \{x, y, z\}, \{x, \bar{y}\}, \{\bar{x}, \bar{z}\} \}.$$ 

Assigning Boolean constants to variables in a CNF formula yields a new CNF formula: Remove all clauses that evaluate to 1, and from the remaining clauses remove all literals that evaluate to 0. We we write assignments as lists of variable substitutions, for example $\beta = [x \mapsto 1, y \mapsto 1]$. For the above formula $F$ we see that $F[\beta] = \{\{\bar{z}\}\}$.

**Constraint Satisfaction Problems**

SAT is a member of a larger family of Constraint Satisfaction Problems. The constraint satisfaction problems we consider in this thesis are similar to CNF
formulas, just that each variable can take on one of \( d \) different values. Consequently, a literal is an expression of the form \( x \neq c \), where \( c \in \{1, \ldots, d\} \). A constraint is a disjunction of literals, and a CSP formula is a conjunction of constraints. If each variable ranges over \( d \) values and each constraint consists of at most \( k \) literals, we call the formula a \((d, \leq k)\)-CSP formula. For example

\[
F = (x \neq 1 \lor y \neq 3) \land (x \neq 2 \lor z \neq 2) \land (x \neq 2 \lor y \neq 3)
\]

is a \((3, \leq 2)\)-CSP formula. The variable assignment mechanism of CNF formulas works for CSP formulas, as well: Substituting 2 for \( x \) makes the literal \( x \neq 1 \) evaluate to true and \( x \neq 2 \) evaluate to false, thus

\[
F[x \mapsto 2] = (z \neq 2) \land (y \neq 3).
\]

A CSP formula is satisfiable if there is an assignment that makes the formula evaluate to true. Note that a \((2, \leq k)\)-CSP formula is essentially the same as a \((\leq k)\)-CNF formula, although they look slightly different. To ease notation, we regard a CSP formula as a set of constraints and a constraint as a set of literals of the form \((x \neq c)\).

## 1.1 Why SAT?

Why does SAT play a central role in theoretical computer science and take such a prominent place among NP-complete problems? I think it is because SAT is both simple enough and general enough. Simple enough to allow combinatorial reasoning, and general enough to model all sorts of other problems in a quite natural fashion.

**SAT is simple.** If SAT is called the mother of all NP-complete problems, then the great-great-grandmother would be NDTM-ACCEPTANCE: Given the description of a nondeterministic Turing machine \( M \), a “yardstick” \( t \) and an input string \( x \), does \( M \) accept \( x \) within \( t \) steps? This problem is even more general then SAT, but it is definitely not simple. It is so general that the theorem “NDTM-ACCEPTANCE is NP-complete” becomes a triviality, whereas the Cook-Levin Theorem, “3-SAT is NP-complete”, is one of the most fundamental results in complexity theory.

Also, SAT is much more amenable to combinatorial reasoning than Turing machines are. Take for example the rich and deep topic of random \( k \)-SAT: It is natural to define a probability distribution on \( k \)-CNF formulas, and, as it happens, these distributions exhibit many interesting phenomena. In contrast
to this, defining a probability distribution on triples \((M, 1^t, x)\), the instances of \(\text{NDTM-ACCEPTANCE}\), feels much less natural and will probably not give very interesting results.

**SAT is general.** While SAT has a simple combinatorial structure, it is still general enough to serve as a “modeling language” for many problems. If you think that any NP-complete language \(L\) can model any other NP-language \(L'\), via many-one-reductions, you are of course right. However, some reductions are very natural, while some are less so. For example, formulating \(\text{HAMILTONIAN PATH}\) as a SAT problem is more or less straightforward, whereas reducing SAT to \(\text{HAMILTONIAN PATH}\) requires the design of several clever gadgets.

Another advantage of SAT is the variable substitution mechanism. Let \(F\) be a CNF formula, and suppose we have already decided that we want to set \(x\) and \(y\) to 1 and \(z\) to 0. We want to know whether we can complete this partial assignment to a satisfying assignment. We could call this the Assignment Completion Problem. Note that this is the same as asking whether the CNF formula \(F[\overline{x} \mapsto 1, y \mapsto 1, z \mapsto 0]\) is satisfiable. We conclude that an instance of the Assignment Completion Problem is again an instance of SAT.

Consider the same scenario for \(\text{HAMILTONIAN PATH}\). We are given a graph, and suppose we have already decided that the edges \(e_1, e_2, \) and \(e_3\) should be part of our Hamiltonian path. Can we complete this to a Hamiltonian path? This Hamiltonian Path Completion Problem is in itself not an instance of \(\text{HAMILTONIAN PATH}\), and I do not see any direct way to make it one. Maybe there exists a way, but it is definitely not as simple and immediate as for SAT.

### 1.2 Algorithms for SAT

Since SAT is NP-complete, there is little hope for finding an efficient algorithm, and researchers have settled for the more modest goal of finding *moderately exponential* algorithms. Typically one expresses the running time in terms of \(n\), the number of variables in a formula. In the Boolean case, there are \(2^n\) assignments to these variables, thus a brute-force search yields an \(O(2^n \text{poly}(n))\) algorithm. A moderately exponential algorithm is one running in time \(O(a^n \text{poly}(n))\), where \(a\) is smaller than 2. Interestingly, for SAT in its full generality no such algorithm is known. However, if we restrict the size of clauses, i.e., when focusing on \(k\)-SAT for some constant \(k\), quite sophisticated algorithms have been discovered.
The earliest papers investigating algorithms for SAT are *A computing procedure for quantification theory* by Davis and Putnam [DP60] from 1960 and *A machine program for theorem-proving* by Davis, Logemann, and Loveland [DLL62] from 1962. The titles nicely illustrate that SAT has its roots in mathematical logic and artificial intelligence, and not in combinatorics. Also note that the papers appeared roughly ten years before the notion of NP-completeness was discovered by Cook and Levin [Coo71, Lev73].

### Splitting Algorithms

The algorithms presented in [DP60, DLL62] already contain a key feature of many subsequent SAT algorithms: Splittings. Let $F$ be a CNF formula and $x$ a variable. Then $F$ is satisfiable if and only if $F[x \mapsto 0]$ is satisfiable or $F[x \mapsto 1]$ is. Thus one can decide satisfiability of $F$ by recursively solving $F[x \mapsto 0]$ and $F[x \mapsto 1]$. If $F$ contains $n$ variables, then both subsequent formulas contain at most $n - 1$ variables. Naively implemented, this gives an $O(2^n \text{poly}(n))$ algorithm solving SAT. This feature is sometimes called *self-reducibility*, a feature exhibited by many NP-complete problems, although, as discussed above, it is not always as obvious as for SAT.

For small $k$, one can solve $k$-SAT in significantly less than $O(2^n)$ steps: Consider 3-SAT. Take any clause $C \in F$. Suppose $C$ has three literals, i.e., $C = \{u, v, w\}$. Note that $F$ is satisfiable if and only if at least one of the three formulas $F[u \mapsto 1]$, $F[u \mapsto 0, v \mapsto 1]$, and $F[u \mapsto 0, v \mapsto 0, w \mapsto 1]$ is satisfiable. Recursing on these formulas, we obtain an algorithm running in time $O(t_n \text{poly}(n))$, where $t_0 = t_1 = t_2 = 1$ and $t_n = t_{n-1} + t_{n-2} + t_{n-3}$, for $n \geq 3$. Using standard methods, one can show that $t_n \in \Theta(a^n)$ for $a \approx 1.84$ being the unique positive root of $a^3 - a^2 - a - 1$. Note that this algorithm can be improved if $F$ contains a clause of size 2 (or 1, or 0). In [MS85], Monien and Speckenmeyer made the crucial observation that "in most cases", $F$ contains a 2-clause. Thus, they improved the running time to $O(b^n \text{poly}(n))$, where $b \approx 1.619$ is the unique positive root of $x^2 - x - 1$ (this is the inverse of the golden ratio). Their algorithm is a good illustration how in the field of exponential algorithms, simple but non-obvious insights can lead to significantly faster algorithms.

Over time, researchers have improved the worst-case running time for 3-SAT to $O(1.505^n)$ (Kullmann [Ku99]) and $O(1.476^n)$ (Rodošek [Rod96]). Unfortunately, these algorithms became more and more complicated.
Chapter 1. Introduction

Non-Splitting Algorithms

Since the mid-90s, a new generation of SAT-algorithms has emerged. They differ from previous ones in two crucial points: They do not use splittings and they are randomized. In 1997, Paturi, Pudlák, and Zane \textsuperscript{[PPZ99]} presented a simple randomized algorithm that can be seen as inspired by splitting algorithms, but the analysis of its running time is completely different. Their algorithm, henceforth abbreviated \textit{ppz}, works as follows: Pick a variable \(x\) randomly from all variables in \(F\), and set it randomly to 0 or 1, unless the correct value of \(x\) is “clear”. Here, “clear” means that \(F\) contains a unit clause \(\{x\}\) (in which case 1 is clearly the correct value), or \(\{\bar{x}\}\) (in which case it is 0). Paturi, Pudlák, and Zane show that if \(F\) is a satisfiable \((\leq k)\)-CNF formula, than this procedure finds a satisfying assignment with probability at least \(2^{-(1-1/k)n}\). Repeated \(2^{(1-1/k)n}\) times, this gives a Monte Carlo algorithm for \(k\)-SAT. For \(k = 3\), the running time is \(O(2^{2n/3}\text{poly}(n)) \subseteq O(1.588^n)\). This is much slower than the best splitting algorithms mentioned above, but notice the fantastic simplicity of this algorithm! Now let us change the meaning of “clear” in this algorithm: Set \(x\) randomly unless there is a constant-size subformula of \(F\) that implies \(\{x\}\) (in which case we set it to 1) or implies \(\bar{x}\) (set it to 0). This gives a much more powerful algorithm, called \textit{ppsz} after its authors Paturi, Pudlák, Saks, and Zane \textsuperscript{[PPSZ05]}. Simple though this algorithm appears, analyzing its success probability turns out to be challenging. For 3-SAT, the authors prove a lower bound of \(1.362^{-n}\), but the true value is still not known.

A completely different approach was discovered by Schöning \textsuperscript{[Sch99]}, based on random walks. Given a CNF formula \(F\), choose some assignment \(\alpha\) uniformly at random. Then locally correct it up to \(3n\) times. That means, pick an unsatisfied clause (if there is any), pick a variable therein at random, and change the value \(\alpha\) assigns to it. Thus that clause is satisfied, but other clauses might become unsatisfied. In this thesis, we call this algorithm \textit{Schöning}. The idea of using random walks for SAT dates back to Papadimitriou \textsuperscript{[Pap91]}, who used it to obtain a quadratic algorithm for 2-SAT. For \(k \geq 3\), Schöning proved that if \(F\) is a satisfiable \((\leq 3)\)-CNF formula, the algorithm \textit{Schöning} finds a satisfying assignment with probability at least \(\left(\frac{k}{2(k-1)}\right)^n\). For \(k = 3\), this gives a Monte Carlo algorithm for 3-SAT running in time \(O(1.334^n)\). This is much faster than everything we have seen so far.

This is not the end of the story. First, Schöning’s algorithm can be improved if the initial assignment \(\alpha\) is not chosen uniformly at random, but tak-

\[^1\]Here, “imply” is meant in its literal sense from logic: A formula \(G\) implies \(x\) if any assignment that satisfies \(G\) sets \(x\) to 1.
ing the structure of $F$ into account \cite{HSSW02,Rol03}. Second, Iwama and Tamaki \cite{IT04} showed that ppsz performs better on formulas with few satisfying assignments, while Schöning is better on formulas with many satisfying assignments. Thus, running both ppsz and Schöning simultaneously should improve the success probability in general. The following table gives an overview of the stepwise improvements for $3$-SAT.

<table>
<thead>
<tr>
<th>Time</th>
<th>Authors</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1.3633^n$</td>
<td>Pudlák, Paturi, Saks, and Zane \cite{PPSZ05}</td>
<td>ppsz</td>
</tr>
<tr>
<td>$1.334^n$</td>
<td>Schöning \cite{Sch99}</td>
<td>Schöning</td>
</tr>
<tr>
<td>$1.3302^n$</td>
<td>Hofmeister et al. \cite{HSSW02}</td>
<td>Schöning improved</td>
</tr>
<tr>
<td>$1.3279^n$</td>
<td>Rolf \cite{Rol03}</td>
<td>Schöning improved</td>
</tr>
<tr>
<td>$1.3237^n$</td>
<td>Iwama and Tamaki \cite{IT04}</td>
<td>ppsz + Schöning</td>
</tr>
<tr>
<td>$1.3221^n$</td>
<td>Rolf \cite{Rol05}</td>
<td>ppsz + Schöning</td>
</tr>
<tr>
<td>$1.3215^n$</td>
<td>Hertli, Moser, and Scheder \cite{HMS}</td>
<td>ppsz + Schöning improved</td>
</tr>
<tr>
<td>$1.3211^n$</td>
<td>Iwama, Seto, Takai, and Tamaki \cite{ISTT}</td>
<td>ppsz + Schöning improved</td>
</tr>
<tr>
<td>$1.321^n$</td>
<td>Hertli, Moser, and Scheder \cite{HMS}</td>
<td>ppsz + Schöning improved</td>
</tr>
<tr>
<td>$1.307^n$</td>
<td>Hertli \cite{Her11}</td>
<td>ppsz</td>
</tr>
</tbody>
</table>

**Deterministic Algorithms**

The algorithms Schöning and ppsz, as well as their improvements and combinations, are all randomized. This immediately raises the question of what we can achieve with deterministic algorithms. Shortly after Schöning published his algorithm, Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning \cite{DGH+02} presented a deterministic version of it, albeit with a worse running time, namely $O\left(\left(\frac{2^n}{n} \right) \text{poly}(n)\right)$ for $k$-SAT. For $k = 3$, this is $O(1.5^n \text{poly}(n))$. Since then, several researchers have improved upon this running time. We give a summary over this development for $k = 3$, ignoring polynomial factors in $n$.  

<table>
<thead>
<tr>
<th>Time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1.334^n)$</td>
<td>Schöning \cite{Sch99}, randomized</td>
</tr>
<tr>
<td>$O(1.5^n)$</td>
<td>Dantsin et al. \cite{DGH+02}</td>
</tr>
<tr>
<td>$O(1.481^n)$</td>
<td>Dantsin et al. \cite{DGH+02}</td>
</tr>
<tr>
<td>$O(1.473^n)$</td>
<td>Brueggemann and Kern \cite{BK04}</td>
</tr>
<tr>
<td>$O(1.465^n)$</td>
<td>Scheder \cite{Sch08}</td>
</tr>
<tr>
<td>$O(1.439^n)$</td>
<td>Kutzkov and Scheder \cite{KS10}</td>
</tr>
<tr>
<td>$O(1.334^n)$</td>
<td>Moser and Scheder \cite{MS10}</td>
</tr>
</tbody>
</table>
Chapter 1. Introduction

Results

In Chapter 3, we present a simple deterministic version of Schöning’s algorithm solving $k$-SAT in time $O \left( \left( \frac{2k}{k+1} \right)^{n+o(n)} \right)$, exceeding the randomized running time by only a subexponential factor. This is joint work with Robin Moser [MS10]. In Chapters 4, we generalize the algorithm by Dantsin et al. [DGH+02] to $(d, k)$-CSP problems and study a surprising way to improve it for $d \geq 3$. Finally we show how our deterministic version of Schöning’s algorithm for $k$-SAT can be turned into an algorithm for $(d, k)$-CSP. This too is joint work with Robin Moser. Finally, in Chapter 6, we generalize the algorithm ppz to $(d, k)$-CSP and provide an analysis of its success probability.

Methods

Dantsin et al. make Schöning algorithm deterministic in two steps: First, instead of choosing the initial assignment randomly, they construct a covering code. Roughly speaking, they cover the solution space $\{0, 1\}^n$ with Hamming balls of a certain radius, one of which is guaranteed to contain a satisfying assignment – provided the input formula is satisfiable. This derandomizes Schöning’s random choice of the initial assignment. The subsequent random walk is replaced by a recursive procedure. This procedure decides deterministically whether a given Hamming ball contains a satisfying assignment. Unfortunately, the running time of this procedure is suboptimal and makes the whole algorithm considerably slower than Schöning.

Our main achievement is to replace the recursive procedure of Dantsin et al. by a second application of covering codes, but now the code is not binary, but over an alphabet of size $k$. Simple though this seems, it took almost decade to be discovered.

In Chapter 6 we generalize ppz to $(d, k)$-CSP. This itself is straightforward. Analyzing its success probability is more difficult. For the analysis of ppz for $k$-SAT, one basically has to consider one clause per variable, but for $(d, k)$-CSP, one considers up to $d - 1$ clauses per variable, which may intersect in complex patterns. We have to study the correlation between certain random variables to prove that a certain simple case which we can analyze is indeed the worst case.
A Bigger Picture

Why do we try to find faster and faster algorithms for SAT? After all, we are pretty much convinced that no algorithm will truly be fast—given that SAT is NP-complete and most likely $P \neq NP$. Furthermore, the improvements seem to become smaller and smaller. Does this go on forever, or is there some insurmountable barrier? To formalize this question, Impagliazzo, Paturi, and Zane [IPZ01] define

$$c_k := \inf\{c \geq 0 \mid 3\text{-SAT can be solved in time } O(2^{cn})\}$$

and formulate the exponential time hypothesis, short ETH:

**Exponential Time Hypothesis [IPZ01]:** $c_3 > 0$. In other words, there is some $c > 0$ such that 3-SAT cannot be solved in time $O(2^{cn})$.

Employing a non-trivial sparsification algorithm, Impagliazzo, Paturi, and Zane [IPZ01] proved that if $c_3 = 0$, then $c_k = 0$ for all $k \geq 3$. This means that the choice of $k = 3$ in the formulation of ETH is less arbitrary than it first appears. If ETH holds, then the exact value of $c$ is probably less interesting than the question which algorithm achieves it. Is there a fairly simple algorithm running in time $c^{n+o(n)}$? Or, can we approach $c^n$ only by more and more complicated algorithms? Are these algorithms based on random walks? Is it similar to ppz and ppsz? Does it follow a completely different approach? Can it be made deterministic?

Although most experts believe that $c_3 > 1$, there is no good candidate for it at the moment. Let us do the following thought experiment: Tomorrow some smart person comes up with an algorithm solving 3-SAT in time $O(2^{n/3}\text{ poly}(n))$, which is faster than current algorithms. Suppose further that for the next ten to twenty years nobody could improve upon this running time. I bet that many people will conjecture that $c_3 = \sqrt[3]{2}$, based solely on the lack of progress. My hope is that by consistently trying to improve current algorithms and to discover new ones, we will at some point hit such a border.

Maybe more important than the exact value of $c_3$ is the asymptotical behavior of $c_k$ as $k$ grows. The running time of all currently known algorithms for $k$-SAT converges towards the trivial upper bound of $2^n$ for large $k$. The conjecture that the $c_k$ converge to 1 is known as the strong exponential time hypothesis (SETH [IPZ01]). For large $k$, we observe that Schöning’s algorithm [Sch99], ppsz [PPSZ05], and as well ppz [PPZ99] have an asymptotic running time of the form $2^{(1-c/k)n}$, where the constant $c$ is among those three
algorithms smallest for \( \text{ppsz} \) and largest for \( \text{ppz} \). We do not know of any algorithm for \( k \)-SAT beating this asymptotic behavior, i.e., it is unknown whether \( c_k \in 1 - 1/o(k) \). Although it is probably premature to conjecture that \( c_k \in 1 - 1/\Omega(k) \), finding an algorithm beating that bound would clearly be surprising. This comes with a nice piece of advice: If you discover a new algorithm for \( k \)-SAT, it might be very interesting even if its running time for \( 3 \)-SAT looks unimpressive. In any case, you should check its asymptotic behavior for large \( k \). Let us give a list of conjectures about the complexity of \( k \)-SAT, sorted from weak to strong:

- \( \text{P} \neq \text{NP} \), meaning that \( k \)-SAT cannot be solved in polynomial time.
- Exponential Time Hypothesis (ETH [IPZ01]): \( c_3 > 0 \). Equivalently, \( c_k > 0 \) for all \( k \geq 3 \).
- Strong Exponential Time Hypothesis (SETH [IPZ01]): \( \lim_{k \to \infty} c_k = 1 \).
- \( c_k \in 1 - 1/\Omega(k) \).

Why do we care whether a certain algorithm can be made deterministic? First, I think that it is a fundamental question in complexity theory whether randomness can really speed up computation, or whether it is just a convenient tool for us to deal with our lack of knowledge. Second, if a deterministic algorithm finds out that a given formula is unsatisfiable, then this serves as a proof of unsatisfiability. For example, most splitting algorithms implicitly build up a resolution proof when running on an unsatisfiable formula. Thus, there is a connection between resolution complexity and splitting algorithms. Can we define a different proof system for unsatisfiability that relates to deterministic local search algorithms in a way in which resolution relates to splitting algorithms?

Finally, recent results have shown that non-trivial deterministic algorithms and non-trivial probabilistic algorithms for NP-complete problems can have very different consequences. Consider \text{CIRCUITSAT}, the problem of deciding whether a given boolean circuit with \( n \) input variables and \( m \) gates can be made to evaluate to 1. Ryan Williams [Wil10] showed that if there is a deterministic algorithm beating the trivial running time of \( O(2^n \text{poly}(m)) \) for \text{CIRCUITSAT} by any superpolynomial factor in \( n \), then \( \text{NEXP} \) does not have polynomial circuits. Actually many people, maybe most, believe that \( \text{NEXP} \) does not have polynomial circuits, but a proof of this would be a major breakthrough. On the other hand, Paturi and Pudlák [PP10] showed that if there is a one-sided error polynomial-time randomized algorithm solving \text{CIRCUITSAT} with success probability of at least \( 2^{1-\epsilon n} \), for any \( \epsilon > 0 \), then \text{CIRCUITSAT}
can be solved in subexponential randomized time. This would violate ETH, since a $k$-CNF formula is a boolean circuit. We see that deterministic and randomized algorithms can have very different consequences. Therefore, it is important to think about which randomized algorithms we can derandomize, and which deterministic exponential-time algorithms one can transform into polynomial-time randomized algorithms with exponentially small success probability.

### 1.3 Extremal Combinatorics

We introduce the subject of extremal combinatorics of CNF formulas by discussing a toy problem. Consider the CNF formula

$\{\{x, y, z\}, \{\bar{x}, y, u\}, \{\bar{x}, y, \bar{z}\}, \{\bar{u}, y, z\}, \{\bar{x}, \bar{y}, \bar{u}\}\}$.

It is not difficult to see that this formula is satisfiable. For example, the assignment $x \mapsto 1, y \mapsto 1, z \mapsto 1, u \mapsto 0$ satisfies it. But now imagine that in two thousand years some archaeologist discovers, in the charred remnants of some ETH library, a copy of this thesis. The above formula is barely readable anymore, and all our geologist can make out is

$\{\{x, *, *\}, \{*, *, u\}, \{*, \bar{y}, *\}, \{*, *, *\}, \{*, *, z\}, \{\bar{x}, *, \bar{u}\}\}$,

where "*" stands for a character that cannot be read anymore. Can she determine whether this formula is satisfiable? Yes, she can! Her computer science friend tells her that any $3$-CNF formula with fewer than eight clauses is satisfiable. Why is this so? Set every variable to $0$ or $1$ uniformly at random, independently. A $3$-clause is unsatisfied with probability $1/8$. By linearity of expectation, the expected number of unsatisfied clauses in the above formula is $6 \cdot 1/8$, which is less than $1$. Therefore, there must be some assignment for which the number of unsatisfied clauses is less than $1$, hence $0$, i.e., an assignment that satisfies the formula. We can easily generalize this statement:

Any $k$-CNF formula with fewer than $2^k$ clauses is satisfiable.

It is not difficult to see that this is tight, i.e., that there are unsatisfiable $k$-CNF formulas with $2^k$ clauses, for every $k$. Note that it is important to distinguish between $(\leq k)$-CNF formulas, in which each clause has at most $k$ literals, and $k$-CNF formulas, in which each clause has exactly $k$ literals. The above observation opens the door to extremal combinatorics of CNF formulas, a subject which we investigate in the second part of this thesis. The generic extremal
combinatorial question reads like this: We have some integer-valued complexity measure $\mu$ and some property $\mathcal{P}$ of CNF formulas and want to answer the question

What is the largest $d \in \mathbb{N}_0$ such that every CNF formula $F$ with property $\mathcal{P}$ and $\mu(F) \leq d$ is satisfiable?

Let us denote this number $d$ by $\text{ex}(\mu, \mathcal{P})$. If we define $\mu(F) := |F|$, the number of clauses, and let $\mathcal{P}$ be the property of being a $k$-CNF formula, then the observation by our archeologist’s computer science friend can be succinctly stated as

$$\text{ex}(|\cdot|, k\text{-CNF}) = 2^k - 1.$$ 

Related Work

We will give some examples of extremal combinatorial questions concerning CNF formulas that have been investigated in the past.

Clause Neighborhoods For a clause $C$ and a CNF formula $F$, define the neighborhood of $C$ in $F$ to be

$$\Gamma_F(C) := \{ D \in F \mid C \neq D, \text{vbl}(C) \cup \text{vbl}(D) \neq \emptyset \}$$

and $l(F) := \max\{|\Gamma_F(C)| \mid C \in F\}$. What is $\text{ex}(l, k\text{-CNF})$? In words, what is the largest $d$ such that every $k$-CNF formula $F$ with $l(F) \leq d$ is satisfiable? One obtains a lower bound from the famous Lovász Local Lemma [EL75]:

Let $F$ be a $k$-CNF formula with $l(F) \leq 2^k/e - 1$. Then $F$ is satisfiable.

Consider the $k$-CNF formula $F$ consisting of all $2^k$ clauses over the variables $x_1, \ldots, x_k$. This formula is unsatisfiable, and $l(F) = 2^k - 1$. Thus,

$$2^k/e - 1 \leq \text{ex}(l, k\text{-CNF}) \leq 2^k - 1.$$ 

Recently, Gebauer, Szabó, and Tardos [GST10] managed to determine this parameter up to lower order terms: $\text{ex}(l, k\text{-CNF}) = (1 + o(1))2^k/e$. Here, $o(1)$ is some function that tends to 0 as $k$ grows.
1.3. Extremal Combinatorics

**Variable Degree**  The *degree* of a variable $x$ in a CNF formula $F$ is

$$\deg_F(x) := |\{C \in F \mid x \in \text{vbl}(C)\}|,$$

and $\deg(F) := \max\{\deg_F(x) \mid x \in \text{vbl}(F)\}$ is the *maximum degree* of $F$. Note that if $F$ is a $k$-CNF formula, then

$$l(F) \leq k(\deg(F) - 1).$$

Again, our extremal combinatorial question is

What is the largest $d \in \mathbb{N}_0$ such that every $k$-CNF formula $F$ with $\deg(F) \leq d$ is satisfiable?

More succinctly, what is $\varex(\deg, k$-CNF$)$? From the above discussion, it follows that

$$\varex(\deg, k$-CNF$) \geq (\varex(l, k$-CNF$) + 1)/k \geq \frac{2^k}{ek}.$$

This result is due to Kratochvíl, Savický, and Tuza [KST93]. For almost two decades, the true asymptotics of $\varex(\deg, k$-CNF$)$ have been unknown. In 2009, Gebauer [Geb09] constructed unsatisfiable $k$-CNF formulas $F_k$ with $\deg(F_k) \leq 2^{k+2}/k$, thus determining this parameter up to a constant factor. Shortly afterwards, Gebauer, Szabó, and Tardos [GST10] also determined the correct constant factor:

$$\varex(\deg, k$-CNF$) = (1 + o(1)) \frac{2^{k+1}}{ek}$$

**Results**

We briefly outline the extremal combinatorial problems that we investigate in this thesis. The first concerns *conflicts*, whereas the latter deals with *linear formulas*.

**Conflicts**  We say two clauses $C, D$ have a *conflict* if $C \cap \bar{D} \neq \emptyset$. For example, \{x, y, z\} and \{\bar{x}, \bar{y}, u\} have a conflict. For a CNF formula, we can count the total number of pairs of clauses that have a conflict. We denote this number by $\text{gc}(F)$. The $g$ stands for *global* and the $c$ for *conflicts*. Intuitively, a $k$-CNF formula with very few conflicts should be satisfiable. Consequently, in Chapter 7, we try to determine the asymptotics of $\varex(\text{gc}, k$-CNF$)$. It is not difficult to show that

$$2^k - 1 \leq \varex(\text{gc}, k$-CNF$) \leq \binom{2^k}{2} = \Theta(4^k).$$
Our main result from Chapter 7 is to provide non-trivial upper and lower bounds:

\[ \Omega(2^{0.69k}) \leq \text{ex}(gc, k\text{-CNF}) \leq O(3.51^k). \]

We suspect that the correct asymptotics is of the form \( a^k + o(k) \). However, our above result rules out the two “obvious” candidates for \( a \), namely \( a = 2 \) and \( a = 4 \). We have no idea which number in \([2.69, 3.51]\) would be a reasonable candidate.

In addition to \( gc \), we investigate other notions of conflicts. For example, two clauses \( C, D \) have a 1-conflict if \( |C \cap \overline{D}| = 1 \). Again, we can ask similar questions as above. Of all extremal parameters, this seems to be the only one we completely understand. In particular, all results concerning 1-conflicts are proven using elementary combinatorial reasoning, whereas for most other results we need the Lovász Local Lemma (for proving lower bounds) and probabilistic constructions (for proving upper bounds). The results of Section 7.3 and 7.4 are joint work with Philipp Zumstein [SZ08a, SZ08b]. The results of Section 7.2 have not yet been published.

### Linear Formulas

In the extremal combinatorial questions above we study what happens if we restrict the number of “interactions” between clauses in a formula. Let us now restrict not the number, but the quality of such interactions. Call a CNF formula \( F \) linear if any two clauses have at most one variable in common, i.e., \( |\text{vbl}(C) \cap \text{vbl}(D)| \leq 1 \). Clearly, if we require that this intersection be empty for any two clauses, then \( F \) is satisfiable, unless it contains the empty clause. We see that linearity is quite severe a restriction. It is not obvious whether unsatisfiable \( k \)-CNF formulas exist at all. In Chapter 8 we investigate the size and structure of such formulas. We try to determine \( \text{ex}(|\cdot|, \text{linear } k\text{-CNF}) \) and prove that

\[ \frac{4k}{8e^2k^2} - \frac{2^k}{4e(k-1)} \leq \text{ex}(|\cdot|, \text{linear } k\text{-CNF}) \leq 4k^24^k \]

In words, there are unsatisfiable linear \( k\)-CNF formulas with at most \( 4k^24^k \) clauses, and every linear \( k\)-CNF formula with at most \( 4^k/(e^2k^2) \) clauses is satisfiable. Furthermore, we show that unsatisfiable linear \( k\)-CNF formulas need to exhibit a complex structure, for example their treelike resolution complexity is rather large. The results of this chapter have been published in [Sch10].

### Methods

Most results in Chapter 7 and 8 use probabilistic methods. To prove upper bounds on extremal parameters, i.e., to prove that certain unsatisfiable formu-
las exists, we usually sample a formula from a suitable distribution and prove that with positive probability it is unsatisfiable. Building a random CNF formula over the variable set \( V \) can be thought of as a two-step process: First, we build some random hypergraph with vertex set \( V \). Second, we assign positive and negative signs, i.e., we replace each hyperedge \( e = \{x_1, \ldots, x_k\} \) by a clause \( \{u_1, \ldots, u_k\} \), choosing \( u_i \in \{x_i, \bar{x}_i\} \) according to some distribution. Interestingly, in our applications only one of these two steps is random, while the other is deterministic. For example, in Chapter \[7\] we construct unsatisfiable formulas with a ”sparse” conflict structure. For this, we build a random hypergraph, but assign positive and negative signs in a simple deterministic manner. This is in contrast to Chapter \[8\]. Here, we use an algebraic technique to construct the underlying hypergraph, but then randomly choose positive and negative signs.

Proving a lower bounds for an extremal parameter means showing that a certain CNF formula is satisfiable. We do this mostly using some version of the Lovász Local Lemma. Typically, a naive application of this fails. We have to subject our formula to some ”truncation process” that shaves literals off its clauses, thus making the formula less satisfiable, but in some way making satisfiability easier to detect for the Lovász Local Lemma. Especially when proving a lower bound on \( \text{ex}(gc, k\text{-CNF}) \), this becomes quite technical.

For proving lower bounds on the treelike resolution complexity of unsatisfiable linear \( k\text{-CNF} \) formulas we use the idea of taking a random walk in a given resolution tree. To our knowledge, this is quite different from the methods usually used for proving lower bounds on resolution complexity.
Chapter 2

Notation

The notation of this thesis follows the notation of Emo Welzl’s lecture notes [Wel05]. Should the reader come across some notation he or she does not understand, we recommend he or she first search for a definition on the current and previous page; second, search in this notation chapter; third, look at Emo Welzl’s lecture notes. If all this does not help, the reader is welcome to ask me for clarification, preferably before this thesis has been submitted to ETH Zürich.

Variables, Clauses, Formulas.

We denote Boolean variables by $x, y, \ldots$ and literals by $u, v, \ldots$ A literal is either a Boolean variable $x$ or its negation $\overline{x}$. We can also negate literals: Since *duplex negatio affirmat*, we set $\overline{u} := x$ if $u = \overline{x}$. A pair of literals $u, v$ is contradicting if $v = \overline{u}$. A clause is a set of non-contradicting literals, interpreted as the disjunction ($\lor$). A CNF formula is a set of clauses, interpreted as their conjunction ($\land$). By convention, an empty disjunction always evaluates to 0. We denote this empty clause by $\square$, which is always unsatisfied. An empty conjunction evaluates to 1. This is the empty formula, denoted by $\{\}$, which is always satisfied. We use letters $F, G, H$ for formulas and $C$ and $D$ for clauses. We define $\text{vbl}(x) = \text{vbl}(\overline{x}) := x$, $\text{vbl}(C) = \{\text{vbl}(u) \mid u \in C\}$ and $\text{vbl}(F) = \bigcup_{C \in F} \text{vbl}(C)$.
Thus, \( vbl(F) \) is the set of variables occurring in \( F \). Finally, for a clause \( C \) we define \( \overline{C} := \{ \overline{u} \mid u \in C \} \). Please note that in general \( \overline{C} \) is not \( \neg C \).

A clause \( C \) of size \( k \) is called a \( k \)-clause. If all clauses in \( F \) have size \( k \) (at most \( k \)), then \( F \) is a \( k \)-CNF formula, (a \( (\leq k) \)-CNF formula, respectively).

SAT is the problem of deciding whether a given CNF formula is satisfiable, and \( k \)-SAT is the problem of deciding whether a given \( (\leq k) \)-CNF formula is satisfiable.

### Assignments

For a variable set \( V \), assignments are functions \( V \to \{0, 1\} \). We use Greek letters \( \alpha, \beta, \gamma \) for assignments. An assignment extends to negated variables via \( \alpha(\overline{x}) := \neg \alpha(x) \). An assignment \( \alpha \) satisfies a literal \( u \) if \( \alpha(u) = 1 \). It satisfies a clause if it satisfies a literal therein, and finally satisfies a formula if it satisfies all of its clauses. Note that an assignment need not be defined on all variables of a formula or a clause. For example, \( \alpha = [x \mapsto 1] \) satisfies the formula \( \{\{x, y\}, \{x, \overline{z}\}\} \). For a clause \( C \) and an assignment \( \alpha \) we write \( \alpha \models C \) as an abbreviation for “\( \alpha \) satisfies \( C \)”, similarly \( \alpha \not\models C \) if \( \alpha \) does not satisfy \( C \), and \( \alpha \models F \) for a formula \( F \). If \( \alpha \) is an assignment and \( u \) a literal, then \( \alpha[u \mapsto 1] \) is the assignment that agrees with \( \alpha \) on all variables besides \( vbl(u) \) and maps \( u \) to 1.

Formulas can be manipulated by substituting Boolean constants for variables. If \( \alpha \) is an assignment and \( F \) is a CNF formula, then \( F[^{\alpha}] \) denotes the CNF formula we obtain by doing just this: We remove every clause that evaluates to 1 under \( \alpha \) and in the remaining clauses remove all literals that evaluate to 0.

For a set of variables \( V \) and a CNF formula \( F \), we denote by \( \text{sat}_V(F) \) the set of all assignments \( \alpha : V \to \{0, 1\} \) that satisfy \( F \). Although we treat CNF formulas like combinatorial objects, one should not forget that they represent Boolean functions. If \( V \) is a variable set and \( F \) and \( G \) are CNF formulas with \( vbl(F) \subseteq V \) and \( vbl(G) \subseteq V \), we call \( F \) and \( G \) equivalent, if they describe the same Boolean function \( \{0, 1\}^V \to \{0, 1\} \). More precisely,

\[
F \equiv G :\iff \text{sat}_V(F) = \text{sat}_V(G) .
\]

By that definition, all unsatisfiable formulas are equivalent to \( \square \) and equivalent to each other. We write \( F \equiv_{\text{SAT}} G \) if either both \( F \) and \( G \) are satisfiable, or both are unsatisfiable. This notation is convenient but must not be confused with \( F \equiv G \).
Degrees and Neighborhoods

For a literal \( u \) and a formula \( F \), we can count the number of occurrences of \( u \) in \( F \). This is

\[
\text{occ}_F(u) := |\{C \in F \mid u \in C\}|.
\]

Note that a clause cannot contain both a variable \( x \) and its negation \( \bar{x} \). Therefore, \( \text{deg}_F(x) := \text{occ}_F(x) + \text{occ}_F(\bar{x}) \) is the number of clauses in \( F \) containing \( x \) or \( \bar{x} \), and we call it the degree \( x \) in \( F \). We define \( \text{deg}(F) := \max_{x \in \text{vbl}(F)} \text{deg}_F(x) \), the maximum degree of \( F \). The neighborhood of \( C \) in \( F \) is the set \( \Gamma_F(C) := \{D \in F \mid C \neq D, \text{vbl}(C) \cap \text{vbl}(D) \neq \emptyset\} \) and \( l(F) := \max_{C \in F} |\Gamma_F(C)| \). Two clauses \( C \) and \( D \) have a conflict if \( C \cap \bar{D} \neq \emptyset \). We define \( \Gamma'_F(C) := \{D \in F \mid C \cap \bar{D} \neq \emptyset\} \) and \( lC(F) := \max_{C \in F} |\Gamma'_F(C)| \).

The complete formula over a variable set \( V \) is the formula consisting of all \( 2^{|V|} \) \(|V|\)-clauses over \( V \). One checks that this formula is unsatisfiable, and every pair of clauses in \( F \) has a conflict. We denote the complete formula by \( CF(V) \).

Cubes and Balls

If \( F \) is a formula over the variables \( V \) with \( |V| = n \), then there are \( 2^n \) assignments \( \alpha : V \to \{0, 1\} \). Assuming an order on the variables in \( V \), every assignment is a bit string of length \( n \), i.e., an element of \( \{0, 1\}^n \). We call \( \{0, 1\}^n \) the Hamming cube of dimension \( n \). This is endowed with a metric, the Hamming distance

\[
d_H(\alpha, \beta) := |\{i \in \{1, \ldots, n\} \mid \alpha_i \neq \beta_i\}|.
\]

Every metric induces balls: For \( r \in \mathbb{N}_0 \) and \( \alpha \in \{0, 1\} \),

\[
B_r(\alpha) := \{\beta \in \{0, 1\}^n \mid d_H(\alpha, \beta) \leq r\}.
\]

The cardinality of such a ball – we call it the volume – depends only on its radius \( r \) and the dimension, not on its center \( \alpha \), and equals

\[
\text{vol}(n, r) = \sum_{j=0}^{r} \binom{n}{j}.
\]

Constraint Satisfaction Problems

Most of the above terminology generalizes the constraint satisfaction problems (CSP). Variables are not Boolean anymore, but can take on values from 1
to $d$. In analogy to graph coloring problems, these values are often called colors. We speak of $d$-ary variables. Literals are expressions of the form $(x \neq c)$. A constraint is a set of literals, interpreted as their disjunction ($\lor$), and a CSP formula is a set of constraints, interpreted as their conjunction ($\land$). We always assume that when a CSP formula is given, the value $d$ is given.

A constraint with $k$ literals is a $k$-constraint. A $(d, k)$-CSP formula is a CSP formula in which every variable can take on values from 1 to $d$ and all constraints have size $k$. A $(d, \leq k)$-CSP formula is defined analogously.

Let be a set $V$ of $d$-ary variables. An assignment $\alpha$ is a function $V \to \{1, \ldots, d\}$. We say $\alpha$ satisfies a literal $(x \neq c)$ if $\alpha(x) \neq c$. For a CSP formula $F$ and an assignment $\alpha$, the CSP formula $F^{[\alpha]}$ is defined analogously to the Boolean case.

$(d, k)$-CSP is the problem of deciding whether a given $(\leq d, k)$-CSP formula is satisfiable. Note that $(2, k)$-CSP is the same as $k$-SAT, just with a different syntax.
Part I

Algorithms for SAT and CSP
Chapter 3

Local Search Algorithms for SAT

Come, let us hasten to a higher plane,
Where dyads tread the fairy fields of Venn,
Their indices bedecked from one to \( n \),
Commingled in an endless Markov chain!

Stanislaw Lem, The Cyberiad

In this chapter we present a deterministic algorithm for \( k \)-SAT running in time \((2(k - 1)/k)^n + o(n)\) based on deterministic local search. This is joint work with Robin Moser [MS10]. The starting point of this chapter is Schöning’s random walk algorithm for \( k \)-SAT [Sch99], for which we will give a self-contained analysis. This algorithm, published in 1999, came as a surprise, since it is much faster and simpler than previous algorithms. Its running time is \( O((2(k - 1)/k)^n \text{poly}(n))\). For 3-SAT, this gives \( O(1.334^n)\). Not much afterwards, Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavavan and Schöning [DGH+02], henceforth abbreviated as Dantsin et al., gave a deterministic algorithm running in time \( O((2k/(k + 1))^n \text{poly}(n))\), which is \( O(1.5^n \text{poly}(n))\) for 3-SAT. Their result can be seen as an attempt to derandomize Schöning’s random walk. Although theirs is faster and conceptually simpler than previous deterministic algorithms, it falls short of achieving Schöning’s randomized running time. Over the years, several researchers have improved the deterministic running time for 3-SAT, but only recently has the gap between randomized and deterministic algorithms been closed. We

23
summarize the development of deterministic algorithms in the following table. The four middle rows do not have an entry for general \( k \) as those papers deal mainly with the case \( k = 3 \).

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>for ( k )-SAT</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1.334^n )</td>
<td>( \left( \frac{2(k-1)}{k} \right)^n )</td>
<td>Schöning [Sch99], randomized</td>
</tr>
<tr>
<td>( 1.5^n )</td>
<td>( \left( \frac{2k}{k+1} \right)^n )</td>
<td>Dantsin et al. [DGH+02]</td>
</tr>
<tr>
<td>( 1.481^n )</td>
<td>-</td>
<td>Dantsin et al. [DGH+02]</td>
</tr>
<tr>
<td>( 1.473^n )</td>
<td>-</td>
<td>Brueggemann and Kern [BK04]</td>
</tr>
<tr>
<td>( 1.465^n )</td>
<td>-</td>
<td>Scheder [Sch08]</td>
</tr>
<tr>
<td>( 1.439^n )</td>
<td>-</td>
<td>Kutzkov and Scheder [KS10]</td>
</tr>
<tr>
<td>( 1.334^n )</td>
<td>( \left( \frac{2(k-1)}{k} \right)^{n+o(n)} )</td>
<td>Moser and Scheder [MS10]</td>
</tr>
</tbody>
</table>

We explain Schöning’s random walk algorithm in Section 3.1 and the deterministic algorithm of Dantsin et al. in Section 3.2. Finally, in Section 3.3 we present the derandomization of Schöning’s algorithm by Moser and myself.

### 3.1 Schöning’s Algorithm

Schöning’s algorithm [Sch99] for \( k \)-SAT is extremely simple: It chooses a random initial assignment \( \alpha \) by setting every variables of \( F \) independently to 0 or 1, each with probability 1/2. Then it performs up to \( 3n \) local correction steps. In a local correction step, it chooses an arbitrary unsatisfied clause from \( F \) (if one exists), chooses a literal therein uniformly at random and locally modifies \( \alpha \) as to satisfy that literal, and thus \( C \). If within these \( 3n \) steps, it encounters a satisfying assignment, it returns it. Otherwise, it reports failure. The pseudocode listings below give a detailed description of the algorithm. The benefits of stating it as two separate algorithms will become clear soon.

**Algorithm 1** Schöning(\( F \): an \( (\leq k) \)-CNF formula)

1: \( \alpha \leftarrow \text{u.a.r.} \{0, 1\}^{\text{vbl}(F)} \) // sample \( \alpha \) uniformly at random
2: return Schöning-Walk(\( F, \alpha \))

**Theorem 3.1** (Schöning [Sch99]). If \( F \) is a satisfiable \( k \)-CNF formula on \( n \) variables and \( k \geq 3 \), then Schöning(\( F \)) returns a satisfying assignment with probability at least \( \left( \frac{2(k-1)}{k} \right)^{-n} \frac{1}{\text{poly}(n)} \).
Algorithm 2 Schöning–Walk($F$: an ($\leq k$)-CNF formula, $\alpha$: an assignment)

1: for $i = 0, \ldots, 3|\text{vbl}(F)|$ do
2: if $\alpha$ satisfies $F$ then
3: return $\alpha$
4: else
5: $C \leftarrow$ any clause of $F$ unsatisfied by $\alpha$
6: $u \leftarrow \text{u.a.r.} C$ \hspace{1cm} // a random literal from $C$
7: $\alpha \leftarrow \alpha[u \mapsto 1]$
8: end if
9: end for
10: return failure

The algorithm Schöning uses randomness in two ways: Once for choosing the initial assignment $\alpha$ in line 1 of Schöning, and then for steering the random walk, i.e., the at most $3n$ choices in line 6 of the algorithm Schöning–Walk. These two procedures are derandomized separately.

Lemma 3.2 (Schöning [Sch99]). Let $F$ be a satisfiable ($\leq k$)-CNF formula, $\alpha^*$ some fixed satisfying assignment of $F$, and $\alpha$ an arbitrary assignment. Let $r := d_H(\alpha, \alpha^*)$. Then Schöning–Walk($F, \alpha$) returns some satisfying assignment with probability at least $(k - 1)^{-r/poly(r)}$.

Note that the satisfying assignment which the algorithm returns can be different from $\alpha^*$. Theorem 3.1 follows easily from the lemma.

Proof of Theorem 3.1 Suppose $F$ is a satisfiable formula and let $\alpha^*$ be any satisfying assignment. We say that Schöning is successful if it returns some satisfying assignment, possibly different from $\alpha^*$. We want to show that

$$\text{Pr}[\text{Schöning}(F) \text{ successful}] \geq \left(\frac{2(k - 1)}{2k}\right)^{-n} \frac{1}{\text{poly}(n)}.$$ 

Consider the initial assignment $\alpha$ that Schöning chooses uniformly at ran-
dom from \( \{0, 1\}^n \).

\[
\Pr[\text{Schöning}(F) \text{ successful}]
\]
\[
= 2^{-n} \sum_{\alpha \in \{0, 1\}^n} \Pr[\text{Schöning-Walk}(F, \alpha) \text{ successful}]
\]
\[
\geq 2^{-n} \sum_{\alpha \in \{0, 1\}^n} (k - 1)^{-d_H(\alpha, \alpha^*)}/\text{poly}(n)
\]
\[
= \frac{1}{2^n \text{poly}(n)} \sum_{r=0}^{n} \binom{n}{r} (k - 1)^r
\]
\[
= \left( \frac{2(k - 1)}{k} \right)^n \frac{1}{\text{poly}(n)}. 
\]

This completes the proof of Theorem 3.1. \( \square \)

It remains to prove Lemma 3.2, which we will do in the remainder of this section. Let us outline the proof. We analyze the evolution of \( d_H(\alpha, \alpha^*) \), the distance between the current assignment and some fixed satisfying assignment \( \alpha^* \). We show that this can be modeled as a random walk on the integers that starts at \( r \) and in every step moves left with probability \( 1/k \) and right with probability \( 1 - 1/k \). The goal then is to compute the probability that this random walk reaches 0 within \( 3n \) steps. We bound this by computing the probability that the random walk is in state 0 after \( \lceil kr/(k - 2) \rceil \) steps. This probability can easily be computed using bounds on the binomial coefficient.

Proof of Lemma 3.2 Let \( F, \alpha, \) and \( \alpha^* \) be as in the statement of the lemma, let \( r := d_H(\alpha, \alpha^*) \), and let \( n \) denote the number of variables in \( F \). We first give a proof sketch. We examine how the Hamming distance \( d_H(\alpha, \alpha^*) \) evolves. In every iteration of Schöning-Walk, this distance increases or decreases by exactly 1. There is a probability of least \( 1/k \) that it decreases by 1. We model this by a random walk on the integers that starts at \( r \) and in every step decreases with probability exactly \( 1/k \). We then estimate the probability that this random walk reaches 0 within \( 3n \) steps.

Now let us start with the formal proof. Let \( T \) denote the number of iterations performed by Schöning-Walk. This is a random variable, but certainly it holds that \( 0 \leq T \leq 3n \). Let \( \alpha_0 := \alpha \) and for \( 1 \leq i \leq T \) let \( \alpha_i \) denote the assignment \( \alpha \) right after the \( i \)th iteration of Schöning-Walk. For all integers \( i \geq 1 \) we define random numbers \( D_i, D'_i \in \{-1, +1\} \) as follows: If \( i > T \), we set \( D_i = D'_i = -1 \) with probability \( 1/k \) and \( D_i = D'_i = +1 \) with probability
1 - 1/k, independently. For 1 \leq i \leq T, let C be the clause Schöning-Walk picks in the \(i^{th}\) iteration and let \(u \in C\) be the literal it picks from \(C\). We set \(D_i\) to \(-1\) if \(\alpha^*(u) = 1\), and to \(+1\) otherwise. Note that \(\Pr[D_i = -1] \geq 1/k\), since \(\alpha^*\) satisfies \(C\), therefore \(C\) contains at least one literal \(u\) such that \(\alpha^*(u) = 1\).

The definition of \(D'_i\) is more subtle: We want to define \(D'_i\) in such a way that (i) \(D'_i \geq D_i\) and (ii) \(D'_i\) is \(-1\) with probability exactly \(1/k\). This is done by setting \(D'_i := \max(D_i, Z_i)\), where \(Z_i\) is chosen independently and randomly from \(\{-1, +1\}\) such that \(\Pr[Z_i = -1] = 1/(k \Pr[D_i = -1])\). Clearly, \(D'_i \geq D_i\) holds, thus (i) is satisfied. We calculate \(\Pr[D'_i = -1] = \Pr[D_i = Z_i = -1] = \Pr[D_i = -1]/(k \Pr[D_i = -1]) = 1/k\). Therefore (ii) holds.

We observe that for \(0 \leq i \leq T\) it holds that \(d_H(\alpha_i, \alpha^*) = r + D_1 + D_2 + \cdots + D_i\). We define \(Y_0 := r\) and \(Y_i := r + D'_1 + D'_2 + \cdots + D'_i\) for \(i \geq 1\).

By the properties of the \(D'_i\), the variables \(Y_i\) form a random walk on the integers. Note that for \(0 \leq i \leq T\) it holds that \(d_H(\alpha_i, \alpha^*) \leq Y_i\). Suppose that \(Y_i = 0\) for some \(0 \leq i \leq 3n\). There are two cases: Either \(i \leq T\), in which case \(d_H(\alpha_i, \alpha^*) \leq Y_i \leq 0\). This implies that \(d_H(\alpha_i, \alpha^*) = 0\), thus \(\alpha_i = \alpha^*\), and Schöning-Walk returns a satisfying assignment. Or, \(T < i \leq 3n\), which means that Schöning-Walk stops before completing \(3n\) iterations. This means that it returns a satisfying assignment. We conclude that

\[
\Pr[\text{Schöning-Walk}(F, \alpha) \text{ successful}] \geq \Pr[\exists 0 \leq i \leq 3n : Y_i = 0].
\]

There are two reasons why this is an inequality rather than an equality: First, if \(F\) has multiple satisfying assignments, it could be that for some \(0 \leq i \leq 3n\), the assignment \(\alpha_i\) satisfies \(\alpha_i\), although \(\alpha_i \neq \alpha^*\). In this case, the algorithm returns a satisfying assignment after \(i\) iterations, but \(Y_i\) can still be large. Second, it could happen that the clause \(C\) picked by Schöning-Walk contains several literals \(u\) for which \(\alpha^*(u) = 1\) or \(|C| < k\). In this case, \(D_i\) is \(-1\) with probability greater than \(1/k\).

What is the probability that \(Y_i = 0\) for some \(0 \leq i \leq 3n\)? If we have an answer to this question, we have a lower bound for the success probability of Schöning-Walk. Let \(t := \lceil kr/(k - 2) \rceil\). The probability that \(Y_i = 0\) for some \(0 \leq i \leq 3n\) is at least \(\Pr[Y_t = 0]\).

**Lemma 3.3.** Suppose \(Y_0 = r\) and set \(t = \lceil kr/(k - 2) \rceil\). Then \(\Pr[Y_t = 0] \geq (k - 1)^{-r}/\text{poly}(r)\).

**Proof.** For the sake of readability we will drop the ceiling notation. Since we are ready to give away polynomial factors in \(r\) anyway, this will cause no problems. If during the \(t\) steps \(Y_i\) decreases \(a := \frac{k-1}{k-2} r\) times and increases \(b := \frac{1}{k-2} r\) times, then and only then \(Y_t = 0\). One checks that \(a + b = t\) and
r + b - a = 0. The incrementing and decrementing steps can of course be mixed, in a total of \( \binom{t}{a} \) ways. Thus, we conclude that

\[
\Pr[Y_t = 0] = \left( \frac{1}{k} \right)^a \left( \frac{k - 1}{k} \right)^b \binom{t}{a}.
\]

We will use the following approximation of the binomial coefficient.

**Lemma 3.4** (MacWilliams, Sloane [MS77], Chapter 10, Corollary 9). For \( 0 \leq \rho \leq 1/2 \) and \( t \in \mathbb{N} \), it holds that

\[
\binom{t}{\rho t} \geq \frac{1}{\sqrt{8t\rho(1-\rho)}} \left( \frac{1}{\rho} \right)^{\rho t} \left( \frac{1}{1-\rho} \right)^{(1-\rho)t}.
\]

Often, one abbreviates this expression by writing \( \left( \frac{1}{\rho} \right)^{\rho t} \left( \frac{1}{1-\rho} \right)^{(1-\rho)t} = 2^{H(\rho)t} \), where \( H(\rho) = -\rho \log_2 \rho - (1 - \rho) \log_2(1 - \rho) \) is the binary entropy function. The rest of the proof of Lemma 3.3 is just a calculation:

\[
\Pr[Y_t = 0] = \left( \frac{1}{k} \right)^a \left( \frac{k - 1}{k} \right)^b \binom{t}{a} \geq \left( \frac{1}{k} \right)^a \left( \frac{k - 1}{k} \right)^b \binom{t}{a} \left( \frac{t}{b} \right)^b \frac{1}{\text{poly}(t)} \geq \left( \frac{1}{k} \right)^a \left( \frac{k - 1}{k} \right)^b \left( \frac{k}{k - 1} \right)^a \frac{1}{\text{poly}(t)} = \frac{(k - 1)^{b-a}}{\text{poly}(t)} = \frac{(k - 1)^{-r}}{\text{poly}(t)}.
\]

We conclude that \( \Pr[Y_t = 0] \geq (k - 1)^{-r}/\text{poly}(r) \). □

We have shown that \( \Pr[Y_t = 0] \geq (k - 1)^r/\text{poly}(r) \), where \((Y_i)_{i \in \mathbb{N}_0}\) is the random walk on the integers described above, \( r = Y_0 \), and \( t = \lceil kr/(k-2) \rceil \). Let us finish the proof of Lemma 3.2. Above, we have seen that \( X_i \leq Y_i \) if Schöning-Walk **runs for at least** \( i \) steps. Since \( X_i = 0 \) implies that Schöning-Walk returns a satisfying assignment, we conclude that this happens with probability at least \( (k - 1)^r/\text{poly}(r) \). □

Let us reflect on the term \( (k - 1)^{-r} \) for some seconds. Note that with probability \( k^{-r} \), the random walk \( Y_i \) makes \( r \) steps to the left and reaches 0 within
3.2 The Algorithm by Dantsin et al.

As mentioned above, Schöning’s algorithm uses randomness in two ways: First to choose an initial assignment, then to steer a random walk. In 2002, Dantsin et al. used a deterministic construction of covering codes to derandomize the first step, losing only a polynomial factor in $n$ in the running time. For the second step, the random walk, they gave a simple recursive procedure that unfortunately falls short of achieving Schöning’s randomized running time. In this section we explain their approach in detail.

**Theorem 3.5** (Dantsin et al. [DGH+02]). There is a deterministic algorithm, called cover-search, solving $k$-SAT in time $(2k/(k+1))^n\text{poly}(n)$.

Note that $k/(k+1) > (k-1)/k$, therefore the running time of cover-search is larger than the expected running time of Schöning. For example, for 3-SAT Schöning has an expected running time of $O(1.334^n)$, whereas cover-search achieves $O(1.5^n\text{poly}(n))$. For large values of $k$, the difference diminishes, but this is not surprising, as both running times converge to the trivial upper bound of $O(2^n)$. To understand the approach of Dantsin et al., we introduce an auxiliary problem which we call BALL-$k$-SAT.

**BALL-$k$-SAT**: Given a $(d,\leq k)$-CNF formula $F$ over $n$ variables, a assignment $\alpha$ to these variables, and a natural number $r$. Decide whether $B_r(\alpha)$ contains a satisfying assignment.

**Proposition 3.6** (Dantsin et al. [DGH+02]). The algorithm sat-searchball solves BALL-$k$-SAT in time $O(k^r\text{poly}(n))$

**Proof.** The running time is easy to analyze: If $F$ is a $(\leq k)$-CNF formula, then each call to sat-searchball causes at most $k$ recursive calls. To see correctness of the algorithm, we proceed by induction on $r$. If $r = 0$, then $B_0(\alpha) = \{\alpha\}$, and the algorithm returns true if and only $\alpha$ satisfies $F$. For
Algorithm 3 sat-searchball($k$-CNF formula $F$, assignment $\alpha$, radius $r$)

1: if $\alpha$ satisfies $F$ then
2: return true
3: else if $r = 0$ then
4: return false
5: else
6: $C \leftarrow$ any clause of $F$ unsatisfied by $\alpha$
7: return $\bigvee_{u \in C} \text{sat-searchball}(F[u=1], \alpha, r-1)$
8: end if

the induction step, first consider the case that there is some $\alpha^*$ that satisfies $F$ and $d_H(\alpha, \alpha^*) \leq r$. Let $C$ be the clause selected in line 6. Since $\alpha^*$ satisfies $C$ but $\alpha$ does not, there is at least one literal $u \in C$ such that $\alpha^*(u) = 1$ and $\alpha(u) = 0$. Let $\alpha' := \alpha^*[u := 0]$. We observe that $d(\alpha, \alpha') \leq r-1$ and $\alpha'$ satisfies $F[u=1]$ (although not necessarily $F$). Therefore the induction hypothesis ensures that the recursive call to sat-searchball($F[u=1], \alpha, r-1$) returns true. On the other hand, if $B_r(\alpha)$ contains no satisfying assignment of $F$, then $B_{r-1}(\alpha)$ does not contain any satisfying assignment for $F[u=1]$. Therefore, sat-searchball returns false in this case.

One can view sat-searchball as a deterministic version of the randomized algorithm Schöning-Walk, albeit having a worse running time. How do Dantsin et al. derandomize Schöning, i.e., the choice of the initial assignment?

Covering Codes

Definition 3.7. Let $C \subseteq \{0, 1\}^n$ and $r \in \mathbb{N}_0$. If

$$\bigcup_{\alpha \in C} B_\alpha(r) = \{0, 1\}^n,$$ (3.1)

then we call $C$ a covering code of radius $r$ and length $n$.

If $C$ is a covering code of radius $r$ and length $n$, then we can decide satisfiability of a $k$-CNF formula $F$ over $n$ variables by calling sat-searchball($F, \alpha, r$) for each $\alpha \in C$. This takes time

$$O(|C|k^r \text{poly}(n)).$$ (3.2)
Note that the volume \( \text{vol}(n, r) := |B_\alpha(r)| = \sum_{i=0}^{r} \binom{n}{i} \) is independent of \( \alpha \). By a volume argument, it is immediately clear that \( |C| \geq 2^n / \text{vol}(n, r) \) for any code \( C \) of length \( n \) and covering radius \( r \).

**Lemma 3.8 ([DGH+02])**. For all \( n \in \mathbb{N} \), \( 0 \leq r \leq n \), every code \( C \) of covering radius \( r \) and length \( n \) has at least \( \frac{2^n}{\text{vol}(n, r)} \) elements. Furthermore, there is such a \( C \) with

\[
|C| \leq \frac{2^n \text{poly}(n)}{\text{vol}(n, r)},
\]

and furthermore, \( C \) can be constructed deterministically in time \( |C| \text{poly}(n) \).

We will encounter several covering code constructions in this chapter. We will therefore give a detailed proof of Lemma 3.8 in Chapter 5. Continuing with the proof of Theorem 3.5, observe that the lemma implies that we can solve \( k \)-SAT in

\[
O \left( \frac{2^n k^r \text{poly}(n)}{\text{vol}(n, r)} \right)
\]

steps, by calling \( \text{sat-searchball}(F, \alpha, r) \) for each \( \alpha \in C \). All that remains now is to estimate \( \binom{n}{r} \) and find the optimal radius \( r \).

**Theorem 3.9** (Dantsin et al. [DGH+02]). Suppose algorithm \( A \) solves \( \text{BALL-} k \)-SAT in \( O(a^r \text{poly}(n)) \) steps. Then there is an algorithm \( B \) solving \( k \)-SAT in time \( O((2a/(a+1))^n \text{poly}(n)) \), and \( B \) is deterministic if \( A \) is.

**Proof**. Set \( r := n/(a+1) \) and construct a covering code \( C \) of radius \( r \) and length \( n \) and call \( A(F, \alpha, r) \) for each \( \alpha \in C \). To estimate the running time, we use the lower bound on \( \text{vol}(n, r) \) given in Lemma 3.4. The total running time is at most

\[
|C|a^r \text{poly}(n) \leq \frac{2^n a^r \text{poly}(n)}{\text{vol}(n, r)} \leq \frac{2^n a^{n/(a+1)} \text{poly}(n)}{(a+1)^{n/(a+1)} (a+1)^{na/(a+1)}} = \left( \frac{2a}{a+1} \right)^n \text{poly}(n).
\]

A more detailed calculation (using elementary calculus for example) also shows that the choice \( r = n/(a + 1) \) is indeed optimal. This completes the proof. \( \square \)
Algorithm 4 cover-search((≤k)-CNF formula F over n variables)

1: \( r := \frac{n}{(k+1)} \).
2: construct a covering code \( C \) of radius \( r \) and \( |C| \leq \frac{2^n}{n^{poly(n)}} \).
3: return \( \bigvee_{\alpha \in C} \text{sat-searchball}(F, \alpha, r) \)

Since sat-searchball solves BALL-k-SAT in time \( O(k^r \text{poly}(n)) \), it also solves PROMISE-BALL-k-SAT in time \( O(k^r \text{poly}(n)) \), and therefore we can solve k-SAT in time \( O((2k/(k+1))^{n \text{poly}(n)}) \). This proves Theorem 3.5. We summarize this in the algorithm cover-search.

Note that the running time of cover-search is much larger than that of Schönig, which is \( O((2(k-1)/k)^n \text{poly}(n)) \). In this sense, cover-search is not a complete derandomization of Schönig. In the next section we will close this gap by presenting an improved deterministic algorithm.

3.3 A Complete Derandomization of Schönig’s Algorithm

In this section we will present a complete derandomization of Schönig’s algorithm. This is joint work with Robin Moser [MS10]. One part of the derandomization of Schönig’s algorithm has already been solved by Dantsin et al. [DGH+02] via covering code constructions. In our terminology, they have completely (i.e., with only subexponential loss in the running time) derandomized the algorithm Schönig we defined above. However, their algorithm sat-searchball, which is the deterministic analog to Schönig-Walk, runs in time \( O(k^r \text{poly}(n)) \), thus falling short of achieving the running time of the Monte Carlo version of Schönig-Walk, which is \( O((k-1)^r \text{poly}(n)) \).

We present an algorithm searchball-fast that does not solve BALL-k-SAT but a promise version of it, which we call PROMISE-BALL-k-SAT:

**PROMISE-BALL-k-SAT.** Given a \((\leq k)\)-CNF formula \( F \), an assignment \( \alpha \) and a radius \( r \), PROMISE-BALL-k-SAT is the following promise problem: If \( B_r(\alpha) \) contains a satisfying assignment for \( F \), answer \text{true}; if \( F \) is unsatisfiable, answer \text{false}; otherwise, i.e., if \( F \) is satisfiable but \( B_r(\alpha) \) contains no satisfying assignment, answer \text{true} or \text{false}, arbitrarily.

Note that every algorithm solving BALL-k-SAT solves PROMISE-BALL-k-SAT, but the converse does not necessarily hold. In particular, it is easy to see that
3.3. A Complete Derandomization of Schöning’s Algorithm

Promise-Ball-2-SAT can be solved in polynomial time, simply by deciding whether the given ($\leq 2$)-CNF formula is satisfiable, whereas Ball-2-SAT is NP-complete, as a simple reduction from Vertex Cover shows. The main result of this chapter is the following theorem:

**Theorem 3.10.** There is a deterministic algorithm solving Promise-Ball-$k$-SAT in time $O((k - 1)^{r + o(r)} \text{poly}(n))$.

In Theorem 3.9, we showed how an algorithm for Ball-$k$-SAT translates to an algorithm for $k$-SAT. In fact, exactly the same proof shows that this works also if the given algorithm does not solve Ball-$k$-SAT but only Promise-Ball-$k$-SAT. Thus, Theorem 3.9 and Theorem 3.10 together yield the following result.

**Theorem 3.11.** There is a deterministic algorithm solving Promise-Ball-$k$-SAT in time $2^{(k - 1)/k}n^{o(n)}$.

**Proof Sketch of Theorem 3.10.** Given $F$, $\alpha$, and $r$, our new algorithm tries to collect a maximal set of pairwise independent unsatisfied clauses. This yields a subformula $G$ with $m$ clauses and $km$ variables. If $m$ is small, we can iterate over all $2^{km}$ assignments $\beta$ to those variables and consider $F[\beta]$. By maximality of $G$, all unsatisfied clauses of $F[\beta]$ are of size at most $k - 1$. For such a formula, Promise-Ball-$k$-SAT and even Ball-$k$-SAT can easily be solved in time $O((k - 1)^{r} \text{poly}(n))$.

The interesting case is when $m$ is very large. In this case, we consider the set $S$ of assignments $\beta \in \{0, 1\}^{\text{vbl}(G)}$ that satisfy exactly one literal in each clause in $G$. We construct a covering code for $S$ of some specific radius and recurse for each element of this code.

**Proof of Theorem 3.10.** Let us reconsider sat-searchball on page 30. We have seen above that sat-searchball solves Ball-$k$-SAT (and thus also solves Promise-Ball-$k$-SAT) and runs in time $O(k^r \text{poly}(n))$. Suppose now in line 6, the algorithm selects an unsatisfied clause $C \in F$ of size $k - 1$. In this case, the algorithm will call itself $k - 1$ times recursively. Suppose further that $F$ is such that all clauses currently unsatisfied by $\alpha$ have size at most $k - 1$. The algorithm recurses on formulas of the form $F[u \mapsto 1]$. Clearly, this formula can contain new unsatisfied clauses: Setting $u \mapsto 1$ can make a clause unsatisfied if $\bar{u}$ was the only satisfied literal in that clause. However, once a clause has lost one literal, its size is at most $k - 1$. Hence every formula $F[u \mapsto 1]$ also has the property that all unsatisfied clauses have at most $k - 1$ literals. We see that this property propagates throughout the whole recursion tree, and obtain the following result:
Proposition 3.12. Suppose $F$ is an $(\leq k)$-CNF formula, $\alpha$ an assignment to its variables, and $r \in \mathbb{N}$. If every clause in $F$ that is not satisfied by $\alpha$ has size at most $k - 1$, then $\text{sat-searchball}(F, \alpha, r)$ runs in time $O((k - 1)^r \text{poly}(n))$.

$k$-ary Covering Codes

We generalize the concept of covering codes introduced in Chapter 3 to alphabets of size greater than 2. In this section, we consider the $k$-ary Hamming cube $\{1, \ldots, k\}^t$. It is endowed with a metric, namely the Hamming distance $d_H$: For two elements $w, w' \in \{1, \ldots, k\}^t$, the distance $d_H(w, w')$ is the number of coordinates in which $w$ and $w'$ do not agree. We define balls:

$$B_s^{(k)}(w) := \{w' \in \{1, \ldots, k\}^t \mid d_H(w, w') \leq s\}.$$

Let us calculate the volume of such a ball. There are $\binom{t}{s}$ possibilities to choose the set of coordinates in which $w$ and $w'$ are supposed to differ, and for each such coordinate, there are $k - 1$ ways in which they can differ. Therefore,

$$\text{vol}^{(k)}(t, s) := |B_s^{(k)}(w)| = \sum_{i=0}^{s} \binom{t}{i} (k - 1)^i.$$

We are interested in the question of how many balls $B_s^{(k)}(w)$ we need to cover all of $\{1, \ldots, k\}^t$. Note that by symmetry, $w \in B_s^{(k)}(v)$ iff $v \in B_s^{(k)}(w)$, for all $v, w \in \{1, \ldots, k\}^t$.

Definition 3.13. Let $t \in \mathbb{N}$. A set $C \subseteq \{1, \ldots, k\}^t$ is called a code of covering radius $s$ and length $t$ if

$$\bigcup_{w \in C} B_s^{(k)}(w) = \{1, \ldots, k\}^t.$$

In other words, for each $w' \in \{1, \ldots, k\}^t$, there is some $w \in C$ such that $d_H(w, w') \leq s$.

The following lemma is an adaptation of Lemma 3.8.

Lemma 3.14. For all $t, k \in \mathbb{N}$ and $0 \leq s \leq t/2$, there exists a code $C \subseteq \{1, \ldots, k\}^t$ of covering radius $s$ such that

$$|C| \leq \left\lceil \frac{t \ln(k)k^t}{\binom{t}{s}(k - 1)^s} \right\rceil$$

and such a code $C$ can be constructed deterministically in time $O(|C|\text{poly}(t))$. 
3.3. A Complete Derandomization of Schöning’s Algorithm

Note that this bound is optimal up to a polynomial factor in \( t \ln(k) \), since every Hamming ball of radius \( r \leq t/2 \) has at most \( t^t(k-1)^6 \) elements. We prove the lemma in Chapter 5.

A Deterministic Algorithm for PROMISE-BALL-\( k \)-SAT

We will now describe our deterministic algorithm. If the input formula \( F \) is unsatisfiable, it will never return \text{true}. So let us assume that \( B_r(\alpha) \) contains some satisfying assignment and let \( \alpha^* \) be one of them. We fix \( \alpha^* \) for the rest of the proof. Define \( t := \lfloor \ln n \rfloor \). Compute a code \( C \subseteq \{1, \ldots, k\}^t \) of covering radius \( s := t/k \) according to Lemma 3.14. The resulting code has size at most \( \lceil t \ln(k)k^t \rceil \) and can be constructed deterministically in time \( O(|C|\text{poly}(t)) \leq O(k^t\text{poly}(t)) = O(n^{t_k} \text{poly}(\ln n)) = \text{poly}(n) \), since \( k \) is a constant. We estimate its size using Lemma 3.14 and the lower bound for the binomial coefficient in Lemma 3.4, which reads like

\[
\binom{t}{pt} \geq \frac{1}{\sqrt{8tp(1-\rho)}} \left( \frac{1}{\rho} \right)^pt \left( \frac{1}{1-\rho} \right)^{(1-\rho)t}.
\]

We apply this bound with \( \rho = 1/k \):

\[
\binom{t}{t/k} \geq \frac{1}{\sqrt{8t}} k^t/k \left( \frac{k}{k-1} \right)^{(k-1)t/k} = \frac{k^t}{\sqrt{8t}(k-1)^{(k-1)t/k}}.
\]

Together with Lemma 3.14, we obtain, for sufficiently large \( t \):

\[
|C| \leq \left\lceil \frac{t \ln(k)k^t}{(t/k)(k-1)^t/k} \right\rceil \leq t^2 k^t(k-1)^{(k-1)t/k}/k^t(k-1)^t/k \leq t^2(k-1)^{t-2t/k}.
\]

The algorithm computes this code and stores it for further use. Given \( F, \alpha \) and \( r \), it first greedily constructs a maximal set \( G \) of pairwise disjoint unsatisfied \( k \)-clauses of \( F \). That is, \( G = \{C_1, C_2, \ldots, C_m\} \), the \( C_i \)'s are pairwise disjoint, no \( C_i \) in \( G \) is satisfied by \( \alpha \), and each unsatisfied \( k \)-clause \( D \) in \( F \) shares at least one literal with some \( C_i \). Note that since each \( C_i \) is unsatisfied by \( \alpha \), it not only holds that \( C_i \cap C_j = \emptyset \) for \( 1 \leq i, j \leq m \), but also \( \text{vbl}(C_i) \cap \text{vbl}(C_j) = \emptyset \).

At this point, the algorithm considers two cases. First, if \( m < t \), it enumerates all \( 2^{km} \) assignments to the variables in \( G \). For each such assignment \( \beta \), it calls \( \text{sat-searchball}(F[\beta], \alpha, r) \) and returns \text{true} if at least one such call returns \text{true}. Correctness is easy to see: At least one \( \beta \) agrees with the
satisfying assignment $\alpha^*$, and therefore $\alpha^*$ still satisfies $F^{[\beta]}$. To analyze the running time, observe that for any such $\beta$, the formula $F^{[\beta]}$ contains no unsatisfied clause of size $k$. This follows from the maximality of $G$. Therefore, Proposition 3.12 tells us that $\text{sat-searchball}(F^{[\beta]},\alpha,r)$ runs in time $O((k-1)^r\text{poly}(n))$, and therefore this case takes time $2^{kr}O((k-1)^r\text{poly}(n)) \leq O(2^{kt}(k-1)^r\text{poly}(n)) \leq O((k-1)^r\text{poly}(n))$, since $k$ is a constant and $m \leq t = \lceil \ln n \rceil$.

The second case is more interesting: If $m \geq t$, the algorithm chooses $t$ clauses from $G$ to form $H = \{C_1,\ldots,C_t\}$, a set of pairwise disjoint $k$-clauses, all unsatisfied by $\alpha$. For $w \in \{1,\ldots,k\}^t$, let $\alpha[H,w]$ be the assignment obtained from $\alpha$ by flipping the value of the $w_i$th literal in $C_i$, for $1 \leq i \leq t$. We assume there is some ordering on $H$ as well as on the literals in each $C_i$. Note that $\alpha[H,w]$ satisfies exactly one literal in each $C_i$, for $1 \leq i \leq t$. For convenience, we will write $\alpha[w]$ instead of $\alpha[H,w]$ if $H$ is understood.

**Example.** Let $\alpha = (0,\ldots,0)$, $t = 3$ and $H = \{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \{x_3, y_3, z_3\}\}$. Let $w = (2,3,3)$. Then $\alpha[w]$ is the assignment that sets $y_1, z_2$, and $z_3$ to 1 and all other variables to 0.

Consider the satisfying assignment $\alpha^*$ with $d_H(\alpha,\alpha^*) \leq r$. We define $w^* \in \{1,\ldots,k\}^t$ as follows: For each $1 \leq i \leq t$, we set $w^*_i$ to $j$ such that $\alpha^*$ satisfies the $j$th literal in $C_i$. Since $\alpha^*$ satisfies at least one literal in each $C_i$, we can do this, but since $\alpha^*$ possibly satisfies multiple literals in $C_i$, the choice of $w^*$ is not unique.

**Observation 3.15.** The following facts about $\alpha[w]$ hold:

- $d_H(\alpha,\alpha[w]) = t$ for every $w \in \{1,\ldots,k\}^t$.
- There is some $w^* \in \{1,\ldots,k\}^t$ such that $d_H(\alpha[w^*],\alpha^*) = d_H(\alpha,\alpha^*) - t$.
- Let $w, w' \in \{1,\ldots,k\}^t$. Then $d_H(\alpha[w],\alpha[w']) = 2d_H(w,w')$.

We could now call $\text{sat-searchball}(F,\alpha[w],r-t)$ for each $w \in \{1,\ldots,k\}^t$. This would yield a running time of $O(k^r\text{poly}(n))$, i.e., no improvement over Dantsin et al. Instead, we iterate over all $w \in C$.

**Lemma 3.16.** Let $t$ and $H$ be defined as above, and let $C \subseteq \{1,\ldots,k\}^t$ be a $k$-ary code of length $t$ and covering radius $s$. If $\alpha^*$ is a satisfying assignment of $F$, then there is some $w \in C$ such that $d_H(\alpha[w],\alpha^*) \leq d_H(\alpha,\alpha^*) - t + 2s$.

In particular, if $B_r(\alpha)$ contains a satisfying assignment, then there is some $w \in C$ such that $B_{r-t+2s}(\alpha[w])$ contains it, too.
3.3. A Complete Derandomization of Schöning’s Algorithm


\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.1.png}
\caption{Illustration of Lemma 3.16. The distance from $\alpha[w]$ to $\alpha^*$ is at most the distance from $\alpha[w^*]$ to $\alpha^*$ plus $2s$.}
\end{figure}

Proof. Proof of Lemma 3.16. By Observation 3.15, there is some $w^* \in \{1, \ldots, k\}^t$ such that $d_H(\alpha[w^*], \alpha^*) = d_H(\alpha, \alpha^*) - t \leq r - t$. Since $C$ has covering radius $s$, there is some $w \in C$ such that $d_H(w, w^*) \leq s$, and by Observation 3.15, $d_H(\alpha[w], \alpha[w^*]) \leq 2s$. The lemma now follows from the triangle inequality.

Recall that $s = t/k$, therefore $r - t + 2s = r - (t - 2t/k)$. We write $\Delta := (t - 2t/k)$. Our algorithm calls itself recursively with $\alpha[w]$ and $r - \Delta$ for each $w \in C$. By Lemma 3.16, there is some $w \in C$ such that $B_{r-t+2s}(\alpha[w])$ contains $\alpha^*$, and therefore at least one call will be successful. Let us analyze the running time: We cause $|C|$ recursive calls and decrease the complexity parameter $r$ by $\Delta$ in each step. This is good, since $|C|$ is only slightly bigger than $(k - 1)^\Delta$. We conclude that the number of leaves in this recursion tree is at most

$$|C|^{r/\Delta} \leq (t^2(k - 1)^{\Delta}/(k - 1) t^{2/\Delta})^r.$$ 

Since $t^2/\Delta$ goes to 1 as $t$ grows, the above term is at most $(k - 1)^{r+o(n)}$. Finally, if $F$ is unsatisfiable, the algorithm clearly returns false. If $F$ is satisfiable, but $B_r(\alpha)$ contains no satisfying assignment, our algorithm may return true or false. In any case, its running time is at most $(k - 1)^{r+o(n)}$. This proves Theorem 3.10.

We summarize the whole algorithm in searchball-fast.

Hindsight: Why and How?

At this point, I want to give some intuition on why searchball-fast is faster than sat-searchball and what the underlying differences between them are.
Algorithm 5 searchball-fast\((k \in \mathbb{N}, (\leq k)\text{-CNF formula } F, \text{ assignment } \alpha, \text{ radius } r, \text{ code } C \subseteq \{1, \ldots, k\}^t)\)

1: if \(\alpha\) satisfies \(F\) then
2: return true
3: else if \(r = 0\) then
4: return false
5: else
6: \(G \leftarrow\) a maximal set of pairwise disjoint \(k\)-clauses of \(F\) unsatisfied by \(\alpha\)
7: if \(|G| < t\) then
8: return \(\bigvee_{\beta \in \{0,1\}^{v_{bl}(G)}}\) sat-searchball\((F[\beta], \alpha, r)\)
9: else
10: \(H \leftarrow \{C_1, \ldots, C_t\} \subseteq G\)
11: return \(\bigvee_{w \in C}\) searchball-fast\((k, F, \alpha[H, w], r - (t - 2t/k), C)\)
12: end if
13: end if

An Easier Problem? The first reason is that searchball-fast solves the problem PROMISE-BALL-\(k\)-SAT, which is simpler than BALL-\(k\)-SAT, the problem which sat-searchball solves. It could well happen that \(F\) is satisfiable, \(B_r(\alpha)\) contains no satisfying assignment, yet nevertheless the call to searchball-fast\((F, \alpha, r)\) returns some satisfying assignment outside the ball. The same can happen to Schöning-Walk.

We suspect that BALL-\(k\)-SAT is really more difficult than PROMISE-BALL-\(k\)-SAT. This is supported by the case \(k = 2\), for which PROMISE-BALL-2-SAT can be solved in polynomial time by simply solving the given \((\leq 2)\)-CNF formula, whereas BALL-2-SAT is an NP-complete problem, as can easily be seen by a reduction from VERTEX COVER.

Allowing Mistakes

In Section 3.1 we analyzed Schöning-Walk by fixing some satisfying assignment \(\alpha^*\) and modeling the evolution of \(d_H(\alpha, \alpha^*)\) as a Markov chain. At any given step of Schöning-Walk let us say that it makes a mistake if this distance increases by 1.

The probability that Schöning-Walk returns a satisfying assignment is at least \((k - 1)^{-r}/\text{poly}(r)\), where \(r\) is the initial distance \(d_H(\alpha, \alpha^*)\). However, if we insist on Schöning-Walk being successful without making any mistakes, this probability drops to \(k^{-r}\): In the worst case, there is exactly one correct choice in each step! It is much more realistic to reach the goal after having
made (and corrected) some mistakes on the way than to reach the goal without any mistakes (that’s a lesson for life). Note that searchball-fast allows mistakes: It performs $t$ correction steps at once, and within this window, it tolerates up to $t/k$ mistakes.

**Correction Strings**

Imagine Schöning-Walk is called with two additional parameter $t$ and $w$: Schöning-Walk($F,t,\alpha,w$), where $t$ is the number of corrections steps to be made, and $w \in \{1,\ldots,k\}^t$ is a correction string telling the algorithm which corrections to make. That is, if $w_i = j$, then in the $i^{th}$ step the algorithm flips the assignment of the $j^{th}$ literal in $C$. Clearly, if $t = 3n$ and $w \in \{1,\ldots,k\}^t$ is sampled uniformly at random, this is equivalent to calling the original procedure Schöning-Walk($F,\alpha$). The function Schöning-Walk($F,t,\alpha$) is called with $t = 3n$, thus $\{1,\ldots,k\}^{3n}$ is the space of all possible correction strings. Why can we not simply compute a covering code for this space?

Imagine two iterations $j_1 < j_2$ of the loop in Schöning-Walk. In step $j_1$ the algorithm selects an unsatisfied clause $C_i$, and in step $j_2$ it selects $C_j$. Which clause $C_j$ it picks may depend on the random choice it made at step $j_1$. Therefore, what it means for a correction string to be correct at its $j_2^{th}$ position depends on its $j_1^{th}$ position. Therefore, two correction strings might have Hamming distance 1, but still it is possible that the first string is correct in every position and the second is wrong everywhere. This renders useless the whole machinery of covering codes.

Note that the picture becomes less bleak if there are disjoint unsatisfied clauses $C_1,\ldots,C_t$ and our correction string tells us only what to do on those clauses: Which literal of $C_j$ is the “correct one” does in fact not depend on which literal the algorithm flipped in clause $C_i$. Therefore, in this scenario the covering code machinery works, and this is exactly what we exploit in searchball-fast.
Chapter 4

Local Search Algorithms for CSP

In this chapter we investigate generalizations of Schöning’s random walk algorithm and deterministic versions thereof to \((d, k)\)-CSP. In Section 4.1 contains a brief description of the algorithm of Dantsin et al. and its analysis for CSP. In Section 4.2 we present a simple method that significantly speeds up the algorithm of Dantsin et al. for \((d, k)\)-CSP when \(d \geq 3\). Note that any algorithm for SAT can be used to solve CSP: Given a \((d, \leq k)\)-CSP formula, one randomly restricts every variable to two truth values and formulates the resulting problem as a \(k\)-CNF formula. If the CSP formula is satisfiable, the CNF formula is satisfiable with probability at least \((2/d)^n\). The result we present in Section 4.3 is a deterministic version of this reduction. It can be used to obtain a deterministic algorithm for \((d, k)\)-CSP the running time of which matches that of Schöning’s random walk algorithm, up to a subexponential factor. The result of Section 4.3 is joint work with Robin Moser [MS10]. Note that when we speak about \((d, k)\)-CSP as a decision problem, we consider \(d\) and \(k\) as parameters of that language and not as part of the problem instance. This means that we consider \(d\) and \(k\) to be constants that might be hidden by the \(O\)-notation.

4.1 Schöning and cover-search for \((d, \leq k)\)-CSP formulas

It is obvious how to generalize Schöning’s algorithm to CSP problems with more than two truth values, and Schöning [Sch99] in fact deals with boolean
SAT as well as with CSP problems. Below we state the algorithm CSP–Schöning, the version of Schöning for general values of $d$.

Algorithm 6 Schöning–CSP($(d, \leq k)$-CSP formula $F$)

1: $\alpha \leftarrow_{\text{u.a.r.}} [d]^n$ // $n := \lvert \text{vbl}(F) \rvert$
2: for $i = 0, \ldots, cn$ do
3: // $c$ is a constant depending on $d$ and $k$, but not on $n$
4: if $\alpha$ satisfies $F$ then
5: return $\alpha$
6: else
7: $C \leftarrow$ any constraint of $F$ unsatisfied by $\alpha$
8: $(x \neq c) \leftarrow_{\text{u.a.r.}} C$ // a random literal from $C$
9: $c' \leftarrow_{\text{u.a.r.}} [d] \setminus \{c\}$ // choose a new color for $x$
10: $\alpha \leftarrow \alpha[x \mapsto c']$ // change the coloring $\alpha$
11: end if
12: end for
13: return failure

Theorem 4.1 (Schöning [Sch99]). If $F$ is a satisfiable $(d, \leq k)$-CSP formula on $n$ variables, then Schöning–CSP returns a satisfying assignment with probability at least \[
\left( \frac{d(k-1)}{k} \right)^{-n} \frac{1}{\text{poly}(n)}.
\]

Although Dantsin et al. [DGH+02] only consider the boolean case, their algorithm seamlessly generalizes to CSP problems with more values. The running time is again worse than Schöning’s:

Theorem 4.2. There is a deterministic algorithm solving $(d, k)$-CSP and running in time $O \left( \left( \frac{dk}{k+1} \right)^n \text{poly}(n) \right)$.

The proof of the theorem is very similar to the one of Theorem 3.5 on page 29 for the boolean case. It still makes sense to have a look at it, since it lays the ground for our improvement in Section 4.2. Since we are working over $d$ values, we consider the $d$-ary cube $\{1, \ldots, d\}^n$. We already encountered this object in Section 3.3. For two assignments $\alpha, \alpha' \in \{1, \ldots, d\}^n$, the Hamming distance $d_H(\alpha, \alpha')$ is the number of variables on which $\alpha$ and $\alpha'$ disagree, and $B_r^{(d)}(\alpha)$ is the $d$-ary Hamming ball of radius $r$ around $\alpha$, i.e., the set

$$B_r^{(d)}(\alpha) := \{ \alpha' \in \{1, \ldots, d\}^n \mid d_H(\alpha, \alpha') \leq r \}.$$
Note that here we consider the \(d\)-ary cube, where \(d\) is the number of truth values, whereas in Section [3.3] we considered the \(k\)-ary cube, where \(k\) is the size of the clauses. As we have seen in Section 3.3, the cardinality of this set, denoted by \(\text{vol}^{(d)}(n, r)\), is \(\sum_{i=0}^{r} \binom{n}{i}(d-1)^i\). In analogy to the decision problem \textsc{BALL-}k\textsc{-SAT}, we define \textsc{BALL-}(\(d, k\))-CSP: Given a \((d, \leq k)\)-CSP formula, an assignment \(\alpha\) and a radius \(r\), does there exist a satisfying assignment \(\alpha^*\) such that \(d_H(\alpha, \alpha^*) \leq r\)? We give an algorithm for this problem:

**Algorithm 7** \texttt{csp-searchball}((\(d, \leq k\))-CSP formula \(F\), assignment \(\alpha\), radius \(r\))

1. if \(\alpha\) satisfies \(F\) then
   2. return true
3. else if \(r = 0\) then
   4. return false
5. else
6. \(C \leftarrow \) any constraint of \(F\) unsatisfied by \(\alpha\)
7. return \(\bigvee_{x \neq c} \bigvee_{c' \neq c} \text{csp-searchball}(F, \alpha[x := c'], r - 1)\)
8. end if

**Lemma 4.3.** The algorithm \texttt{csp-searchball} solves \textsc{BALL-} \((d, k)\)-CSP in time \(O((k(d - 1))^r \text{poly}(n))\).

*Proof sketch.* How many recursive calls does a call to \texttt{csp-searchball} cause? If it executes line 7, the operator \(\bigvee_{x \neq c} \bigvee_{c' \neq c}\) iterates over the at most \(k\) literals of \(C\), and the operator \(\bigvee_{c' \neq c}\) over the \(d - 1\) colors besides \(c\). Thus, it causes at most \(k(d - 1)\) recursive calls.

According to Lemma [3.14] on page 34, there is a \(d\)-ary covering code \(C\) for \(\{1, \ldots, d\}^n\) of radius \(r\) and length \(n\) such that

\[
C \leq \left\lfloor \frac{n \ln(d) d^n}{\binom{n}{r} (d-1)^r} \right\rfloor,
\]

and \(C\) can be constructed deterministically in time \(O(|C| \text{poly}(n))\). We solve \((d, k)\)-CSP by calling \texttt{csp-searchball}(\(F, \alpha, r\)) for every \(\alpha \in C\). The total running time is

\[
\frac{d^n (k(d - 1))^r}{\binom{n}{r} (d-1)^r} \text{poly}(n) = \frac{d^n k^r}{\binom{n}{r}} \text{poly}(n).
\]
We re-write this expression as
\[ \frac{d^n k^r}{\binom{n}{r}} \text{poly}(n) = \frac{2^n k^r}{\binom{n}{r}} \frac{d^n}{2^n} \text{poly}(n) \]
and see that \(2^n k^r / \binom{n}{r}\) is exactly the expression we encountered in the boolean case. Therefore we conclude that also here a radius \(r = n/(k+1)\) is optimal, and the running time is
\[ \left( \frac{2k}{k+1} \right)^n \frac{d^n}{2^n} \text{poly}(n) = \left( \frac{dk}{k+1} \right)^n \text{poly}(n), \]
which proves Theorem 4.2. Here is the pseudocode of csp-cover-search:

**Algorithm 8** csp-cover-search((d, \(\leq k\))-CSP formula \(F\) over \(n\) variables)

1: \(r := n/(k+1)\)
2: construct a \(d\)-ary covering code \(C\) of radius \(r\) and \(|C| \leq \frac{[d]^n}{\text{vol}(d)(n,r)} \text{poly}(n)\)
3: return \(\bigvee_{\alpha \in C} \text{csp-searchball}(F, \alpha, r)\)

### 4.2 A Better Deterministic Algorithm for \((d, k)\)-CSP

We will describe a surprisingly simple improvement of csp-cover-search. This improvement does not yet reach the running time of Schöning and in particular does not improve the boolean case \(d = 2\). In the light of the recent algorithm by Moser and myself [MS10], the algorithm we present in this section may seem obsolete. However, we include it for several reasons: First, the idea seems interesting and should be kept in mind. Second, thinking about how to improve deterministic algorithms for \((d, k)\)-CSP for \(d \geq 3\) made us (Robin Moser and me) start thinking about \(d\)-ary covering codes, ultimately leading to the complete derandomization of \(k\)-SAT [MS10], which we described in Section 3.3.

**A Cheap Trick**

We begin by presenting a simple idea that appears to be completely stupid, but surprisingly improves the running time of csp-cover-search. Any \((4, k)\)-CSP formula with \(n\) variables can be encoded as a \((2, 2k)\)-CSP formula with \(2n\) variables: Just replace each 4-ary variable \(x\) by two binary variables \(x_1, x_2\).
encoding the colors 1, 2, 3, 4 by two bits in some way. For example, we could encode 1 by 01, 2 by 10, 3 by 11, and 4 by 00. With this encoding, a constraint

\[(x \neq 2 \lor y \neq 3)\]

translates into

\[(x_1 \neq 1 \lor x_2 \neq 0 \lor y_1 \neq 1 \lor y_2 \neq 1)\].

A \(k\)-constraint involving 4-ary variables becomes a \(2k\)-constraint involving binary variables. In other words, we obtain a \(2k\)-CNF formula. By Theorem 4.2 on page 42, we can solve \((2, 2k)\)-CSP in time

\[O \left( \left( \frac{4k}{2k + 1} \right)^{2n} \text{poly}(n) \right) . \quad (4.1)\]

For example, for \(k = 3\) this is \(O \left( \left( \frac{144}{49} \right)^{n} \text{poly}(n) \right) \leq O(2.939^{n})\), better than the running time of \(3^{n}\text{poly}(n)\) we obtain we directly applying Theorem 4.2 to a \((4, 3)\)-CSP formula. Not a great improvement, but one that has been obtained in a surprisingly simple way. Clearly, the same trick works whenever \(d\) is a power of 2:

**Theorem 4.4.** Suppose \(d = 2^\ell\) for some \(\ell \in \mathbb{N}\). Then \((d, k)\)-CSP can be solved in time \(O \left( \left( \frac{2\ell k}{\ell k + 1} \right)^{\ell n} \text{poly}(n) \right) . \)

It is not difficult to check that \(\left( \frac{2\ell k}{\ell k + 1} \right)^{\ell} < \frac{dk}{k+1}\) for any \(\ell \geq 2\), and thus Theorem 4.4 is an improvement over Theorem 4.2 whenever \(d\) is a power of 2 and \(d \geq 4\). This is surprising, since by replacing a \(2^\ell\)-ary variable \(x\) by \(\ell\) binary variables \(x_1, \ldots, x_\ell\), we are throwing away a lot of information: A clause contains \(x_i\) if and only if it also contains all \(x_1, \ldots, x_\ell\), but our \((2, \ell k)\)-CSP algorithm does not use this fact at all. Two question arise: First, can we improve Schöning’s algorithm in a similar fashion? Second, what do we do if \(d\) is not a power of 2? The first question is answered quickly: Schöning’s running time on a \((2^\ell, k)\)-CSP formula with \(n\) variables is at most \(O \left( \left( \frac{2(\ell k - 1)}{\ell k} \right)^{n} \text{poly}(n) \right) . \)

Using binary encoding, this becomes \(O \left( \left( \frac{2(\ell k - 1)}{\ell k} \right)^{\ell n} \text{poly}(n) \right) . \) Again, a simple calculation shows that \(\left( \frac{2(\ell k - 1)}{\ell k} \right)^{\ell} > \frac{d(k - 1)}{k}\), and thus binary encoding seems to make Schöning’s algorithm worse. The second question is more difficult: If \(d\) is not a power of 2, say \(d = 3\), how can we encode our variables? Using two binary variables to encode one ternary variable is wasteful.
and does not lead to an improvement. It turns out that it makes sense to again study $(4,k)$-CSP, but this time not applying any explicit encoding in binary.

Suppose \textit{csp-searchball} runs on a $(4,3)$-CSP formula and encounters the unsatisfied constraint $(x \neq 1) \lor (y \neq 2) \lor (z \neq 4)$. It causes $k(d - 1) = 9$ recursive calls, exhausting all possibilities to change the assignment at exactly one variable occurring in the constraint. See Figure 4.1 for an illustration. How does the branching look after applying binary encoding? Suppose we encode the values 1, 2, 3, and 4 by 01, 10, 11, and 00, respectively. The first literal $(x \neq 1)$ translates into $(x_1 \neq 0) \lor (x_2 \neq 1)$, and the algorithm in one branch changes $x_1$ to 1, and in the other one changes $x_2$ to 0. In other words, it changes $x$ to 3 and $x$ to 4, but does not change $x$ to 2 in any branch. The branching now looks like in Figure 4.2 and we have eliminated three recursive calls. Rather than viewing this from the point of binary encoding, the think of defining a graph on the colors as in Figure 4.3: The colors 3 and 4 are neighbors of color 1, but 2 is not a neighbor of 1. Therefore, the first literal $(x \neq 1)$ causes no call with $x$ being set to 2. We only change colors to neighboring colors. Reducing the number of neighbors clearly speeds up \textit{csp-searchball}. However, as there is no free lunch, it reduces the number of elements at distance $r$, i.e., the volume of the Hamming balls. Apparently, the favorable effect is stronger, and we can improve upon the running time. Since we are altering the notion of neighbors and thus of distance, we should define everything formally.
4.2. A Better Deterministic Algorithm for $(d, k)$-CSP

![Figure 4.3: The neighborhood relation between the colors 1, 2, 3, 4 when using binary encoding.](image)

**G-Distance, G-Balls, and G-searchball**

We will systematically explore the possibilities offered by defining a graph on the $d$ colors. Let $[d]$ be the set of colors, and let $G = ([d], E)$ be a (possibly directed) graph. For two colors $c, c'$, we denote by $d_G(c, c')$ the length of a shortest path from $c$ to $c'$ in $G$. If $G$ is directed, this is not necessarily a metric, and therefore we rather call it a *distance function*. It gives rise to a distance function on $[d]^n$: For two assignments $\alpha, \beta \in [d]^n$, we define

$$d_G(\alpha, \beta) = \sum_{i=1}^{n} d_G(\alpha_i, \beta_i).$$

(4.2)

This distance induces the notion of balls $B_r^{(G)}(\alpha) := \{ \beta \in [d]^n \mid d_G(\alpha, \beta) \leq r \}$, and of dual balls $\{ \beta \in [d]^n \mid d_G(\beta, \alpha) \leq r \}$. If $G$ is undirected, balls and dual balls coincide, and for $G$ being $K_d$, the complete undirected graph, $d_G$ is simply the Hamming distance. If $G$ is vertex-transitive (and possibly directed), the cardinality $|B_r^{(G)}(\alpha)|$ does not depend on $\alpha$, and we define $\text{vol}^{(G)}(n, r) := |B_r^{(G)}(\alpha)|$. By double-counting, this is also the cardinality of dual balls. In particular, a vertex-transitive graph is regular. Let $\delta$ denote the number of edges leaving each vertex in $G$. As before, we define a parametrized problem: Given $F, \alpha$ and $r$, does $B_r^{(G)}(\alpha)$ contain a satisfying assignment? The algorithm G-searchball, which is almost identical to csp-searchball on page 43, solves this problem in time $O((\delta k)^r \text{poly}(n))$. 

**Covering Codes and the Volume of G-Balls**

Wherever there is a distance function, there are balls. Wherever there are balls, there are covering codes: Set $C \subseteq [d]^n$ is a code of length $n$ and covering $G$-
Algorithm 9 G-searchball(CSP formula $F$, assignment $\alpha$, radius $r$)

1: if $\alpha$ satisfies $F$ then
2: return true
3: else if $r = 0$ then
4: return false
5: else
6: $C \leftarrow$ any constraint of $F$ unsatisfied by $\alpha$
7: return $\bigvee_{(x \neq c) \in C} \bigvee_{c' : (c,c') \in E(G)} G$-searchball($F, \alpha[x := c'], r - 1$)
8: end if

radius $r$ if

$$\bigcup_{\alpha \in C} B^{(G)}_r(\alpha) = [d]^n.$$ 

A volume argument shows that any such code has at least $d^n / \text{vol}^{(G)}(n, r)$ elements. This lower bound can be achieved up to a polynomial factor in $n$, as we will show below in Lemma 4.6 on the facing page. To employ covering codes in this context, we have to estimate the volume of $G$-balls. Fix some vertex $u \in V(G)$ and let $\alpha$ be the assignment that sets every variable to $u$. We will estimate the volume of $B^{(G)}_r(\alpha)$. Let $T_G(n, r) := \text{vol}^{(G)}(n, r) - \text{vol}^{(G)}(n, r - 1)$, i.e., the number of assignments having distance exactly $r$ from $\alpha$. We claim that for any $x \in \mathbb{R}$,

$$\left( \sum_{v \in V(G)} x^{d_G(u,v)} \right)^n = \sum_{i=0}^{(d-1)n} T_G(n, i)x^i. \quad (4.3)$$

Let us try to understand why this holds: In how many ways do we get the term $x^i$ when expanding the product on the left-hand side? For every of the $n$ factors, we choose a summand, i.e., a vertex $v_j$. This yields $x^i$ if and only if $\sum_j d_G(u,v_j) = i$. But this exactly means that $(v_1, v_2, \ldots, v_n)$ is an assignment $\beta$ with $d_G(\alpha, \beta) = i$. Therefore, $T_G(n, i)$, the coefficient of $x^i$ on the right-hand side, is the number of such assignments. Note that $d_G(u, v) \leq d - 1$ for any vertex $v \in V(G)$, therefore $d_G(\alpha, \beta) \leq (d - 1)n$ for any two assignments, and thus it suffices to let the sum run up to $(d - 1)n$. The expression in (4.3) is a function in $x$, called the generating function of $T_G(n, i)$, and we denote it by $f_G(x)$. Generating functions are a well-established tool in combinatorics, see for example the book generatingfunctionology [Wil06]. Here we use the generating function to bound the volume of $G$-balls:
Lemma 4.5. Let $G$ be a vertex-transitive graph. The following upper and lower bounds on $\text{vol}^{(G)}(n, r)$ hold:

(i) For every $0 < x \leq 1$, $n \in \mathbb{N}$ and $0 \leq r \leq (d - 1)n$, it holds that $\text{vol}^{(G)}(n, r) \leq f_G(x)^n / x^r$.

(ii) For every $0 < x \leq 1$ and $n \in \mathbb{N}$, there is a $0 \leq r \leq (d - 1)n$ such that $\text{vol}^{(G)}(n, r) \geq f_G(x)^n dn / x^r$.

Proof. For (i), we use (4.3) and compute

$$f_G(x)^n = \sum_{i=0}^{(d-1)n} T_G(n, i) x^i \geq \sum_{i=0}^r T_G(n, i) x^i \geq \sum_{i=0}^r T_G(n, i) x^r = \text{vol}^{(G)}(n, r) x^r$$

and solve for $\text{vol}^{(G)}(n, r)$. For (ii), we again use (4.3), but now we want to bound its right-hand side from above. Clearly, there is some $0 \leq r \leq (d - 1)n$ that maximizes $T_G(n, r) x^r$. For this $r$, it holds that

$$f_G(x)^n = \sum_{i=0}^{(d-1)n} T_G(n, i) x^i \leq ((d - 1)n + 1) T_G(n, r) x^r \leq dn T_G(n, r) x^r \leq dnx^r \text{vol}^{(G)}(n, r),$$

and we solve for $\text{vol}^{(G)}(n, r)$. This completes the proof. \hfill \Box

Let us generalize Lemma 3.14 to arbitrary vertex-transitive graphs $G$ on $d$ vertices.

Lemma 4.6. Let $G$ be a vertex-transitive graph on $d$ vertices. For all $0 < x \leq 1$ and $n \in \mathbb{N}$, there is a radius $r \in \{0, 1, \ldots, (d - 1)n\}$ and a code $C \subseteq \{1, \ldots, d\}^n$ such that

$$\bigcup_{\alpha \in C} B_r^{(G)}(\alpha) = \{1, \ldots, d\}^n$$

and

$$|C| \leq \frac{d^nx^r}{f_G(x)^n} \text{poly}(n).$$

Furthermore, $r$ can be computed in polynomial time in $n$ and $C$ can be constructed using $O(|C| \text{poly}(n))$ space and time.
We will prove this lemma in Chapter 5. By calling G-searchball\((F, \alpha, r)\) for each \(\alpha \in \mathcal{C}\), we can solve \((d, k)\)-CSP deterministically in time
\[
d^n x^r (k \delta)^r / f_G(x)^n \text{poly}(n), \tag{4.4}
\]
where we are free to choose any vertex-transitive graph \(G\) and any \(x \in (0, 1]\). By setting \(x = 1/kd\), the terms \(x^r\) and \((k \delta)^r\) in (4.4) cancel, and nothing will depend on \(r\) anymore. As it turns out, when we choose \(x = 1/kd\), Lemma 4.6 automatically gives us the optimal radius \(r\). It remains to find the optimal graph \(G\).

**Directed Cycles**

We analyze the algorithm for \(G = C_d\), the directed cycle on \(d\) vertices. Clearly, \(\delta = 1\), and therefore G-searchball runs in time \(k^r\). This is as fast as we can expect for any vertex-transitive graph that is not the empty graph. What is \(\text{vol}(C_d)(n, r)\)? Fix a vertex \(u \in V(G)\). For each \(0 \leq i \leq d - 1\), there is exactly one vertex \(v \in V(G)\) with \(d_G(u, v) = i\). Therefore
\[
f_{C_d}(x) = 1 + x + x^2 + \cdots + x^{d-1}.
\]
We evaluate (4.4) with \(x = 1/k\) and see that the running time is at most
\[
d^n x^r k^r / f_G(x)^n \text{poly}(n) = d^n / (1 + k^{-1} + k^{-2} + \cdots + k^{-d+1}) n \text{poly}(n) = d(k - 1) / k \cdot k^d / (k^d - 1)^n \text{poly}(n),
\]
and we have proven the main theorem of this section.

**Theorem 4.7.** For all \(d\) and \(k\), there is a deterministic algorithm solving \((d, k)\)-CSP in time
\[
O\left(\left(\frac{d(k - 1)}{k} \cdot \frac{k^d}{k^d - 1}\right)^n \text{poly}(n)\right).
\]

**Optimality of the Directed Cycle**

We will show that our analysis cannot be improved by choosing a different vertex-transitive graph \(G\) or a different radius \(r\). Actually we could work with a weaker notion than vertex-transitivity. What we need is that there are
numbers \(d_0, d_1, \ldots\) such that for every vertex \(u \in V(G)\) and \(i \leq 0\) there are exactly \(d_i\) vertices \(v\) with \(d_G(u, v) = i\). Such a graph is \(d_1\)-regular and satisfies our needs. Certainly every vertex-transitive graph \(G\) has these properties. Since \(G\) is finite, the sequence \(d_0, d_1, \ldots\), eventually becomes 0. Denoting the diameter of \(G\) by \(s\), it holds that \(d_i = 0\) for all \(i > s\). If \(G\) is connected (which we do not assume), the \(d_i\) add up to \(d\). Since \(G\) is vertex-transitive, the \(d_i\) do not depend on the vertex \(u\), and \(G\) is \(d_1\)-regular. Therefore, \(G\)-searchball runs in time \((kd_1)^r\text{poly}(n)\) on a \((d, \leq k)\)-CSP formula. By the upper bounds on \(\text{vol}^r_G(n, r)\) in Lemma 4.5 on page 49, any code \(C \subseteq [d]^n\) with \(\bigcup_{\alpha \in C} B_r^G(\alpha) = [d]^n\) satisfies

\[
|C| \geq \frac{d^n}{\text{vol}^G(n, r)} \geq \frac{d^n x^r}{f_G(x)^n} = \frac{d^n x^r}{(\sum_{i=0}^s d_i x^i)^n}
\]

for all \(x \in (0, 1]\). Since \(G\)-searchball takes time \((kd_1)^r\), the total running time is at least

\[
|C|(kd_1)^r \geq \frac{d^n x^r (kd_1)^r}{(\sum_{i=0}^s d_i x^i)^n},
\]

where this inequality holds for all \(x \in (0, 1]\). Setting \(x = \frac{1}{kd_1}\), we see that the running time is at least

\[
d^n \left(\sum_{i=0}^s d_i k^{-i} d_1^{-i}\right)^n.
\]

In a \(d_1\)-regular graph, the number of vertices at distance \(i\) from \(u\) can be at most \(d_i^1\). In other words, \(d_i \leq d_1^i\), and the above expression is at least

\[
d^n \left(\sum_{i=0}^s k^{-i}\right)^n,
\]

which, up to a polynomial factor, is the same as what we get for the directed cycle on \(d\) vertices. This is not a lower bound on the running time when using the graph \(G\). For this, we would also have to provide a lower bound on the running time of \(G\)-searchball. However, it shows that the analysis cannot be improved alone by using a graph \(G\) different from the directed cycle.

**Applying \(G\)-Distance to Schöning**

We can apply the same idea to Schöning’s algorithm: When picking a literal \((x \neq c)\) uniformly at random from an unsatisfied constraint of \(F\) (see line 8 of Schöning-CSP), we choose a new truth value \(c'\) uniformly at random.
from the set \( \{ c' \in [d] \mid (c, c') \in E(G) \} \). With \( G = K_d \), this is the original algorithm Schöning. Is there a graph on \( d \) vertices for which modified Schöning achieve a success probability that is significantly better than for \( K_d \)? Stefan Schneider answered this question in his Master’s Thesis [Sch09]:

**Theorem 4.8** (Stefan Schneider [Sch09]). Let \( G \) be a vertex-transitive graph on \( d \) vertices. Then the modified Schöning using \( G \) has a success probability of at least

\[
\left( \frac{d(k - 1)}{k} \right)^n \left( \frac{1}{\text{poly}(n)} \right),
\]

and this is tight up to polynomial factors in \( n \).

Actually Schneider proved this theorem for a more general graph class, so-called distance-regular graphs. Note that his result seemingly contradicts our back-of-the-envelope calculation that Schöning deteriorates when using binary encoding, i.e., when \( G \) is the \( \log_2(d) \)-dimensional Hamming cube. There is no contradiction, however: Our calculation above was correct, but simply suboptimal, not capturing the full power of Schöning’s algorithm when operating with a graph on the colors.

### 4.3 A Deterministic Reduction from \((d, k)\)-CSP to \(k\)-SAT

In this section we describe a simple way how to turn any given algorithm for \( k \)-SAT into an algorithm solving \((d, k)\)-CSP. This is joint work with Robin Moser [MS10].

**Theorem 4.9.** There is a randomized algorithm that takes a \((d, \leq k)\)-CSP formula \( F \) over \( n \) variables as input, runs in polynomial time in \( n \), and outputs a \( k \)-CNF formula \( F' \) over \( n \) variables such that

- if \( F \) is unsatisfiable, then \( F' \) is unsatisfiable, too,
- if \( F \) is satisfiable, then \( F' \) is satisfiable with probability at least \( (2/d)^n \).

**Proof.** For each variable \( x \) in \( F \), choose two distinct truth values from \( \{1, \ldots, d\} \) uniformly at random and independently and restrict \( x \) to these two values. The resulting problem is a boolean problem, since we have only two choices for every variable. Clearly, if \( F \) is unsatisfiable, it cannot become satisfiable by this restriction. On the other hand, if \( \alpha \) is a satisfying assignment of \( F \), then
4.3. A Deterministic Reduction from \((d, k)\)-CSP to \(k\)-SAT

with probability exactly \((2/d)^n\), \(\alpha\) has survives the restriction process. Therefore, if \(F\) is satisfiable, then with probability at least \((2/d)^n\), at least one satisfying assignment survives the restriction process, in which case the resulting boolean formula \(F'\) is satisfiable.

**Corollary 4.10.** If there is a Monte Carlo algorithm solving \(k\)-SAT in time \(t(n)\) with probability at least \(1/2\), then there is a Monte Carlo algorithm solving \((d, k)\)-CSP in time \((d/2)^n t(n)\)poly\((n)\) with success probability at least \(1/2\).

We state and prove a deterministic version of this reduction.

**Theorem 4.11.** There is a deterministic algorithm that takes a \((d, \leq k)\)-CSP formula \(F\) over \(n\) variables as its input and outputs a sequence \(F_1, F_2, \ldots, F_m\) of \(m\leq(d/2)^n\)poly\((n)\) many \(k\)-CNF formulas such that

- if \(F\) is unsatisfiable, then all \(F_i\) are unsatisfiable, too,
- if \(F\) is satisfiable, then at least one \(F_i\) is satisfiable.

Furthermore, this algorithm runs in \(O((d/2)^n\)poly\((n)\)) time.

If \(A\) is a deterministic algorithm for \(k\)-SAT, we can use it to solve \((d, k)\)-CSP by calling it for \(F_1, \ldots, F_m\).

**Corollary 4.12.** If there is a deterministic algorithm solving \(k\)-SAT in time \(t(n)\), then there is a deterministic algorithm that solves \((d, k)\)-CSP and has a running time of \(t(n)(d/2)^n\)poly\((n)\).

For example, if we take \(A\) to be the deterministic algorithm solving \(k\)-SAT in time \((2(k - 1)/k)^n + o(n)\), this gives a deterministic algorithm for \((d, k)\)-CSP:

**Corollary 4.13.** For any \(d, k \geq 2\), there is a deterministic algorithm solving \((d, k)\)-CSP and running in time

\[
\left(\frac{d(k - 1)}{k}\right)^{n + o(n)}
\]

**Proof of Theorem 4.11** A 2-box \(B \subseteq \{1, \ldots, d\}^n\) is a set of the form \(B = B_1 \times \cdots \times B_n\) where \(B_i \subseteq \{1, \ldots, d\}\) and \(|B_i| = 2\) for all \(1 \leq i \leq n\). In words, \(B\) is a subcube of \(\{1, \ldots, d\}^n\) of dimension \(n\) and side length 2. One can encode a 2-box by writing down each \(B_i\), i.e., writing down \(2n\) numbers between 1 and \(d\). Since \(d\) is a constant, this requires \(O(n)\) space.

Given a \((d, \leq k)\)-CSP formula \(F\) and the \(O(n)\)-sized encoding of a 2-box \(B \subseteq \{1, \ldots, d\}^n\), the problem of deciding whether \(B\) contains a satisfying assignment translates directly into a \(k\)-CNF formula over \(n\) variables. Hence
our strategy is the following: Cover \( \{1, \ldots, d\}^n \) with 2-boxes, and then for each 2-box decide whether it contains a satisfying assignment. How many 2-boxes do we need to cover \( \{1, \ldots, d\}^n \)? Let \( C \) be a set of 2-boxes such that

\[
\bigcup_{B \in C} B = \{1, \ldots, d\}^n.
\]

Since every 2-box has \( 2^n \) elements, a volume argument shows that \( |C| \geq (d/2)^n \). Since we are consistently lucky when it comes to covering something, we can almost achieve this bound:

**Lemma 4.14.** For all \( d, n \in \mathbb{N} \) there is a set \( C \) of 2-boxes in \( \{1, \ldots, d\}^n \) with

\[
|C| \leq \frac{d^n}{2^n} \text{poly}(n),
\]

and \( C \) can be constructed using \( O(|C|) \) time and space.

We will prove this lemma in Chapter 5. We iterate through all 2-boxes \( B \in C \) and output the corresponding \( k \)-CNF formula. If \( F \) is satisfiable, then at least one 2-box contains the satisfying assignment, and the corresponding \( k \)-CNF formula is satisfiable. This completes the proof. \( \square \)
Chapter 5

Construction of Covering Codes

If you had done something twice, you are likely to do it again.

Brian Kernighan and Bob Pike, *The Unix Programming Environment*, p. 97)

We will now discuss the deterministic construction of covering codes. In the previous chapters we have seen four instances where we needed covering codes: (i) the algorithm \texttt{cover-search}, which solves \( k \)-SAT, needs to cover \( \{0, 1\}^n \) with Hamming balls (cf. Lemma 3.8 on page 31); (ii) the complete derandomization as well as \texttt{csp-cover-search} cover \( \{1, \ldots, d\}^n \) with \( d \)-ary Hamming balls (Lemma 3.14 on page 34); (iii) our improved algorithm for \((d, k)\)-CSP covers \( \{1, \ldots, d\}^n \) with \( G \)-balls (Lemma 4.6 on page 49); finally, the deterministic reduction from \((d, k)\)-CSP to \( k \)-SAT needs to cover \( \{1, \ldots, d\}^n \) with 2-boxes (Lemma 4.14 on the preceding page). In all cases, we can deterministically construct almost optimal coverings. We will demonstrate this for Hamming balls, proving Lemma 3.8 on page 31, and later introduce a framework that formalizes the conditions under which such constructions are possible. The results of this chapter are generalizations of the technique of Dantsin et al. [DGH+02].
5.1 Covering \( \{0, 1\}^n \) With Hamming Balls

**Lemma 5.1 (Existence of Good Covering Codes).** For all \( n \in \mathbb{N} \) and \( 0 \leq r \leq n \), every code \( C \subseteq \{0, 1\}^n \) of covering radius \( r \) and length \( n \) has at least \( \frac{2^n}{\text{vol}(n, r)} \) elements.

Furthermore, there is such a \( C \) with \(|C| \leq \left\lceil \frac{2^n \text{poly}(n)}{\text{vol}(n, r)} \right\rceil \).

**Proof.** The lower bound, i.e., that every code \( C \) of covering radius \( r \) and length \( n \) has at least \( \frac{2^n}{\text{vol}(n, r)} \) elements, follows from a volume argument.

We prove the upper bound with a probabilistic construction. Form \( C \) by sampling \( \left\lfloor \frac{n \ln(2) 2^n}{\text{vol}(n, r)} \right\rfloor \) elements from \( \{0, 1\}^n \) independently and uniformly at random with replacement. We claim that with positive probability this set \( C \) is a covering code of radius \( r \). To justify this claim, let us estimate the probability that it is not a covering code of radius \( r \), i.e.,

\[
\Pr\left[ \bigcup_{\alpha \in C} B_r(\alpha) \neq \{0, 1\}^n \right].
\]

Fix some \( \alpha^* \in \{0, 1\}^n \). If \( \alpha \in \{0, 1\}^n \) is sampled uniformly at random, then

\[
\Pr[\alpha^* \notin B_r(\alpha)] = 1 - \frac{\text{vol}(n, r)}{2^n}.
\]

Since every element in \( C \) is sampled independently, we obtain

\[
\Pr\left[ \alpha^* \notin \bigcup_{\alpha \in C} B_r(\alpha) \right] = \left(1 - \frac{\text{vol}(n, r)}{2^n}\right)^{|C|} < e^{-\frac{\text{vol}(n, r)}{2^n}|C|} \leq e^{-n \ln(2)} = 2^{-n}.
\]

This probability is the same for every \( \alpha^* \), since by the symmetry of \( \{0, 1\}^n \), every assignment “looks the same”. By the union bound, we observe that

\[
\Pr[\exists \alpha^* \in \{0, 1\}^n : \alpha^* \notin \bigcup_{\alpha \in C} B_r(\alpha)] \\
\leq \sum_{\alpha^* \in \{0, 1\}^n} \Pr[\alpha^* \notin \bigcup_{\alpha \in C} B_r(\alpha)] \\
< 2^n 2^{-n} = 1.
\]

With positive probability, the set \( C \) is a covering code of radius \( r \). This shows existence and completes the proof.

By investing an additional factor of \( n \), say by sampling \( \frac{2^n}{\text{vol}(n, r)} n^2 \) elements, this probability is not only positive, but very large: The probability that \( C \) is not a covering code of radius \( r \) is exponentially small in \( n \). This does not help us, however, since we want to explicitly construct such a code.
Definition 5.2. Let $\Sigma$ be a finite alphabet and $\mathcal{C} \subseteq \Sigma^*$ be some finite set. We say $\mathcal{C}$ can be constructed using $t$ time and $s$ space if there exists an algorithm running in time at most $t$ and using at most $s$ space that outputs the elements of $\mathcal{C}$ one after the other.

As usual in complexity theory, the space used by the algorithm does not take into account the size of the output. In particular, the space requirement can be polynomial even though the output size is exponential. If we can construct a covering code $\mathcal{C}$ of radius $r$ and length $n$ in time $O(|\mathcal{C}|)$ and space polynomial in $n$, then the algorithm cover-search runs in polynomial space, too: For each element $\alpha \in \mathcal{C}$ we call $\text{sat-searchball}(F, \alpha, r)$. After that call we discard $\alpha$, i.e., we do not have to store the whole set $\mathcal{C}$.

Lemma 5.3. For every $n \in \mathbb{N}$ and $0 \leq \rho \leq 1/2$, there is a covering code $\mathcal{C} \subseteq \{0, 1\}^n$ of length $n$ and radius $r := \lfloor \rho n \rfloor$ with $|\mathcal{C}| \leq 2^{\frac{n}{\rho n}} \cdot \frac{\text{vol}(n, r)}{\text{vol}(n, r')}$. Furthermore, $\mathcal{C}$ can be constructed using $O(|\mathcal{C}| \cdot \text{poly}(n))$ time and poly($n$) space.

Proof. Let $t := \lfloor \log_2 \log_2 n \rfloor$ and set $r' := rt/n$. By Lemma 5.1 there exists a code $\mathcal{C}' \subseteq \{0, 1\}^t$ of length $t$ and radius $r'$ such that

$$|\mathcal{C}'| \leq 2^t \frac{\text{poly}(t)}{\text{vol}(t, r')} .$$

We can find this code by iterating over all $2^t \leq n$ subsets of $\{0, 1\}^t$. Once we have found such a code $\mathcal{C}'$, we define

$$\mathcal{C} = \mathcal{C}' \times \mathcal{C}' \times \cdots \times \mathcal{C}' ,$$

the $n/t$-fold Cartesian product of $\mathcal{C}'$. Here, for $\alpha_1, \ldots, \alpha_{n/t} \in \mathcal{C}'$, we identify $(\alpha_1, \ldots, \alpha_{n/t})$ with the concatenated $n$-bit string $\alpha$. Therefore, $\mathcal{C}$ is a code of length $n$. We claim that is a code of radius $r$: Let $\alpha^* \in \{0, 1\}^n$. We partition this $n$-bit vector into $t$-bit vectors $\alpha_1^*, \ldots, \alpha_{n/t}^*$. For each $\alpha_i^*$, there exists $\alpha_i \in \mathcal{C}'$ such that $d_H(\alpha_i, \alpha_i^*) \leq r' = rt/n$. Therefore

$$d_H(\alpha, \alpha^*) = \sum_{i=1}^{n/t} d_H(\alpha_i, \alpha_i^*) \leq n/t \cdot rt/n = r .$$

The cardinality of $\mathcal{C}$ is

$$|\mathcal{C}| = |\mathcal{C}'|^{n/t} \leq \left( \frac{2^t \text{poly}(t)}{\text{vol}(t, r')} \right)^{n/t} . \quad (5.1)$$
It remains to compare \( \text{vol}(t, r')^{n/t} \) to \( \text{vol}(n, r) \). Using the bounds of Lemma 3.4 for the size of the binomial coefficients, we calculate

\[
\text{vol}(t, r')^{n/t} \geq \left( \frac{2^{H(r'/t)t}}{\sqrt{8t}} \right)^{n/t} = \frac{2^{H(r/n)n}}{(8t)^{n/2t}} \geq \text{vol}(n, r).
\]

Plugging this into (5.1) we obtain

\[
|C| \leq \frac{2^{n(8t)^{n/2t}}\text{poly}(n)}{\text{vol}(n, r)} \leq \frac{2^{n+o(n)}}{\text{vol}(n, r)}.
\]

The last inequality follows from the fact that \((8t)^{n/2t} \leq 2^{o(n)}\) provided that \(t = t(n) \to \infty\) as \(n \to \infty\). Finally, we can construct \(\mathcal{C}\) using polynomial space in \(n\): We only store \(\mathcal{C}'\) and output the elements of \(\mathcal{C}'^{n/t}\) one by one.

The construction here is slightly different from Dantsin et al. [DGH+02]. Our construction leaves a subexponential gap between the constructed code and an optimal one, whereas Dantsin et al. achieve a polynomial gap. On the other hand, our construction works in polynomial space, whereas theirs needs exponential space. The difference between our approach and that of Dantsin et al. [DGH+02] is that they employ a polynomial-time approximation algorithm for \textsc{Set Cover} for computing \(\mathcal{C}'\). This is what we will do in the next section.

### 5.2 A General Framework for Covering Codes

In this section we formalize the conditions under which a covering code construction as above works. We mainly generalize and abstract the techniques of Dantsin et al. [DGH+02].

Let \(H = (V, E)\) be a hypergraph. A cover of \(H\) is a subset \(F \subseteq E\) such that \(\bigcup_{e \in F} e = V\). In our applications, \(V\) can be \(\{0, 1\}^n\) or \(\{1, \ldots, d\}^n\), and \(E\) can be the set of all binary Hamming balls, \(d\)-ary Hamming balls, \(G\)-Hamming balls, or \(2\)-boxes. We say \(H\) is \(k\)-uniform if all edges have the same cardinality \(k\). \(H\) is regular if every \(v \in V\) is contained in the same number of edges. In all our applications, \(H\) is uniform and regular. This is already enough to guarantee the existence of almost optimal covers.

**Lemma 5.4.** Let \(H = (V, E)\) be a \(k\)-uniform and regular hypergraph. Then

(i) every cover of \(H\) has at least \(|V|/k\) elements,

(ii) there is a cover of \(H\) with at most \(\lceil |V| \ln |V|/k \rceil\) elements.
5.2. A General Framework for Covering Codes

Proof. The lower bound is clear since every cover $F$ of $H$ has to satisfy $|F| \cdot k \geq |V|$. For the upper bound we use a probabilistic construction. Form $F$ by sampling $\lceil |V| \ln |V|/k \rceil$ elements uniformly at random and independently with replacement from $V$. We claim that

$$\Pr[\bigcup_{e \in F} e = V] > 0,$$

which proves that a cover of the claimed size exists. To see that (5.2) holds, fix an arbitrary vertex $u$. What is the probability that $u \not\in \bigcup_{e \in F} e$? We know that $H$ is $d$-regular, therefore $u$ is contained in exactly $d$ hyperedges, so for a fixed vertex $u$ and one randomly sampled hyperedge $e$, it holds that

$$\Pr[u \in e] = \frac{d}{|H|} = \frac{k}{|V|},$$

where the last equality follows from double counting. Since we sample the edges of $F$ independently, it holds that

$$\Pr\left[u \not\in \bigcup_{e \in F} e\right] = \Pr[\forall e \in F : u \not\in e] = \left(1 - \frac{k}{|V|}\right)^{|F|} < e^{-k|F|/|V|} \leq e^{-\ln |V|} = \frac{1}{|V|}.$$

This is the probability that a fixed vertex $u$ is not covered. By the union bound, we obtain

$$\Pr\left[\bigcup_{e \in F} e \neq V\right] = \Pr\left[\exists u \in V : u \not\in \bigcup_{e \in F} e\right] \leq \sum_{u \in V} \Pr\left[u \not\in \bigcup_{e \in F} e\right] < \frac{|V|}{|V|} = 1.$$

This lemma shows the existence of covering codes of the desired size in Lemma 3.8 on page 31 (Hamming balls), Lemma 3.14 on page 34 ($k$-ary Hamming balls), Lemma 4.14 on page 54 (2-boxes), and Lemma 4.6 on page 49 ($G$-balls). For deterministically constructing these codes, we will apply a block code construction. For this, we need that $H$ has certain additional properties.
Efficient Representation

For the rest of the section, we assume that $H = (V, E)$ with $V = \{1, \ldots, d\}^n$ for some constant $d$. We require $H$ to have an efficient representation. That is, given $0 \leq i < |V|$ and $0 \leq j < |E|$, we can decide in polynomial time in $n$ whether the $j^{th}$ edge contains the $i^{th}$ vertex. We assume there is some underlying ordering of the vertices and edges. In our applications of course we do not represent vertices and hyperedges by their indices: We represent a vertex $u \in \{1, \ldots, d\}^n$ simply as a string of length $n$. To represent a hyperedge, for example a ball, we write down its center and radius. This surely is an efficient representation, i.e., we can check efficiently whether $u \in e$ for a given vertex $u$ and hyperedge $e$. Note that describing a Hamming by its center and radius takes polynomial spaces in $n$, although the size of the ball might be exponential in $n$. The idea is to construct an almost optimal cover for a much smaller hypergraph $H'$ by using an approximation algorithm, and then to show how to blow up this cover of $H'$ to a cover for $H$.

Block Code Constructions

**Definition 5.5.** Let $d, n', b \in \mathbb{N}$, $n = n'b$, and let $H = (V, E)$ and $H' = (V', E')$ be uniform and regular hypergraphs with $V = \{1, \ldots, d\}^n$ and $V' = \{1, \ldots, d\}^n$. For $g \in \mathbb{R}$ we say that $H$ and $H'$ allow a block code construction with gap $g$ if

(i) both $H$ and $H'$ have an efficient representation,

(ii) for all $e_1, \ldots, e_b \in E'$ there is an $e \in E$ such that $e_1 \times \cdots \times e_b \subseteq e$, and the index of $e$ in $E$ can be computed given the indices of $e_1, \ldots, e_b$ in $E'$ in time $\text{poly}(n)$,

(iii) $H$ is $k$-uniform and $H'$ is $k'$-uniform and $k/g \leq k'^b \leq k$.

The inequality $k'^b \leq k$ in point (iii) already follows from (ii). The power of block code constructions is that we can choose $n'$ so small to be comfortable to construct an optimal or approximately optimal cover $C'$ of $H'$, and then from there compute a cover of $H$. In the following theorem, $d$ is a constant. This means that the $O(\cdot)$-notation may hide constants that depend on $d$, but neither on $n, n'$, nor $b$.

**Theorem 5.6.** Let $n', b \in \mathbb{N}$, $n = n'b$, and let $H = (\{1, \ldots, d\}^n, E)$ and $H' = (\{1, \ldots, d\}^{n'}, E')$ be uniform and regular hypergraphs. Suppose $H$ is $k$-uniform and $H'$ is $k'$-uniform. If $H$ and $H'$ allow a block code construction with gap $g$, then there is a cover $C$ of $H$ of size

$$\frac{d^n}{k} n'^{O(b)} g$$
and $C$ can be constructed using at most $O(|C| \text{poly}(n) + \text{poly}(d^{n'}) \text{ poly}(d^{n'}))$ time and at most poly($d^{n'}$, $n$) space, where poly is some fixed polynomial, possibly depending on $d$, but neither on $n'$ nor $b$.

This theorem has two typical applications. First, if $n'$ is very small compared to $n$, for example $n' \in O(\log n)$, then the space requirement is polynomial in $n$. The disadvantage is that $b$ becomes large, and the overhead $n'\text{ poly}(b)$ is superpolynomial. Second, if $b$ is a (large) constant and the gap $g$ is polynomial in $n$, then the size of the constructed code exceeds the size of an optimal code only by a polynomial in $n$, but the space requirement becomes exponential.

Proof of Theorem 5.6 We have to give an algorithm that outputs the elements of a suitable cover $C \subseteq \{1, \ldots, d\}^n$ using $O(|C| \text{poly}(n) + \text{poly}(d^{n'}) \text{ poly}(d^{n'}))$ time and $\text{poly}(d^{n'}, n)$ space, where $\text{poly}$ is some fixed polynomial, depending neither on $n'$ nor $b$.

The set $V'$ has cardinality $d^{n'}$. We write down the covering problem explicitly as an instance of the SET COVER problem. That is, we list all vertices $|V'|$ and $|E'|$ edges. There is an approximation algorithms for SET COVER achieving an approximation ratio of $O(\log |V'|)$ and running in polynomial time (see Hochbaum [Hoc97], for example). We know that the optimal code has at most size $\lceil |V'\ln |V'|/k' \rceil = \lceil d^{n'}n' \ln(d)/k' \rceil$, thus the $O(\log |V'|)$-approximation has size

$$|C'| \leq \frac{d^{n'}}{k'} \cdot O(n'^2).$$

Computing this set takes time $\text{poly}(d^{n'})$ where $\text{poly}$ is the polynomial coming from the running time of the approximation algorithm. Once this set is computed, our algorithm stores it, which takes $O(|C|n)$ space (note that each element of $C$ requires $O(n)$ space).

Next we consider the set $C^{mb}$. By point (ii) there is some function

$$\Phi : E^{mb} \to E$$

such that $e_1 \times \cdots \times e_b \subseteq \Phi(e_1, \ldots, e_b)$ and $\Phi$ can be computed in polynomial time. We define

$$C := \{\Phi(e_1, \ldots, e_b) \mid (e_1, \ldots, e_b) \in C^{mb}\}.$$

This is a cover of $H$: Let $u \in V$. We chop up $u$ into $b$ pieces $u_1, \ldots, u_b$, each in $\{1, \ldots, d\}^{n'} = V$. Since $C'$ is a cover of $H'$, there are edges $e_1, \ldots, e_b \in C'$ such that $u_i \in e_i$ for $1 \leq i \leq b$. Clearly $u \in e_1 \times \cdots \times e_b$, hence by construction there
is some $e \in C$ such that $u \in e$. The size of $C$ is
\[
|C| \leq |C'|^b = \left( \frac{d^{n'}}{k^b} O(n'^2) \right)^b \leq \frac{d^n}{k} n'^O(b) g,
\]
where in the last inequality we used that $k^b \geq k/g$. This simply states that $e_1 \times \cdots \times e_b$ comprises at least a $g$-fraction of $e \in E$.

Our algorithm can now output $C$ by iterating through $C'$. In addition to the $p(|d^{n'}| \text{poly}(n))$ space needed for computing and storing $C'$, the algorithm needs $O(n'b)$ additional space to store $b$ counters, each iterating from 1 to $|C'|$. Thus, to output $C$ the algorithm requires $p(|d^{n'}| \text{poly}(n) + O(n'b) = p(|d^{n'}| \text{poly}(n))$ space and $p(d^{n'}) + O(|C| \text{poly}(n))$ time. \hfill \Box

5.3 Application to 2-Boxes

Recall that a 2-box $B \subseteq \{1, \ldots, d\}^n$ is a set of the form $B_1 \times \cdots \times B_n$ where $B_i \subseteq \{1, \ldots, d\}$ and $|B_i| = 2$ for $1 \leq i \leq d$. In other words, a 2-box is a subcube of $\{1, \ldots, d\}^n$ of dimension $n$ and side length 2. Let $H_n$ be the hypergraph $(V_n, E_n)$ where $V_n = \{1, \ldots, d\}^n$ and $E_n$ is the set of all 2-boxes. Note that $|E_n| = (\frac{d}{2})^n \leq d^{2n}$.

**Lemma 5.7.** Let $n', b \in \mathbb{N}$ and $n = n'b$. Then $H_n$ and $H_{n'}$ allow block code constructions with gap 1.

**Proof.** We have to show three things. (i) $H_n$ has an efficient representation. This is not difficult to see: Each 2-box $B = B_1 \times \cdots \times B_n$ can be encoded by just writing down the $n$ sets $B_1, \ldots, B_n$. Obviously this allows us to test efficiently whether $u \in B$ for a given vertex $u$ and a given 2-box $B$. (ii) Given $b$ many 2-boxes $B^{(1)}, \ldots, B^{(b)}$ in $\{1, \ldots, d\}^{n'}$, their Cartesian product $B := B^{(1)} \times \cdots \times B^{(b)} \subseteq \{1, \ldots, d\}^n$ is a 2-box in $\{1, \ldots, d\}^n$ and can be computed in time poly($n$) given $B^{(1)}, \ldots, B^{(b)}$. (iii) $H_n$ is $2^n$-uniform and $n'$ is $2^n'$ uniform. Therefore the required inequality $k^b \leq k/g$ holds with equality, for $g = 1$. \hfill \Box

Having established that $H_n$ and $H_{n'}$ allow block code constructions with gap 1, we apply Theorem 5.6 on the previous page. Let $b$ be some constant. We conclude that $H$ has a cover $C$ of size
\[
\frac{d^n}{k} n'^O(b) g = \frac{d^n}{2^n} \left( \frac{n}{b} \right)^{O(b)} \cdot 1 = \frac{d^n}{2^n} \text{poly}(n)
\]
which can be constructed in time $O(|C| \text{poly}(n) + p(|V'| + |E'|))$. If we choose $b$ to be large enough yet still constant, 0 then $p(|V'| + |E'|) \leq p(d^{n'} + d^{2n'}) \leq (d/2)^n \leq |C|$. This proves Lemma 4.14 on page 54.
5.4 Application to \(G\)-balls

Here we explain how to cover \(\{1, \ldots, d\}^n\) with \(G\)-balls, which we introduced in Section 4.2. Let \(H_n^r\) be the hypergraph with vertex set \(\{1, \ldots, d\}^n\) and the balls \(B^{(G)}_\alpha(r)\) as hyperedges. Recall that we have introduced the generating function \(f_G(x)\) and proved the identity

\[
f_G(x) := \left( \sum_{v \in V(G)} x^{d_G(u,v)} \right)^n = \sum_{i=0}^{(d-1)n} T(n, i) x^i
\]

in (4.3) on page 48. Here \(T(n, i) = \text{vol}^{(G)}(n, i) - \text{vol}^{(G)}(n, i - 1)\) is the number of points at distance exactly \(i\) from a given point. In Lemma 4.5 on page 49 we have proved upper and lower bounds on \(\text{vol}^{(G)}(n, r)\) using \(f_G(x)\).

In the following lemma, \(d\) is again a constant, meaning that the \(O(\cdot)\)-notation may hide constants that depend on \(d\).

**Lemma 5.8.** For each \(0 < x \leq 1\) and \(n' \in \mathbb{N}\) there exists an \(r' \in \{0, 1, \ldots, dn'\}\) such that for all \(b \in \mathbb{N}\), the following holds: For \(n := bn'\) and \(r := br'\), the hypergraphs \(H_n^r\) and \(H_n^{r'}\) allow block code constructions with gap \((dn')^b\). Furthermore, \(H_n^r\) is \(k\)-uniform with \(k' \geq \frac{f_G(x)^n}{dn'x^{r'}}\), and \(r'\) can be computed in polynomial time in \(n'\).

**Proof.** We have to show that the three points of Definition 5.5 on page 60 are satisfied and that the uniformities \(k\) and \(k'\) fulfill the claimed bounds. We start by computing a suitable radius \(r'\). The lower bound in Lemma 4.5 on page 49 states that we can compute some \(r'\) with \(0 \leq r' \leq dn'\) such that \(\text{vol}^{(G)}(n', r') \geq f_G(x)^n/(dn'x^{r'})\). This means that \(H_n^{r'}\) is \(k'\)-uniform with

\[
k' \geq \frac{f_G(x)^n}{dn'x^{r'}}.
\]

We apply the upper bound in Lemma 4.5 on page 49 to conclude that

\[
k \leq \frac{f_G(x)^n}{x^{r'}}.
\]

Therefore we can bound the gap

\[
k'^b \geq \frac{f_G(x)^{bn'}}{(dn')^b x^{br'}} = \frac{f_G(x)^n}{(dn')^b x^r} \geq \frac{k}{(dn')^b}.
\]
This shows that point (iii) of Definition 5.5 is satisfied. To show point (ii), note that for any $u_1, \ldots, u_b \in V_{n'}$, it holds that

$$B_{r'}^{(G)}(u_1) \times \cdots \times B_{r'}^{(G)}(u_b) \subseteq B_r^{(G)}(u),$$

where $u := u_1 \circ \cdots \circ u_b \subseteq \{1, \ldots, d\}^n$ is the concatenation of $u_1, \ldots, u_b$. Therefore $k'^b \leq k$. Given $u_1, \ldots, u_b$ one can compute $u$ in time polynomial in $n$, which shows that point (ii) of Definition 5.5 is satisfied. Furthermore, combined with the bounds above, this shows that

$$k \geq k'^b \geq \frac{f_G(x)^n}{(dn')^b x^n},$$

thus $k$ and $k'$ satisfy the claim in the lemma. Point (i) of Definition 5.5 is clearly satisfied: We can efficiently compute the $G$-distance between two points $x, y \in V_n$. This finishes the proof.

For clarity we copy Lemma 4.6 from page 49:

**Lemma 5.9.** Let $G$ be a vertex-transitive graph on $d$ vertices. For every $0 < x \leq 1$ and $n \in \mathbb{N}$, there is a radius $r \in \{0, 1, \ldots, (d - 1)n\}$ and a code $C \subseteq \{1, \ldots, d\}^n$ such that

$$\bigcup_{\alpha \in C} B_r^{(G)}(\alpha) = \{1, \ldots, d\}^n$$

and

$$|C| \leq \frac{d^nx^r}{f_G(x)^n} \text{poly}(n).$$

Furthermore, $r$ can be computed in polynomial time in $n$ and $C$ can be constructed using time $O(|C| \text{poly}(n))$ space and time.

**Proof.** Let $b$ be some constant, to be determined later. If $n$ is not divisible by $b$, replace $n$ by $\tilde{n} = \lceil n/b \rceil b$. We construct a code for $\{1, \ldots, d\}^{\tilde{n}}$. This yields a cover for $\{1, \ldots, d\}^{\tilde{n}}$ by restricting every $\alpha$ in this code to its first $n$ coordinates. Since $\tilde{n} - n \leq b$, this will not much affect the size of $C$, nor the running time. For readability, we will simply assume that $n$ is divisible by $b$.

Set $n' = n/b$. Applying Lemma 5.8, we can efficiently compute numbers $r'$ and $r$ such that $H_r^{n'}$ and $H_{r'}^{n'}$ allow block code constructions with gap $(dn')^b = \text{poly}(n)$. Further, $H_r^{n'}$ is $k$-uniform with $k \geq \frac{f_G(x)^n}{x^r} \text{poly}(n)$. By Theorem 5.6 on page 61, we can construct a covering code $C$ of $G$-radius $r$ such that

$$|C| \leq \frac{d^nx^r}{k} \text{poly}(n) \leq \frac{d^nx^r}{x^r} f_G(x)^n \text{poly}(n)$$

64
using $O(|\mathcal{C}|\text{poly}(n) + p(d^{n'})$) time and space. Here $p$ is a polynomial that depends on the running time and approximation ratio of the approximation algorithm for Set Cover, but not on $b$. Choosing $b$ large enough, we can ensure that $p(d^{n'}) \leq |\mathcal{C}|$. \hfill $\square$
Chapter 6

PPZ for \((d, k)\)-CSP

So far, the contribution we presented in this thesis has been to obtain deterministic versions of existing randomized algorithms. In this chapter, we will investigate a randomized algorithm. Paturi, Pudlák, and Zane [PPZ99] give an elegant randomized algorithm for \(k\)-SAT, henceforth called \(ppz\), based on encodings of satisfying assignments. Although the success probability of \(ppz\) is exponentially smaller than that of Schöning’s random walk algorithm, it is a very important algorithm for at least two reasons: First, an improved version of it, given by the same authors and Michael Saks [PPSZ05], called \(ppsz\), is one of the two ingredients of the currently
fastest algorithm for 3-SAT (the other ingredient being Schöning’s algorithm). Furthermore, for k-SAT with \( k \geq 4 \), ppz is itself the fastest known algorithm. Second, ppz is better than Schöning when applied to \((d, k)\)-CSP for already moderately large values of \( d \). In this chapter we present and analyze a version of ppz for \((d, k)\)-CSP. Although ppz can easily be adapted to \((d, k)\)-CSP, the analysis of its success probability becomes much more complicated. We prove a correlation inequality to show that a certain easy to analyze case is in fact the worst case for ppz on \((d, \leq k)\)-CSP formulas.

6.1 Introduction

Consider the following randomized algorithm for \( k \)-SAT: Pick a variable uniformly at random from \( \text{vbl}(F) \) and call it \( x \). If the formula \( F \) contains the unit clause \((x)\), set \( x \) to 1. If it contains \((\overline{x})\), set it to 0. If it contains neither, set \( x \) uniformly at random (and if it contains both unit clauses, give up). This algorithm has been proposed and analyzed by Paturi, Pudlák, and Zane [PPZ99] and is called ppz.

The idea behind analyzing its success probability can be illustrated nicely if we assume, for the moment, that \( F \) is a \( k \)-CNF formula with a unique satisfying assignment \( \alpha \) setting all variables to 1. Let \( x \) be a variable. Since \( \alpha[x \mapsto 0] \) is not a satisfying assignment, there is a clause \( C_x = (x \lor \overline{y}_1 \lor \cdots \lor \overline{y}_{k-1}) \). With probability \( 1/k \), the algorithm picks and sets \( y_1, \ldots, y_{k-1} \) before picking \( x \). Supposed the \( y_j \) have been set correctly (i.e., to 1), the clause \( C_x \) is now reduced to \((x)\), and therefore \( x \) is also set correctly. Intuitively, this suggests that on average, the algorithm has to guess \((1 - 1/k)n\) variables correctly and can infer the correct values of the remaining \( n/k \) variables. This increases the success probability of the algorithm from \( 2^{-n} \) (simple guessing) to \( 2^{-n(1-1/k)} \).

It is obvious how to generalize the algorithm to \((d, k)\)-CSP problems. Again we process the variables in random order. When picking \( x \), we collect all unit constraints of the form \((x \neq c)\) and call the value \( c \) forbidden. Values in \([d]\) which are not forbidden are called allowed, and we set \( x \) to a value that we choose uniformly at random from all allowed values. How can one analyze the success probability? Let us demonstrate this for \( d = k = 3 \). Suppose \( F \) has exactly one satisfying assignment \( \alpha = (1, \ldots, 1) \). Let \( x \) be some variable in \( F \). Since neither \( \alpha[x \mapsto 2] \) nor \( \alpha[x \mapsto 3] \) satisfies \( F \), there are constraints of the form

\[
\begin{align*}
(x \neq 2 \lor y \neq 1 \lor z \neq 1) \\
(x \neq 3 \lor u \neq 1 \lor v \neq 1)
\end{align*}
\]
We call these constraints critical for \( x \). If all variables \( y, z, u, v \) are picked before \( x \), then there is only one allowed value for \( x \) left, namely 1, and with probability 1, the algorithm picks the correct values. If \( y, z \) come before \( x \), but at least one of \( u \) or \( v \) come after \( x \), then it is possible that the values 1 and 3 are allowed, and the algorithm picks the correct value with probability 1/2. In theory, we could list all possible cases and compute their probability. But here comes the difficulty: The probability of all variables \( y, z, u, v \) being picked before \( x \) depends on whether these variables are distinct! Maybe \( y = u \), or \( z = v \)... For general \( d \) and \( k \), we get \( d - 1 \) critical constraints

\[
C_2 := (x \neq 2 \lor y_1^{(2)} \neq 1 \lor \cdots \lor y_{k-1}^{(2)} \neq 1) \\
C_3 := (x \neq 3 \lor y_1^{(3)} \neq 1 \lor \cdots \lor y_{k-1}^{(3)} \neq 1) \\
\ldots \\
C_d := (x \neq d \lor y_1^{(d)} \neq 1 \lor \cdots \lor y_{k-1}^{(d)} \neq 1). 
\] (6.1)

We are interested in the distribution of the number of allowed values for \( x \). However, the above constraints can overlap in complicated ways, since we have no guarantee that the variables \( y_j^{(c)} \) are distinct. Our main technical contribution is a correlation lemma showing that in the worst case, the \( y_j^{(c)} \) are indeed distinct, and therefore we can focus on that case, which we are able to analyze. Note that in the boolean case, i.e., \( d = 2 \), there is only one critical constraint, hence there are no overlap issues, which is the reason why the analysis is simpler for \( k \)-SAT than for \((d, k)\)-CSP.

**Previous Work**

Feder and Motwani \[FM02\] were the first to generalize the \texttt{ppz} to CSP problems. In their paper, they consider \((d, 2)\)-CSP problem, i.e., each variable can take on \( d \) values, and every constraint has at most two literals. Consider the special case that the CSP formula \( F \) has exactly one satisfying assignment \( \alpha = (1, \ldots, 1) \). The critical constraints of \( x \) are of the form \((x \neq 2 \lor y^{(2)} \neq 1), \ldots, (x \neq d \lor y^{(d)} \neq 1)\). Suppose further that all \( y^{(i)} \) are distinct. This special case can indeed be analyzed similarly to the Boolean case in Paturi, Pudáč, and Zane \[PPZ99\]. Feder and Motwani show that this special case is the worst case. Unfortunately, their proofs do not generalize to higher values of \( k \), since for \( k \geq 3 \) the constraints in (6.1) can overlap in much more complex patterns than for \( k = 2 \).

Li, Li, Liu, and Xu \[LLLX08\] analyzed \texttt{ppz} for general CSP problems (i.e., \( d, k \geq 3 \)). Their analysis is overly pessimistic, though, since they distinguish
only the following two cases, for each variable $x$: When $ppz$ processes $x$, then either (i) all $d$ values are allowed, or (ii) at least one value is forbidden. In case (ii), $ppz$ chooses one value randomly from at most $d - 1$ values. Since case (ii) happens with some reasonable probability, this gives a better success probability than the trivial $d^{-n}$. However, the authors ignore the case that two, three, or more values are forbidden and lump it together with case (ii). Therefore, their analysis does not capture the full power of $ppz$.

**Our Contribution**

Our contribution is to show that “everything works as expected”, i.e., that in the worst case all variables $y_j^{(c)}$ in (6.1) are distinct and the formula has a unique satisfying assignment. For this case, we can compute (or at least, bound from below) the success probability of the algorithm.

Imagine the following random experiment: Choose $r \in [0,1]$ uniformly at random, set $p := 1 - r^{k-1}$ and then let $Z \sim \text{Bin}(d - 1, p)$ be a binomially distributed random variable. Its expectation is $(d - 1)p$. For some reason that will become clear in the analysis of our algorithm, we are interested in $\log_2(Z + 1)$. By the law of total probability, we have

$$G(d,k) := \mathbb{E}[\log_2(Z + 1)] = \sum_{j=0}^{d-1} \log_2(1 + j) \binom{d-1}{j} \int_0^1 (1 - r^{k-1})^j (r^{k-1})^{d-1-j} dr \quad (6.2)$$

**Theorem 6.1.** For $d, k \geq 2$, there is a randomized algorithm running in polynomial time which, given a satisfiable $(d, \leq k)$-CSP formula over $n$ variables, returns a satisfying assignment with probability at least $2^{-nG(d,k)}$.

Several times in this chapter we make use of Jensen’s inequality:

**Lemma 6.2** (Jensen’s Inequality, Theorem 2.4, page 24 of [MU05]). If $f : \mathbb{R} \to \mathbb{R}$ is a convex function and $X$ is a random variable, then $\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$, provided that both expectations exist. Similarly, if $f$ is concave, then $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$.

Let us analyze the asymptotic behavior of $ppz$ for fixed values of $k$ and $d$ tending towards infinity. Consider how the random variable $Z$ is defined. For fixed $r \in [0,1]$, $Z$ is binomially distributed according to $\text{Bin}(d - 1, 1 - r^{k-1})$. Thus, it should be highly concentrated around its expectation, making Jensen’s inequality a rather tight estimate. Since $Z \mapsto \log_2(Z + 1)$ is a concave
function, Jensen’s inequality yields
\[
\mathbb{E}[\log_2(Z + 1) | r] \leq \log_2(\mathbb{E}[Z | r] + 1)
\]
= \log_2((d - 1)(1 - r^{k-1}) + 1)
\leq \log_2(d(1 - r^{k-1}) + 1)
= \log_2(d) + \log_2 \left(1 - r^{k-1} + \frac{1}{d}\right).
\]

One checks that
\[
\lim_{d \to \infty} \int_0^1 \log_2 \left(1 - r^{k-1} + \frac{1}{d}\right) dr = \int_0^1 \log_2(1 - r^{k-1}) =: c_k,
\]
which at least for me is not entirely obvious. We conclude that the success probability \(2^{-nG(d,k)}\) for very large \(d\) is roughly
\[(2^{c_k}d)^{-n}.
\]

Unfortunately, \(c_k\) does not have a closed form in general, so we have to evaluate it numerically. We compare \((2^{c_k}d)^{-n}\) to the success probability of Schöning’s algorithm, which is
\[
\Omega \left(\left(\frac{k - 1}{k}d\right)^{-n}\right).
\]

We list the respective success probabilities of Schöning and ppz in the two tables below, for different values of \(d\) and \(k\). We write only the bases of the exponential functions, i.e. \(d(k-1)/k\) vs. \(2^{c_k}d\). All bounds are approximate.

<table>
<thead>
<tr>
<th>(d)</th>
<th>(k = 3)</th>
<th>(k = 4)</th>
<th>(k = 5)</th>
<th>(k = 6)</th>
<th>(k = 7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d = 2)</td>
<td>1.334</td>
<td>1.5</td>
<td>1.67</td>
<td>1.72</td>
<td>1.72</td>
</tr>
<tr>
<td>(d = 3)</td>
<td>2.0</td>
<td>2.25</td>
<td>2.4</td>
<td>2.572</td>
<td>2.572</td>
</tr>
<tr>
<td>(d = 4)</td>
<td>2.667</td>
<td>3.0</td>
<td>3.2</td>
<td>3.429</td>
<td>3.429</td>
</tr>
<tr>
<td>(d = 5)</td>
<td>3.334</td>
<td>3.75</td>
<td>4.0</td>
<td>4.1667</td>
<td>4.29</td>
</tr>
<tr>
<td>(d \to \infty)</td>
<td>0.667d</td>
<td>0.75d</td>
<td>0.8d</td>
<td>0.833d</td>
<td>0.858d</td>
</tr>
</tbody>
</table>

Figure 6.1: The success probability of Schöning for \((d, k)\)-CSP.

random walk algorithm is much faster than ppz, but ppz overtakes Schöning already for moderately large values of \(d\) and thus is, to our knowledge, the currently fastest algorithm for \((d, k)\)-CSP.
Figure 6.2: The success probability of ppz for \((d, k)\)-CSP.

### 6.2 The Algorithm

The algorithm itself is simple. It processes the variables \(x_1, \ldots, x_n\) according to some random permutation \(\pi\). When the algorithm processes the variable \(x\), it collects all unit constraints of the form \((x \neq c)\) and calls \(c\) forbidden. A value \(c\) that is not forbidden is called allowed. If the formula is satisfiable when the algorithm processes \(x\), there is obviously at least one allowed value. The algorithm chooses an allowed value \(c\) uniformly at random, replaces \(F\) by \(F[x \mapsto c]\), and proceeds to the next variable. For technical reasons, we think of the permutation \(\pi\) as part of the input to the algorithm, and sampling \(\pi\) uniformly at random from all \(n!\) permutations before calling the algorithm.

To analyze the success probability of the algorithm, we can assume that \(F\) is satisfiable, i.e., the set \(\text{sat}(F)\) of satisfying assignments is nonempty. This is because if \(F\) is unsatisfiable, the algorithm always correctly returns failure. For a fixed satisfying assignment, we will bound the probability

\[
\Pr[\text{ppz-csp}(F, \pi) \text{ returns } \alpha],
\]

where the probability is over the choice of \(\pi\) and over the randomness used by \(\text{ppz-csp}\). The overall success probability is given by

\[
\Pr[\text{ppz-csp}(F, \pi) \text{ is successful}] = \sum_{\alpha \in \text{sat}(F)} \Pr[\text{ppz-csp}(F, \pi) \text{ returns } \alpha].
\]

In the next section, we will bound \(6.3\) from below. The bound depends on the level of isolatedness of \(\alpha\): An assignment \(\alpha'\) is a neighbor of \(\alpha\) if \(\alpha\) and \(\alpha'\) disagree on only one variable. If \(\alpha\) has many satisfying neighbors, the probability to that \(\text{ppz-csp}\) returns \(\alpha\) decreases. However, the existence of many satisfying assignments will in turn increase the sum in \(6.5\). In the end, it turns out that the worst case happens if \(F\) has a unique satisfying assignment. Observe that
Algorithm 10 ppz-csp($F$: a $(d, \leq k)$-CSP formula over variables $V := \{x_1, \ldots, x_n\}, \pi$: a permutation of $V$)

1: $\alpha :=$ the empty assignment
2: for $i = 1, \ldots, n$ do
3:    $x := x_{\pi(i)}$
4:    $S(x, \pi) := \{c \in [d] \mid (x \neq c) \notin F\} // S(x, \pi)$ is the set of allowed values
5: if $S(x, \pi) = \emptyset$ then
6:    return failure
7: end if
8: $b \leftarrow_{u.a.r.} S(x, \pi)$
9: $\alpha := \alpha \cup [x \mapsto b]$
10: $F := F[x \mapsto b]$
11: end for
12: if $\alpha$ satisfies $F$ then
13:    return $\alpha$
14: else
15:    return failure
16: end if

for the ppz-csp-algorithm in the boolean case [PPZ99], the unique satisfiable case is also the worst case, whereas for the improved version ppzs [PPSZ05], it is not, or at least not known to be.

6.3 Analyzing the Success Probability

Preliminaries

We assume that all constraints consist of exactly $k$ literals. Our analysis also holds for the general case, but the notation would sometimes be more cumbersome. We fix a satisfying assignment $\alpha$. For simplicity, assume that $\alpha = (1, \ldots, 1)$, i.e. it sets every variable to 1. What is the probability that ppz-csp returns $\alpha$? For a permutation $\pi$ and a variable $x$, let $\beta$ be the partial assignment obtained by restricting $\alpha$ to the variables that come before $x$ in $\pi$, and define

$$S(x, \pi, \alpha) := \{c \in [d] \mid (x \neq c) \notin F^{[\beta]}\}.$$

Example. Let $d = 3, k = 2$, and $\alpha = (1, \ldots, 1)$. We consider

$$F = (x \neq 2 \vee y \neq 1) \wedge (x \neq 3 \wedge z \neq 1).$$
For \( \pi = (x, y, z) \), no value is forbidden when processing \( x \), therefore we have \( S(x, \pi, \alpha) = \{1, 2, 3\} \). For \( \pi' = (y, x, z) \) we consider the partial assignment that sets \( y \) to 1, obtaining

\[
F[y \rightarrow 1] = (x \neq 2) \land (x \neq 3 \lor z \neq 1),
\]

and \( S(x, \pi', \alpha) = \{1, 3\} \). Last, for \( \pi'' = (y, z, x) \) we set \( y \) and \( z \) to 1, obtaining

\[
F[y \rightarrow 1, z \rightarrow 1] = (x \neq 2) \land (x \neq 3),
\]

thus \( S(x, \pi'', \alpha) = \{1\} \). There are three more permutations we could analyze, but let us keep the example short. \( \square \)

Observe that \( S(x, \pi, \alpha) \) is non-empty: Let \( c = \alpha(x) \). It is not difficult to see that \( c \in S(x, \pi, \alpha) \). What has to happen in order for the algorithm to return \( \alpha \)? In every step of \( \text{ppz-csp} \), the value \( b \) selected in line 8 for variable \( x \) must be \( \alpha(x) \). Assume now that this was the case in each of the first \( i \) steps of the algorithm, i.e., the variables \( x_{\pi(1)}, \ldots, x_{\pi(i)} \) have been set to their respective values under \( \alpha \). Let \( x = x_{\pi(i+1)} \) be the variable processed in step \( i + 1 \). The set \( S(x, \pi, \alpha) \) coincides with the set \( S(x, \pi) \) of the algorithm, and therefore \( x \) is set to \( \alpha(x) \) with probability \( 1/|S(x, \pi, \alpha)| \). Since this holds in every step of the algorithm, we conclude that for a fixed permutation \( \pi \),

\[
\Pr[\text{ppz-csp}(F, \pi) \text{ returns } \alpha] = \prod_{x \in V} \frac{1}{|S(x, \pi, \alpha)|}.
\]

For \( \pi \) being chosen uniformly at random, we obtain

\[
\Pr[\text{ppz-csp}(F, \pi) \text{ returns } \alpha] = \mathbb{E}_{\pi} \left[ \prod_{x \in V} \frac{1}{|S(x, \pi, \alpha)|} \right].
\]

The expectation of a product is an uncomfortable term if the factors are not independent. The usual trick in this context is to apply Jensen’s inequality, hoping that we do not lose too much. We apply it with the convex function being \( f : x \mapsto 2^{-x} \) and the random variable being \( X = \sum_{x \in V} \frac{1}{\log_2 |S(x, \pi, \alpha)|} \). With this notation, \( f(X) = \prod_{x \in V} \frac{1}{|S(x, \pi, \alpha)|} \), the expectation of which we want to bound from below.

\[
\mathbb{E} \left[ \prod_{x \in V} \frac{1}{|S(x, \pi, \alpha)|} \right] \geq 2^{-\sum_{x \in V} \log_2 |S(x, \pi, \alpha)|} \geq 2^{-\sum_{x \in V} \mathbb{E} \left[ \log_2 |S(x, \pi, \alpha)| \right]}.
\]
Proposition 6.3. \( \Pr[\text{ppz-csp}(F, \pi)] \) returns \( \alpha \geq 2^{-\sum_{x \in V} E[\log_2 |S(x, \pi, \alpha)|]} \).

Example: The boolean case. In the boolean case, the set \( S(x, \pi, \alpha) \) is either \( \{1\} \) or \( \{0, 1\} \), and thus the logarithm is either 0 or 1. Therefore, the term \( E[\log_2 |S(x, \pi, \alpha)|] \) is the probability that the value of \( x \) is not determined by a unit clause, and thus has to be guessed. □

So far the calculations are the same as in the boolean case. This will not stay that way for long. In the boolean case, there are only two cases: Either the value of \( x \) is determined by a unit clause, or it is not. For \( d \geq 3 \), there are more cases: The set of potential values for \( x \) can be the full range \([d]\), it can be just the singleton \( \{1\} \), but it can also be anything in between, and even if the algorithm cannot determine the value of \( x \) by looking at unit clauses, it will still be happy if at least, say, \( d/2 \) values are forbidden by unit clauses.

Analyzing \( E[\log_2 |S(x, \pi, \alpha)|] \)

In this section we prove an upper bound on \( E[\log_2 |S(x, \pi, \alpha)|] \). We assume without loss of generality that \( \alpha = (1, \ldots, 1) \). There are \( d \) assignments \( \alpha_1, \ldots, \alpha_d \) agreeing with \( \alpha \) on the variables \( V \setminus \{x\} \): For a value \( c \in [d] \) we define \( \alpha_c := \alpha[x \mapsto c] \). Clearly, \( \alpha_1 = \alpha \) and \( \alpha_i \) is a neighbor of \( \alpha \) for \( 2 \leq i \leq d \).

Definition 6.4. The looseness of \( \alpha \) at \( x \), denoted by \( \ell(\alpha, x) \), is the number of assignments among \( \alpha_1, \ldots, \alpha_d \) that satisfy \( F \).

Since \( \alpha_1 = \alpha \) satisfies \( F \), the looseness of \( \alpha \) at \( x \) is at least 1, and since there are \( d \) possible values for \( x \), the looseness is at most \( d \). Thus \( 1 \leq \ell(\alpha, x) \leq d \). If \( \alpha \) is the unique satisfying assignment, then \( \ell(\alpha, x) = 1 \) for every \( x \). Note that \( \alpha \) being unique is sufficient, but not necessary: Suppose \( \alpha = (1, \ldots, 1) \) and \( \alpha' = (2, 2, 1, \ldots, 1) \) are the only two satisfying assignments. Then \( \ell(\alpha, x) = \ell(\alpha', x) = 1 \) for every variable \( x \).

Why are we considering the looseness \( \ell \) of \( \alpha \) at \( x \)? Suppose without loss of generality that the assignments \( \alpha_1, \ldots, \alpha_\ell \) satisfy \( F \), whereas \( \alpha_{\ell+1}, \ldots, \alpha_d \) do not. The set \( S(x, \pi, \alpha) \) is a random object depending on \( \pi \), but one thing is sure:

\[
\text{for all } \pi \text{ and } 1 \leq c \leq \ell(\alpha, x) : c \in S(x, \pi, \alpha) .
\]

For \( \ell(\alpha, x) < c \leq d \), what is the probability that \( c \in S(x, \pi, \alpha) \)? Since \( \alpha_c \) does not satisfy \( F \), there must be a constraint in \( F \) that is satisfied by \( \alpha \) but not by \( \alpha_c \). Since \( \alpha \) and \( \alpha_c \) disagree on \( x \) only, that constraint must be of the form

\[
(x \neq c \lor y_2 \neq 1 \lor y_3 \neq 1 \lor \cdots \lor y_k \neq 1) , \tag{6.6}
\]
for some $k - 1$ distinct variables $y_2, \ldots, y_k$. If the variables $y_2, \ldots, y_k$ come before $x$ in the permutation $\pi$, then $c \notin S(x, \pi, \alpha)$: This is because after setting to 1 the variables that come before $x$, the constraint in (6.6) has been reduced to $x \neq c$. Note that $y_2, \ldots, y_k$ coming before $x$ is sufficient for $c \notin S(x, \pi, \alpha)$, but not necessary, since there could be multiple constraints of the form (6.6). With probability at least $1/k$, all variables $y_2, \ldots, y_k$ come before $x$, and we conclude:

**Proposition 6.5.** If $\alpha_c$ does not satisfy $F$, then $\Pr[c \in S(x, \pi, \alpha)] \leq 1 - 1/k$.

This proposition is nice, but not yet useful on its own. We can use it to finish the analysis of the running time, however we will end up with a suboptimal estimate.

**A suboptimal analysis of $\text{ppz-csp}$**

The function $t \mapsto \log_2(t)$ is concave. We apply Jensen’s inequality to conclude that

$$
\mathbb{E}[\log_2 |S(x, \pi, \alpha)|] \leq \log_2 (\mathbb{E}[|S(x, \pi, \alpha)|])
$$

$$
= \log_2 \left( \sum_{c=1}^{n} \Pr[c \in S(x, \pi, \alpha)] \right)
$$

We apply what we have learned above: For $c = 1, \ldots, \ell(\alpha, x)$, it always holds that $c \in S(x, \pi, \alpha)$, and for $c = \ell(\alpha, x) + 1, \ldots, d$, we have computed that $\Pr[c \in S(x, \pi, \alpha)] \leq 1 - 1/k$. Therefore

$$
\mathbb{E}[\log_2 |S(x, \pi, \alpha)|] \leq \log_2 \left( \ell(\alpha, x) + (d - \ell(\alpha, x)) \left(1 - \frac{1}{k}\right) \right).
$$

**The unique case.** If $\alpha$ is the unique satisfying assignment, then $\ell(\alpha, x) = 1$ for every variable $x$ in our CSP formula $F$, and the above term becomes

$$
\log_2 \left( 1 + \frac{(d - 1)(k - 1)}{k} \right) = \log_2 \left( \frac{d(k - 1) + 1}{k} \right).
$$

We plug this into the bound of Proposition 6.3:

$$
\Pr[\text{ppz-csp returns } \alpha] \geq 2^{-\sum_{i=1}^{n} \mathbb{E}[\log_2 |S(x_i, \pi, \alpha)|]} \geq 2^{-n \log_2 \left( \frac{d(k - 1) + 1}{k} \right)} = \left( \frac{d(k - 1) + 1}{k} \right)^{-n}.
$$
The success probability of Schöning’s algorithm for \((d,k)\)-CSP problems is \( \left( \frac{d(k-1)}{k} \right)^n \), and we see that even for the unique case, our analysis of \( \text{ppz-csp} \) does not yield anything better than Schöning. Discouraged by this failure, we do not continue this suboptimal analysis for the non-unique case.

**Detour: Jensen’s Inequality Here, There, and Everywhere**

In this section we offer a glimpse behind the scenes of this proof. This will be instructive, but the reader can also skip to ”A Better Analysis” without missing anything that would be essential for the proof.

The main culprit behind the poor performance the above suboptimal analysis is Jensen’s inequality in (6.8). To improve our analysis, we refrain from applying Jensen’s inequality there and instead try to analyze \( E[\log_2 |S(x, \pi, \alpha)|] \) directly. However, recall that we have used Jensen’s inequality before, in (6.5). Is it safe to apply it there? How can we tell when applying it makes sense and when it definitely does not? To discuss this issue, we restate the two applications of Jensen’s inequality:

\[
E \left[ 2^{\sum_{x \in V} \log_2 |S(x, \pi, \alpha)|} \right] \geq 2^E \left[ \sum_{x \in V} \log_2 |S(x, \pi, \alpha)| \right] \tag{6.9}
\]

\[
E[\log_2 |S(x, \pi, \alpha)|] \leq \log_2 (E[|S(x, \pi, \alpha)|]) \tag{6.10}
\]

Jensen’s inequality states that for a random variable \( X \) and a convex function \( f \), it holds that

\[
E[f(X)] \geq f(E[X]) \tag{6.11}
\]

and by multiplying (6.11) by \(-1\) one obtains a similar inequality for concave functions. As a rule of thumb, Jensen’s inequality is pretty tight if \( X \) is very concentrated around its expectation: In the most extreme case, \( X \) is a constant, and (6.11) holds with equality. On the other extreme, suppose \( X \) is a random variable taking on values \(-m\) and \( m \), each with probability \(1/2\), and let \( f : t \mapsto t^2 \), which is a convex function. The left-hand side of (6.11) evaluates to \( E[f(X)] = E[X^2] = m^2 \), whereas the right-hand side evaluates to \( f(E[X]) = f(0) = 0 \), and Jensen’s inequality is very loose indeed. What random variables are we dealing with in (6.9) and (6.10)? These are

\[
X := \sum_{x \in V} \log_2 |S(x, \pi, \alpha)| \quad \text{and} \quad Y := |S(x, \pi, \alpha)|,
\]
and the corresponding functions are \( f : t \mapsto 2^{-t} \), which is convex, and \( g : t \mapsto \log_2 t \), which is concave. In both cases, the underlying probability space is the set of all permutations of \( V \), endowed with the uniform distribution. We see that \( Y \) is not concentrated at all: Suppose \( x \) comes first in \( \pi \): Since our \((d, k)\)-CSP formula \( F \) has no unit constraints, it holds that \( |S(x, \pi, \alpha)| = d \). On the other hand, if \( x \) comes last in \( \pi \), then \( |S(x, \pi, \alpha)| = \ell(\alpha, x) \). Either case happens with probability \( 1/n \), which is not very small. Thus, the random variable \( |S(x, \pi, \alpha)| \) does not seem to be very concentrated.

Contrary to \( Y \), the random variable \( X \) can be very concentrated, in fact for certain CSP formulas it can be a constant: Suppose \( d = 2 \), i.e., the boolean case. Consider the 2-CNF formula

\[
\land_{i=1}^{n/2} (x_i \lor y_i) \land (x_i \lor \overline{y}_i) \land (\overline{x}_i \lor y_i).
\]  

(6.12)

This formula has \( n \) variables, and \( \alpha = (1, \ldots, 1) \) is the unique satisfying assignment. Observe that if \( x_i \) comes before \( y_i \) in \( \pi \), then \( S(x_i, \pi, \alpha) = \{0, 1\} \) and \( S(y_i, \pi, \alpha) = \{1\} \). If \( y_i \) comes before \( x_i \), then \( S(x_i, \pi, \alpha) = \{1\} \) and \( S(y_i, \pi, \alpha) = \{0, 1\} \). Hence \( X = n/2 \) is a constant. Readers who balk at the idea of supplying a 2-CNF formula as an example for an exponential-time algorithm may try to generalize (6.12) for values of \( k \geq 3 \).

A Better Analysis

After this interlude on Jensen’s inequality, let us try to bound the expectation \( \mathbb{E}[\log_2 |S(x, \pi, \alpha)|] \) directly. In this context, \( x \) is some variable, \( \alpha \) is a satisfying assignment, for simplicity \( \alpha = (1, \ldots, 1) \), and \( \pi \) is a permutation of the variables sampled uniformly at random. Again think of the \( d \) assignments \( \alpha_1, \ldots, \alpha_d \) obtained by setting \( \alpha_c := \alpha[x \mapsto c] \) for \( c = 1, \ldots, d \). Among them, \( \ell := \ell(\alpha, x) \) satisfy the formula \( F \). We assume without loss of generality that those are \( \alpha_1, \ldots, \alpha_{\ell} \). Thus, for each \( \ell < c \leq d \), there is a constraint \( C_c \) satisfied by \( \alpha \) but not by \( \alpha_c \). Let us write down these constraints:

\[
C_{\ell+1} := (x \neq \ell + 1 \lor y_1^{(\ell+1)} \neq 1 \lor \cdots \lor y_{k-1}^{(\ell+1)} \neq 1)
\]
\[
C_{\ell+2} := (x \neq \ell + 2 \lor y_1^{(\ell+2)} \neq 1 \lor \cdots \lor y_{k-1}^{(\ell+2)} \neq 1)
\]
\[
\vdots
\]
\[
C_d := (x \neq d \lor y_1^{(d)} \neq 1 \lor \cdots \lor y_{k-1}^{(d)} \neq 1)
\]

(6.13)

Recall that for every \( \ell + 1 \leq c \leq d \), the \( k - 1 \) variables \( y_1^{(\ell+1)}, \ldots, y_{k-1}^{(\ell+1)} \) are distinct, but it is perfectly possible that some variable \( y \) appears in several
6.3. Analyzing the Success Probability

rows. We define binary random variables $Y_j^{(c)}$ for $1 \leq j \leq k - 1$ and $\ell + 1 \leq c \leq d$ as follows:

$$Y_j^{(c)} := \begin{cases} 1 & \text{if } y_j^{(c)} \text{ comes after } x \text{ in the permutation } \pi, \\ 0 & \text{otherwise} \end{cases}$$

We define $Y^{(c)} := \max(Y_1^{(c)}, \ldots, Y_{k-1}^{(c)})$. For convenience we also introduce random variables $Y^{(1)}, \ldots, Y^{(\ell)}$ that are constant 1. Finally, we define $Y := \sum_{c=1}^{d} Y^{(c)}$. Observe that $Y^{(c)} = 0$ if and only if all variables $y_1^{(c)}, \ldots, y_{k-1}^{(c)}$ come before $x$ in the permutation, in which case $c \notin S(x, \pi, \alpha)$. Therefore,

$$|S(x, \pi, \alpha)| \leq Y \quad (6.14)$$

The variables $Y^{(1)}, \ldots, Y^{(\ell)}$ are constant 1, whereas each of the $Y^{(c+1)}, \ldots, Y^{(d)}$ is 0 with probability at least $1/k$. Since $1 \leq \ell \leq d$, the random variable $Y$ can take values from 1 to $d$. We want to bound

$$E[\log_2 |S(x, \alpha, \pi)|] \leq E[\log_2(Y)] = E \left[ \log_2 \left( \ell + \sum_{c=\ell+1}^{d} Y^{(c)} \right) \right]. \quad (6.15)$$

For this, we must bound the probability $Pr[Y = j]$ for $j = 1, \ldots, d$. This is difficult, since the $Y^{(c)}$ are not independent: For example, conditioning on $x$ coming very early in $\pi$ increases the expectation of each $Y^{(c)}$, and conditioning on $x$ coming late decreases it. We use a standard trick, also used by Paturi, Pudákov, Saks, and Zane \cite{PPSZ05} to overcome these dependencies: Instead of viewing $\pi$ as a permutation of $V$, we think of it as a function $V \to [0, 1]$ where for each $x \in V$, its value $\pi(x)$ is chosen uniformly at random from $[0, 1]$. With probability 1, all values $\pi(x)$ are distinct and therefore give rise to a permutation. The advantage is that for $x, y,$ and $z$ being three distinct variables, the events “$y$ comes before $x$” and “$z$ comes before $x$” are independent when conditioning on $\pi(x) = r$:

$$Pr[\pi(y) < \pi(x) \mid \pi(x) = r] = r$$
$$Pr[\pi(z) < \pi(x) \mid \pi(x) = r] = r$$
$$Pr[\pi(x) < \pi(x) \text{ and } \pi(z) < \pi(x) \mid \pi(x) = r] = r^2$$

79
Compare this to the unconditional probabilities:

\[
\begin{align*}
Pr[\pi(y) < \pi(x)] &= \frac{1}{2} \\
Pr[\pi(z) < \pi(x)] &= \frac{1}{2} \\
Pr[\pi(x) < \pi(x) \text{ and } \pi(z) < \pi(x)] &= \frac{1}{3}
\end{align*}
\]

From now on we investigate the random variables \(Y^{(c)}\) conditioned on \(\pi(x) = r\) for some \(r \in [0, 1]\). With this condition, \(Y^{(c)}\) is the maximum of \(k - 1\) independent binary variables, each with expectation \(1 - r\) for \(\ell + 1 \leq c \leq d\) and with expectation 1 for \(1 \leq c \leq \ell\). Therefore, \(Y^{(c)}\) is a binary random variable, too, and for \(\ell + 1 \leq c \leq d\) we observe that

\[
Pr[Y^{(c)} = 1 \mid \pi(x) = r] = 1 - r^{k-1}.
\]

Still, a variable \(y_j^{(c)}\) might occur in several of the constraints \(C_{\ell+1}, \ldots, C_d\), and therefore the variables \(Y^{(c)}\) are not independent. The main technical tool of our analysis is a lemma stating that the worst case is achieved exactly if they in fact are independent, i.e., if all the variables \(y_j^{(c)}\) for \(c = \ell + 1, \ldots, d\) and \(j = 1, \ldots, k - 1\) are distinct.

**Lemma 6.6 (Independence is the Worst Case).** Let \(r, k, \ell\) and \(Y^{(c)}\) be defined as above. Introduce \((d - \ell)(k - 1)\) independent binary random variables \(Z_j^{(c)}, \ell + 1 \leq c \leq d, 1 \leq j \leq k - 1,\) each with expectation \(1 - r\). Let \(Z^{(c)} := \max(Z_1^{(c)}, \ldots, Z_{k-1}^{(c)})\). Then \(\mathbb{E}[Z^{(c)}] = 1 - r^{k-1}\) and

\[
\mathbb{E} \left[ \log_2 \left( \ell + \sum_{c=\ell+1}^{d} Y^{(c)} \right) \mid \pi(x) = r \right] \leq \mathbb{E} \left[ \log_2 \left( \ell + \sum_{c=\ell+1}^{d} Z^{(c)} \right) \right].
\]

The proof of this lemma is elementary but non-trivial and we will give it in the next section. Let us finish the analysis of the algorithm. We apply a somewhat peculiar estimate: Let \(a \geq 1\) and \(b \geq 0\) be integers. Then \(\log_2(a + b) \leq \log_2(a \cdot (b + 1)) = \log_2(a) + \log_2(b + 1)\). Applying this with \(a := \ell\) and \(b := \sum_{c=\ell+1}^{d} Z^{(c)}\) and combining it with Lemma 6.6 and with (6.15) on page 79, we obtain

\[
\mathbb{E}[\log_2 |S(x, \alpha, \pi)| \mid \pi(x) = r] \leq \log_2(\ell) + \mathbb{E} \left[ \log_2 \left( 1 + \sum_{c=\ell+1}^{d} Z^{(c)} \right) \right]. \quad (6.16)
\]
In addition to \(Z^{(\ell+1)}, \ldots, Z^{(d)}\), we introduce \(\ell - 1\) new independent binary random variables \(Z^{(2)}, \ldots, Z^{(\ell)}\), each with expectation \(1 - r^{k-1}\), and define

\[
g(d, k, r) := \mathbb{E}\left[\log_2 \left(1 + \sum_{c=2}^{d} Z^{(c)}\right)\right].
\]

The only difference between the expectation in (6.16) and here is that here, we sum over \(c = 2, \ldots, d\), whereas in (6.16) we sum only over \(c = \ell + 1, \ldots, d\). We get the following version of (6.16):

\[
\mathbb{E}[\log_2 |S(x, \alpha, \pi)| \mid \pi(x) = r] \leq \log_2(\ell) + g(d, k, r).
\]

We get rid of the condition \(\pi(x) = r\) by integrating (6.17) for \(r\) from 0 to 1.

\[
\mathbb{E}[\log_2 |S(x, \alpha, \pi)|] \leq \log_2(\ell) + \int_0^1 g(d, k, r)dr = \log_2(\ell) + G(d, k),
\]

For the value \(G(d, k)\) we have defined in (6.2) on page 70.

**Lemma 6.7** (Lemma 1 in Feder, Motwani [FM02]). Let \(F\) be a satisfiable CSP formula over variable set \(V\). Then

\[
\sum_{\alpha \in \text{sat}_V(F)} \prod_{x \in V} \frac{1}{\ell(\alpha, x)} \geq 1.
\]

This lemma is a quantitative version of the intuitive statement that if a set \(S \subseteq [d]^n\) is small, then there must be rather isolated points in \(S\). We now put everything together:

\[
\Pr[\text{ppsz}(F, \pi) \text{ is successful}] = \sum_{\alpha \in \text{sat}_V(F)} \Pr[\text{ppsz}(F, \pi) \text{ returns } \alpha]
\geq \sum_{\alpha \in \text{sat}_V(F)} 2^{-\sum_{x \in V} \mathbb{E}[\log_2 |S(x, \alpha, \pi)|]},
\]

where the inequality follows from (6.5) on page 74. Together with (6.18) on
page 81, we see that
\[
\sum_{\alpha \in \text{sat}_V(F)} 2^{-\sum_{x \in V} \log_2 |S(x,\alpha,\pi)|} \geq \sum_{\alpha \in \text{sat}_V(F)} 2^{-\sum_{x \in V} \log_2 (\ell(\alpha,x)) + G(d,k)}
\]
\[
= 2^{-nG(d,k)} \sum_{\alpha \in \text{sat}_V(F)} 2^{-\sum_{x \in V} \log_2 (\ell(\alpha,x))}
\]
\[
= 2^{-nG(d,k)} \sum_{\alpha \in \text{sat}_V(F)} \prod_{x \in V} \frac{1}{\ell(\alpha,x)}
\]
\[
\geq 2^{-nG(d,k)} ,
\]
where the last inequality follows from Lemma 6.7. This shows that with probability at least \(2^{-nG(d,k)}\), the algorithm returns some satisfying assignment. This finishes the proof of Theorem 6.1 on page 70.

6.4 A Correlation Inequality

Our goal is to prove Lemma 6.6 on page 80. Recall that we defined
\[
Y_j^{(c)} := \begin{cases} 
1 & \text{if } y_j^{(c)} \text{ comes after } x \text{ in the permutation } \pi , \\
0 & \text{otherwise},
\end{cases}
\]
where \(y_j^{(c)}\) are the variables occurring in the critical constraints for a specific variable \(x\), and \(Y^{(c)} := \max(Y_1^{(c)}, \ldots, Y_{k-1}^{(c)})\). For clarity let us repeat Lemma 6.6 on page 80:

**Lemma 6.8 (Independence is the Worst Case).** Let \(r, k, \ell \) and \(Y^{(c)}\) be defined as above. Introduce \((d - \ell)(k - 1)\) independent binary random variables \(Z_j^{(c)}, \ell + 1 \leq c \leq d, 1 \leq j \leq k - 1\), each with expectation \(1 - r\). Let \(Z^{(c)} := \max(Z_1^{(c)}, \ldots, Z_{k-1}^{(c)})\). Then \(\mathbb{E}[Z^{(c)}] = 1 - r^{k-1}\) and

\[
\mathbb{E} \left[ \log_2 \left( \ell + \sum_{c=\ell+1}^d Y^{(c)} \right) \mid \pi(x) = r \right] \leq \mathbb{E} \left[ \log_2 \left( \ell + \sum_{c=\ell+1}^d Z^{(c)} \right) \right]. \tag{6.20}
\]

On the right side of (6.20) we feed in \((k - 1)(d - \ell)\) independent random bits, while for the left side we may use fewer, since several of the \(Y_j^{(c)}\) may be defined by the same underlying \(d\)-ary variable \(y_j^{(c)}\) of our formula.
The intuition is that independence of the underlying random bits causes the sum \( \sum_{c=\ell+1}^d Z^{(c)} \) to be a “more concentrated” random variable than the sum \( \sum_{c=\ell+1}^d Y^{(c)} \), and since \( t \mapsto \log_2(\ell + t) \) is a concave function, a more concentrated random variable leads to a larger expectation. This intuition is made precise in the following lemma.

**Lemma 6.9.** Let \( X, Y, Y', U, V \) be finite vectors of real random variables, all independent, taking only finitely many values. Suppose \( Y \) and \( Y' \) are over \( \mathbb{R}^n \) and have the same distribution. Let \( f, g : \mathbb{R}^n \to \mathbb{R} \) be monotonically increasing functions, and let \( h : \mathbb{R} \to \mathbb{R} \) be a concave function. Then

\[
E\left[ h(f(X, Y, U) + g(X, Y, V)) \right] \leq E\left[ h(f(X, Y, U) + g(X, Y', V)) \right].
\]

Think of the left hand and right hand side as two different random experiments. The functions \( f \) and \( g \) are fed with random input. The vector \( X \) represents shared random input, \( U \) and \( V \) are individual random input. The vectors \( Y \) and \( Y' \) represent random input that is shared in the first (left hand) experiment and made individual in the second (right hand) experiment. Intuitively, the more shared randomness \( f \) and \( g \) use, the likelier they are to be both large or both small, making the argument to \( h \) vary more wildly and, due to the concavity of \( h \), making the expected value of \( h \) smaller.

By repeatedly using Lemma 6.9 we can take the left hand side of (6.20), replace each \( Y_j^{(c)} \) by \( Z_j^{(c)} \) one by one, thus eventually obtaining the inequality (6.20). Here we implicitly use several simple facts: First, the sum on the left-hand side in (6.20), involving \( d - \ell \) terms, can be partitioned into two terms, one of which is again a sum. These two terms play the role of \( f(X, Y, U) \) and \( g(X, Y, V) \), respectively. The same holds for the right-hand side. Second, the function \( \max \), playing the role of both \( f \) and \( g \), is certainly monotonically increasing. Third, the function \( h : t \mapsto \log_2(\ell + t) \) is concave. Observe that although \( h \) is not a function \( \mathbb{R} \to \mathbb{R} \), we can find a concave function defined on all of \( \mathbb{R} \) that agrees with \( h \) on \([1, \infty)\), which is the range that matters. It remains to prove Lemma 6.9.

**Proof of Lemma 6.9.** First we claim that the conclusion of the lemma holds for any particular choice of \( X, U, \) and \( V \), not only when taking expectation over these three vectors. If we can show this, the lemma follows. By substituting concrete real vectors for \( X, U, \) and \( V \), we basically obtain functions \( f' \) and \( g' \) that depend only on \( Y \) and are still monotone. Hence it suffices to prove that

\[
E[h(f(Y) + g(Y))] \leq E[h(f(Y) + g(Y'))],
\]

where \( Y \) and \( Y' \) are over \( \mathbb{R}^n \) and \( f, g : \mathbb{R}^n \to \mathbb{R} \) are monotonically increasing functions.
Second, we claim that it suffices to examine the case \( n = 1 \). If we can the lemma for \( n = 1 \), we can repeatedly replace every component \( Y_i \) of \( Y \) by \( Y_i' \), eventually obtaining the above inequality. Hence let us assume \( n = 1 \), and \( Y, Y' \) are independent identically distributed real random variables. By symmetry it holds that \( \mathbb{E}[h(f(Y) + g(Y))] = \mathbb{E}[h(f(Y') + g(Y'))] \) and \( \mathbb{E}[h(f(Y) + g(Y'))] = \mathbb{E}[h(f(Y) + g(Y'))] \). Multiplying (6.21) by 2 and using these identities, we obtain

\[
\mathbb{E}[h(f(Y) + g(Y))] \leq \mathbb{E}[h(f(Y') + g(Y'))] \\
\iff \quad 2\mathbb{E}[h(f(Y) + g(Y))] \leq 2\mathbb{E}[h(f(Y') + g(Y'))] \\
\iff \quad \mathbb{E}[h(f(Y) + g(Y))] + \mathbb{E}[h(f(Y') + g(Y'))] \\
\leq \mathbb{E}[h(f(Y) + g(Y'))] + \mathbb{E}[h(f(Y') + g(Y))] \\
\iff \quad \mathbb{E}[h(f(Y) + g(Y)) + h(f(Y') + g(Y'))] \\
\leq \mathbb{E}[h(f(Y) + g(Y')) + h(f(Y') + g(Y))] .
\]

Conveniently, the last inequality holds not only in expectation, but for every particular choice of \( Y, Y' \in \mathbb{R} \): Fix such a choice. Without loss of generality, we can assume that \( Y \leq Y' \) and therefore \( f(Y) \leq f(Y') \) and \( g(Y) \leq g(Y') \), using monotonicity. Second, we can assume without loss of generality that \( f(Y) + g(Y') \leq f(Y') + g(Y) \). Writing \( a := f(Y) + g(Y), b := f(Y) + g(Y'), c := f(Y') + g(Y), \) and \( d := f(Y') + g(Y') \) we see that \( a \leq b \leq c \leq d \). The claimed inequality states that \( h(a) + h(d) \leq h(b) + h(c) \). Divided by two, the statement becomes equivalent to saying that the midpoint of the longer line segment in Figure 6.3 is below the one of the shorter. This follows immediately from \( h \) being a concave function. \( \Box \)
6.4. A Correlation Inequality

Figure 6.3: Illustration of the final inequality in the proof of Lemma 6.9.
Part II

Extremal Combinatorics of CNF Formulas
Chapter 7

The Conflict Structure of CNF Formulas

A neutron walks into a bar and asks how much for a drink. The bartender replies “for you, no charge”.

Dr. Sheldon Cooper in The Big Bang Theory

7.1 Introduction

We investigate several questions concerning the conflict structure of CNF formulas. As outlined in the introduction to this thesis, we have some complexity measure $\mu$ on CNF formulas and want to answer the question

What is the largest integer $d$ such that every $k$-CNF formula $F$ with $\mu(F) \leq d$ is satisfiable?

This question is very generic. In this chapter, we let $\mu$ be a function that measures the conflict structure of $F$ in several ways. The extremal parameter we seek to determine will be a function of $k$. This stands in contrast to the algorithmic context, where typically $k$ is a constant and the number of variables tends to infinity. Here, we are interested in the asymptotic behavior of several extremal parameters as $k$ grows.
Conflicts. Two clauses $C$ and $D$ have a conflict if $C \cap \overline{D} \neq \emptyset$. That is, if there is a literal $u$ such that $u \in C$ and $\overline{u} \in D$. For example, the clauses $\{x, y, z\}$ and $\{\overline{x}, y, u\}$ have a conflict, but $\{x, y, z\}$ and $\{x, u, v\}$ do not.

Definition 7.1 (Conflict Neighborhood). For a CNF formula $F$ and a clause $C$, the set

$$\Gamma'_F(C) := \{D \in F \mid C \cap \overline{D} \neq \emptyset\}$$

is the conflict neighborhood of $C$. By $\text{lc}(F)$ we denote the size of the largest conflict neighborhood of a clause in $F$, i.e.,

$$\text{lc}(F) = \max\{|\Gamma'_F(C)| \mid C \in F\} .$$

For the empty formula $\emptyset$ we define $\text{lc}(\emptyset) = 0$.

Note that $\text{lc}(F) = 0$ if and only no two clauses in $F$ have a conflict. In this case, one can satisfy the clauses of $F$ one by one and never run into a contradiction, unless $F$ contains the empty clause.

Observation 7.2. If $F$ is a CNF formula that does not contain the empty clause and $\text{lc}(F) = 0$, then $F$ is satisfiable.

We define the conflict graph of a CNF formula $F$ as follows: Its vertices are the clauses of $F$, and two clauses $C$ and $D$ are connected by an edge if $C$ and $D$ have a conflict. See Figure 7.1 for an illustration.

1-Conflicts. There is an alternative definition of conflicts, which looks strange at the beginning but turns out to be very fruitful. We say two clauses $C$ and $D$ have a 1-conflict if $|C \cap \overline{D}| = 1$, i.e., if there is a unique literal $u$ such that

Figure 7.1: The conflict graph of $\{\{\overline{x}\}, \{x, \overline{y}, z\}, \{x, \overline{y}, \overline{z}\}, \{x, y, \overline{z}\}, \{y, z\}\}$. 
Figure 7.2: The 1-conflict graph \{\{\bar{x}\}, \{x, \bar{y}, z\}, \{x, \bar{y}, \bar{z}\}, \{x, y, \bar{z}\}, \{y, z\}\}. 

\(u \in C\) and \(\bar{u} \in D\). Note that \(C\) and \(D\) have a 1-conflict if and only if their resolvent exists and is non-trivial. Clearly, any 1-conflict is a conflict, but not vice versa. For example, the clauses \(\{x, y, z\}\) and \(\{\bar{x}, \bar{y}, u\}\) have a conflict, but no 1-conflict. In analogy to Definition 7.1 we introduce some notation.

**Definition 7.3 (1-Conflict Neighborhood).** If \(F\) is a CNF formula and \(C\) is a clause, then 

\[
\Gamma^1_F(C) := \{D \in F \mid |C \cap \bar{D}| = 1\}
\]

is the 1-conflict neighborhood of \(C\). By \(lc_1(F)\) we denote the size of the largest conflict neighborhood of a clause in \(F\), i.e.,

\[
lc_1(F) = \max\{|\Gamma^1_F(C)| \mid C \in F\}.
\]

For the empty formula \(\{}\) we define \(lc_1(\{\}) = 0\).

The reader might wonder whether there is an analogous version of Observation 7.2 for the notion of 1-conflicts. There is, but it is less obvious and requires a proof. We will formally state it below in Proposition 7.9 on page 99.

Let us define the 1-conflict graph of a CNF formula \(F\). Again, its vertices are the clauses of \(F\), and two clauses \(C\) and \(D\) are connected by an edge if \(C\) and \(D\) have a 1-conflict. The 1-conflict graph is a spanning subgraph of the conflict graph (it has the same vertex set but possibly fewer edges). See Figure 7.2 for an illustration. Sometimes it is helpful to describe more precisely which literal gives rise to a conflict between clauses. For this reason, we define the conflict neighborhood of \(C\) generated by literal \(u\): For a CNF formula \(F\) and a literal \(u\) we write

\[
\Gamma_F(u) := \{C \in F \mid u \in C\},
\]

\[
\Gamma'_F(u) := \{C \in F \mid \bar{u} \in C\}.
\]
Thus $\Gamma'_F(u) = \Gamma_F(\bar{u})$. With this notation, it holds that

$$\Gamma'_F(C) = \bigcup_{u \in C} \Gamma'_F(u). \quad \text{(7.1)}$$

Note that in general, a conflict can be generated by several literals: For example, the clauses $C = \{x, y\}$ and $D = \{\bar{x}, \bar{y}\}$ have a conflict, generated by both $x$ and $y$. Therefore, the union in (7.1) is in general not a disjoint union. We define the 1-conflict neighborhood of a clause $C$ with respect a literal $u \in C$ as

$$\Gamma^1_F(C, u) := \{D \in F \mid C \cup \bar{D} = \{u\}\}.$$

The analog of (7.1) for the 1-conflict neighborhood also holds, and here the union is in fact a disjoint union:

$$\Gamma^1_F(C) = \bigcup_{u \in C} \Gamma^1_F(C, u).$$

### Extremal Parameters

In Observation 7.2, we have seen that a formula with no conflicts is satisfiable (unless it contains the empty clause). How large can we allow $\text{lc}(F)$ to be and still guarantee satisfiability? In other words,

For $k \in \mathbb{N}$, what is the largest $c \in \mathbb{N}$ such that every $k$-CNF formula $F$ with $\text{lc}(F) \leq c$ is satisfiable?

Let us denote this number by $\text{lc}(k)$. It follows from the *lopsided Lovász Local Lemma* [ES91, AS00, LS07] that $\text{lc}(k) \geq \lfloor 2^k/e \rfloor - 1$. What about upper bounds? Consider the complete formula $CF(x_1, \ldots, x_k)$. We defined this to be the $k$-CNF formula consisting of all $2^k$ clauses over the variables $x_1, \ldots, x_k$. Since $\text{lc}(CF(x_1, \ldots, x_k)) = 2^k - 1$, we know $\text{lc}(k)$ up to a constant factor. In fact, by techniques developed by Gebauer [Geb09], the upper bound can be improved to $\text{lc}(k) < 2^{k-1}$. See Gebauer, Moser, Schöder and Welzl [GMSW09] for details.

In analogy to $\text{lc}(k)$, we define $\text{lc}_1(k)$ to be the greatest integer $c$ such that any $k$-CNF formula $F$ with $\text{lc}_1(F) \leq c$ is satisfiable. Considering the complete formula $CF(x_1, \ldots, x_k)$, we conclude that $\text{lc}_1(k) \leq k - 1$: The formula is unsatisfiable, and every clause $C$ has one 1-conflicts with each of the $k$ clauses in which exactly one literal has opposite polarity as in $C$. We will see in Section 7.2 that this is tight: Any $k$-CNF formula $F$ with $\text{lc}_1(F) \leq k - 1$ is satisfiable. This proof is rather simple and does not use the Local Lemma.
Variable and Literal Degree. Recall \(\varepsilon _x(\text{deg}, k\text{-CNF})\), the extremal parameter defined in the introduction. This is the greatest integer \(d\) such that any \(k\text{-CNF}\) formulas \(F\) with \(\text{deg}(F) \leq d\) is satisfiable. In the literature, this number is often simply denoted by \(f(k)\). Kratochvıl, Savický, and Tuza \cite{KST93} showed that \(f(k) \geq \left\lfloor 2^k / ek \right\rfloor\) using the Lovász Local Lemma. Therefore, every unsatisfiable \(k\text{-CNF}\) formula \(F\) must have \(\text{deg}(F) \geq 2^k / (ek)\). Gebauer, Szabó, and Tardos \cite{GST10} construct unsatisfiable \(k\text{-CNF}\) formulas achieving this bound up to lower-order terms. Recall that for a formula \(F\) and a literal \(u\) we defined \(\text{occ}_F(u) := |\{C \in F \mid u \in C\}|\). What happens if for all variables \(x \in \text{vbl}(F)\) we restrict \(\text{occ}_F(\bar{x})\) more severely than \(\text{occ}_F(x)\)? For example, if we require that \(\text{occ}_F(\bar{x}) \leq 1.2^k\) and \(\text{occ}_F(x) \leq 3^k\) for every \(x \in \text{vbl}(F)\)? Does this imply that \(F\) is satisfiable? This question will be partially answered below in Theorem 7.20 on page 108. I suspect that every unsatisfiable \(k\text{-CNF}\) formula \(F\) with \(\text{deg}(F)\) not much larger than \(2^k\) (for example \(\text{deg}(F) \leq 2.01^k\)) has a variable \(x\) for which both \(\text{occ}_F(x)\) and \(\text{occ}_F(\bar{x})\) are not much smaller than \(2^k\) (for example \(\text{occ}_F(x), \text{occ}_F(\bar{x}) \geq 1.9^k\)). This certainly holds for the complete formula as well as for the formulas constructed in \cite{GST10}. To investigate this question, we introduce the notion of individual conflicts.

Individual Conflicts. We investigate the term \(\text{occ}_F(x) \cdot \text{occ}_F(\bar{x})\) and define
\[
ic(F) := \max\{\text{occ}_F(x) \cdot \text{occ}_F(\bar{x}) \mid x \in \text{vbl}(F)\}.
\]
If \(\text{vbl}(F) = \emptyset\), we define \(\nic(F) = 0\). Here, \(\nic\) stands for individual conflicts, since we are counting conflicts that are generated by an individual variable \(x\).

For \(k \in \mathbb{N}\), what is the maximum \(c \in \mathbb{N}\) such that every \(k\text{-CNF}\) formula \(F\) with \(\nic(F) \leq c\) is satisfiable?

We denote this number by \(\nic(k)\). For \(F = CF(x_1, \ldots, x_k)\) we have \(\nic(F) = (2^{k-1})^2 = 4^{k-1}\): Every variable appears \(2^{k-1}\) times positively and \(2^{k-1}\) times negatively. The formulas by Gebauer et al. \cite{GST10} are only slightly better: For \(F\) being a \(k\text{-CNF}\) formula constructed there, there is a variable \(x\) with \(\text{deg}_F(x) \geq 2^{k+1} / (ek)\), and this variable is balanced, i.e., \(\text{occ}_F(x) = \text{occ}_F(\bar{x})\). Therefore \(\nic(F) \in \Theta(4^k/k^2)\). This shows that \(\nic(k) \in O(4^k/k^2)\). As for lower bounds, the bound on \(f(k)\) implies that a \(k\text{-CNF}\) formula \(F\) with \(\nic(F) \leq f(k) - 1\) is satisfiable. Since \(f(k) \in \Theta(2^k/k)\), this proves \(\nic(k) \in \Omega(2^k/k)\). Note that there is an exponential gap between these upper and lower bounds for \(\nic(k)\)!

Global Conflicts. What if we do not only restrict the local number of conflicts or the number of conflicts generated by an individual variable, but the
global number of conflicts in a formula? How many conflicts can we allow and still guarantee satisfiability? To be more formal,

\[ gc(F) := \left\{ \{C, D\} \in \binom{F}{2} \mid C \cap \bar{D} \neq \emptyset \right\} . \]

This is the number of edges in the conflict graph of \( F \), and \( \sum_{C \in F} |\Gamma'_F(C)| = 2gc(F) \).

For \( k \in \mathbb{N} \), what is the maximum \( c \) such that every \( k \)-CNF formula \( F \) with \( gc(F) \leq c \) is satisfiable?

We denote this maximum \( c \) by \( gc(k) \). Again, we consider the usual suspects. The complete \( k \)-CNF formula has \( 2^k \) clauses, and any two clauses have a conflict, therefore \( gc(CF(x_1, \ldots, x_k)) = \binom{2^k}{2} \in \Theta(4^k) \). The lower bound on \( f(k) \) gives us a lower bound of \( gc(k) \in \Omega(2^k/k) \). Here, we have an exponential gap, too.

We summarize: We know the correct asymptotics of \( f(k) \), \( lc(k) \) and \( lc_1(k) \), the extremal parameters defined by restricting variable degrees, the size of the conflict neighborhoods and 1-conflict neighborhoods, respectively. In fact, we know the value of \( lc_1(k) \) exactly. In all three cases, the complete formula provides a not too bad upper bound. For \( f(k) \), it is only a factor \( k \) from the true value. For \( ic(k) \) and \( gc(k) \), the extremal parameters defined by restricting the number of conflicts generated by any individual variable and the total number conflicts, respectively, the complete formula gives a very poor upper bound, and we do not know the correct asymptotics. There is an exponential gap between the trivial lower and upper bounds we have given above. We will narrow, although not close, this gap for \( ic(k) \) in Section 7.3 and for \( gc(k) \) in Section 7.4.

**Complexity Jumps**

Several of the parameters defined above exhibit a complexity jump. For example, deciding whether a \( k \)-CNF formula \( F \) with \( \deg(F) \leq f(k) \) is satisfiable is trivial: It is satisfiable. However, satisfiability of \( k \)-CNF formulas with \( \deg(F) \leq f(k) + 1 \) is already NP-complete, as proved by Kratochvil et al. [KST93] (and by Tovey [Tov84] for \( k = 3 \)). What about other parameters? Gebauer, Moser, Welzl, and myself [GMSW09] showed that while satisfiability of \( k \)-CNF formulas with \( lc(F) \leq c \) is trivial for \( c \leq lc(k) \), it is NP-complete for \( c \geq lc(k) + 1 \). Interestingly, the parameter \( lc_1 \) does not exhibit a sudden jump:
Every $k$-CNF formula $F$ with $\text{lcl}_1(F) \leq k - 1$ is satisfiable, and deciding satisfiability for $k$-CNF formulas with $\text{lcl}_1(F) \leq k + 1$ is NP-complete. However, if we require $\text{lcl}_1(F) \leq k$, we obtain a problem that is non-trivial, but can be solved in polynomial time.

### Results

In Section 7.2, we investigate 1-conflicts of CNF formulas. We prove that $\text{lcl}_1(k) = k - 1$, i.e., every $k$-CNF formula $F$ with $\text{lcl}_1(F) \leq k - 1$ is satisfiable and there exists an unsatisfiable $k$-CNF formula $F$ with $\text{lcl}_1(F) = k - 1$. Furthermore, we show that satisfiability of $k$-CNF formulas with $\text{lcl}_1(F) \leq k + 1$ is NP-complete. Finally, when $\text{lcl}_1(F) \leq k$, we can efficiently decide satisfiability. The latter result rests on several interesting combinatorial properties of 1-conflicts.

In Section 7.3, we prove that $\text{ic}(k) \in O(3.01^k)$. That is, we construct an unsatisfiable $k$-CNF formula $F$ such that $\text{occ}_F(x) \cdot \text{occ}_F(\bar{x}) \in O(3.01^k)$ for every variables $x \in \text{vbl}(F)$. To achieve this goal, we develop a machinery for constructing very unbalanced unsatisfiable formulas. We employ both probabilistic and explicit constructions. Unfortunately, we did not manage to prove a lower bound of $\text{ic}(F) \geq a^k$ for any $a > 2$.

In Section 7.4, we use the machinery from Section 7.3 to prove that $\text{gc}(k) \in O(3.51^k)$. This means, we construct an unsatisfiable $k$-CNF formula with at most $O(3.51^k)$ conflicts. The second result of Section 7.4 is the main result of this chapter, and also the most technical. We show that $\text{gc}(k) \in \Omega(2.69^k)$, i.e., every unsatisfiable $k$-CNF formula has $\Omega(2.69^k)$ conflicts. We prove the lower bound by applying the Lovász Local Lemma with a very non-uniform distribution and subjecting the formula to a “truncation process” which makes it less satisfiable, but makes satisfiability easier to detect for the Local Lemma.

We think the results of Section 7.4 are interesting not because of the precise numbers, but since it shows that both the trivial upper bound and the trivial lower bound are far from being tight. We do not see any “natural” candidate $a^k$ for the true asymptotic behavior of $\text{gc}(k)$. The results of Section 7.3 and Section 7.4 are joint work with Philipp Zumstein [SZ08a, SZ08b].

Basically all proofs of lower and upper bounds on $\text{ic}$ and $\text{gc}$, i.e., the number of conflicts generated by an individual variable and the total number of conflicts, respectively, are probabilistic, whereas the proofs concerning the number of 1-conflicts use basic combinatorial reasoning. We summarize the currently best bounds of several extremal parameters the following table.
<table>
<thead>
<tr>
<th>parameter</th>
<th>notation</th>
<th>lower bound</th>
<th>upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable degree</td>
<td>( f(k) )</td>
<td>( \frac{2^{k+1}}{ek} (1 - o(1)) ) \cite{GST10}</td>
<td>( \frac{2^{k+1}}{ek} (1 + o(1)) ) \cite{GST10}</td>
</tr>
<tr>
<td>individual conflicts</td>
<td>( ic(k) )</td>
<td>( \frac{2^{k}}{ek} ) \cite{KST93}</td>
<td>( O(1) ) \cite{SZ08b}</td>
</tr>
<tr>
<td>global conflict number</td>
<td>( gc(k) )</td>
<td>( \Omega(2.69^k) ) \cite{SZ08b}</td>
<td>( O(3.51^k) ) \cite{SZ08b}</td>
</tr>
</tbody>
</table>

**Tools**

**The lopsided Lovász Local Lemma.** We will formally restate the Lopsided Lovász Local Lemma and then distill it into a convenient form we will use. For a graph \( G \) and a vertex \( u, \Gamma_G(u) \) to denote the set of vertices in \( G \) that are adjacent to \( u \), not including \( u \) itself.

**Definition 7.4 (Lopsided Dependency Graph).** Let \( A_1, \ldots, A_m \) be events in some probability space. A graph \( G(V, E) \) with \( V = \{ u_1, \ldots, u_m \} \) is called a lopsided dependency graph if for any \( A_i \) and any set \( U \subseteq V \setminus \Gamma_G(u_i) \) of vertices not adjacent to \( u_i \) with \( \Pr[\bigcap_{u_j \in U} \bar{A}_j] > 0 \), it holds that

\[
\Pr \left[ A_i \bigcap_{u_j \in U} \bar{A}_j \right] \leq \Pr [ A_i ] . \tag{7.2}
\]

**Lemma 7.5 (Lopsided Lovász Local Lemma \cite{ES91, AS00, LS07}).** Let \( A_1, \ldots, A_m \) be events in some probability space, and let \( G \) be a lopsided dependency graph on these events. If there are numbers \( 0 \leq \gamma_i < 1 \), \( 1 \leq i \leq m \), such that for any \( i \),

\[
\Pr [ A_i ] \leq \gamma_i \prod_{u_j \in \Gamma_G(u_i)} (1 - \gamma_j) , \tag{7.3}
\]

then

\[
\Pr [ \bar{A}_1 \cap \cdots \cap \bar{A}_m ] > 0 .
\]

For a CNF formula \( F = \{ C_1, \ldots, C_m \} \) we draw a random assignment \( \alpha \) by setting each variable \( x \) to true with a certain probability \( p(x) \), independent of all other variables. Let \( A_i \) be the event that \( \alpha \) does not satisfy clause \( C_i \). We claim that the conflict graph of \( F \) is a lopsided dependency graph for the events \( A_1, \ldots, A_m \). We have to show that (7.2) holds. In CNF terminology, for any satisfiable subformula \( F' \subseteq F \) consisting of clauses not conflicting with \( C \),

\[
\Pr [ \alpha \not\models C | \alpha \models F' ] \leq \Pr[\alpha \not\models C] .
\]
Showing this basically boils down to proving that two monotone boolean functions are positively correlated. This is a well-known fact, a special case of the so-called FKG inequality \([\text{FKG71}]\), a proof of which can also be found in \([\text{AS00}]\).

**Lemma 7.6** (SAT version of the Lopsided Lovász Local Lemma). Let \(F\) be a CNF formula not containing the empty clause. Sample a assignment \(\alpha\) by independently setting each variable \(x\) to true with some probability \(p(x) \in [0, 1]\). If for every clause \(C \in F\) it holds that

\[
\sum_{D \in \Gamma_F(C)]} \Pr[\alpha \not\models D] \leq \frac{1}{4},
\]

then \(F\) is satisfiable.

**Proof.** Let \(G := CG(F)\). We can assume that every clause \(C \in F\) conflicts with at least one other clause, i.e. \(G\) has no isolated vertices. Otherwise, \(F\) is satisfiable if and only if \(F \setminus \{C\}\) is satisfiable, and we prove the statement for \(F \setminus \{C\}\). Since \(C\) has at least one conflict, (7.4) implies \(\Pr[A_i] \leq \frac{1}{4}\), for all \(1 \leq i \leq m\). For \(\gamma_i := 2 \Pr[A_i]\), we have \(\gamma_i \leq \frac{1}{2}\). We show that (7.3) holds:

\[
\gamma_i \prod_{C_j \in \Gamma_G(C_i)} (1 - \gamma_j) \geq 2 \Pr[A_i] \left(1 - \sum_{C_j \in \Gamma_G(C_i)} 2 \Pr[A_j]\right)
\]

\[
\geq 2 \Pr[A_i] \left(1 - 2 \cdot \frac{1}{4}\right) = \Pr[A_i].
\]

In the first inequality, we used the definition of the \(\gamma_i\) and the fact that \(\prod (1 - \gamma_j) \geq 1 - \sum \gamma_i\), and in the second inequality we used the assumption (7.4). Lemma 7.5 now implies that \(\Pr[\bar{A}_1 \cap \cdots \cap \bar{A}_m] > 0\), i.e. with positive probability, no clause will be unsatisfied by \(\alpha\). In other words, there is a satisfying assignment.

Mostly we do not apply the Lovász Local Lemma directly to the formula for which we want prove satisfiability, but to a truncation of it:

**Definition 7.7** (Truncation). Let \(F = \{C_1, \ldots, C_m\}\) be a CNF formula. A formula \(G = \{D_1, \ldots, D_m\}\) is called a truncation of \(F\) if \(D_i \subseteq C_i\) for all \(1 \leq i \leq m\).

It is possible that \(D_i = D_j\) even if \(C_i \neq C_j\), so \(G\) may have fewer clauses than \(F\).

**Observation 7.8.** If \(F\) is a CNF formula over the variables \(V\) and \(G\) is a truncation of \(F\), then \(\text{sat}_V(G) \subseteq \text{sat}_V(F)\). In particular, if \(G\) is satisfiable, then \(F\) is satisfiable, too.
Hypergraph 2-Colorability. For proving upper bounds on $\text{ic}(k)$ and $\text{gc}(k)$, i.e., constructing unsatisfiable $k$-CNF formulas $F$ for which $\text{ic}(F)$ or $\text{gc}(F)$ are small, we use a second idea, coming from colorability of hypergraphs. A hypergraph $H = (V, E)$ is $k$-uniform if all hyperedges $e \in E$ have size $k$. It is 2-colorable if the vertices can be colored red and blue such that no edge becomes monochromatic. Notice the connection to satisfiability: Given a $k$-uniform hypergraph $H$, we construct a $k$-CNF formula $F$ by taking the vertices of $H$ as our variables, and introducing for every hyperedge $e = \{v_1, \ldots, v_k\}$ the two clauses $\{v_1, \ldots, v_k\}$ and $\{\bar{v}_1, \ldots, \bar{v}_k\}$. Observe that $F$ is satisfiable if and only if $H$ is 2-colorable. Conversely, if $F$ is a monotone $k$-CNF formula, i.e., every clause contains either only positive or only negative literals, then we can build a hypergraph $H = (\text{vbl}(F), E)$ by setting $E := \{\text{vbl}(C) \mid C \in F\}$. Note that if $H$ is 2-colorable, then $F$ is satisfiable, but the converse need not be true.

What is the minimum number of edges in a $k$-uniform hypergraph that is not 2-colorable? In 1963, Erdős [Erd63] raised that question and proved a lower bound of $2^{k-1}$: Every $k$-uniform hypergraph with fewer hyperedges is 2-colorable (simply choose a random 2-coloring). One year later, he [Erd64] gave a probabilistic construction of a non-2-colorable $k$-uniform hypergraph using $ck^2 2^k$ hyperedges for some constant $c$. What is the minimum number of clauses in an unsatisfiable monotone $k$-CNF formula? By the above considerations, the answer must be the same, up to a factor of 2, as for non-2-colorable $k$-uniform hypergraphs. For CNF formulas we can ask a more general question: We consider $(\ell, k)$-CNF formulas, which are CNF formulas containing positive clauses of size $\ell$ and negative clauses of size $k$. Thus, a $(k, k)$-CNF formula is simply a monotone $k$-CNF formula. What is the minimum number of clauses in an unsatisfiable $(\ell, k)$-CNF formula? Generalizing Erdős’ methods, we will prove rather tight bounds on this number in Section 7.3. The upper bounds, i.e., constructions of unsatisfiable $(\ell, k)$-CNF formulas, will help us prove upper bounds on $\text{ic}(k)$ and $\text{gc}(k)$.

The Combinatorial OR. We introduce a very convenient notation. Let $F$ and $G$ be a CNF formulas. Then $F \lor G$ is certainly a Boolean formula, although not in conjunctive normal form. However, by distributivity, we see that it is equivalent to the CNF formula

$$\{C \cup D \mid C \in F, D \in G\},$$

from which we remove trivial clauses, i.e., clauses containing both $x$ and $\bar{x}$. We denote the resulting formula by $F \triangledown G$. The notation extends to clauses via $F \triangledown C := F \triangledown \{C\}$ and to literals via $F \triangledown u := F \triangledown \{u\}$. In any case $F \triangledown G \equiv F \lor G$, that is, an assignment satisfies $F \lor G$ if and only if it satisfies
If \( F \) is a \( k_1 \)-CNF formula and \( G \) is a \( k_2 \)-CNF formula, then \( F \lor G \) is a \((\leq k_1 + k_2)\)-CNF formula. Furthermore, if \( \text{vbl}(F) \cap \text{vbl}(G) = \emptyset \), then \( F \lor G \) is a \((k_1 + k_2)\)-CNF formula. The main application of the combinatorial or is the simple observation that \( F \lor G \equiv G \) is \( F \) is unsatisfiable.

### 7.2 1-Conflicts

In this section we explore the combinatorial properties of 1-conflicts and study the extremal parameter \( l_{c_1} \). We prove that every \( k \)-CNF formula \( F \) with \( l_{c_1}(F) \leq k - 1 \) is satisfiable, deciding satisfiability of \( k \)-CNF formulas with \( l_{c_1}(F) = k + 1 \) is NP-complete, and, most interestingly, deciding satisfiability of \( k \)-CNF formulas with \( l_{c_1}(F) = k \) can be done in polynomial time.

#### 7.2.1 Combinatorial Properties

Let us start with a simple but surprising observation.

**Proposition 7.9.** Every CNF formula \( F \) with \( \Box \not\in F \) and \( l_{c_1}(F) = 0 \) is satisfiable.

Note that without such a property, the notion “1-conflict” would be misleading. After all, under any reasonable notion of conflict, a formula without conflicts should be satisfiable (extreme cases like \( \Box \in F \) excluded).

**Proof.** By completeness of resolution, a CNF formula \( F \) is unsatisfiable if and only if there is a resolution derivation of the empty clause. Since \( F \) has no 1-conflicts, we cannot build any new resolvents. Since \( \Box \not\in F \), the formula is satisfiable. \( \square \)

Somewhat amusingly, almost every “reasonable” algorithm for \( k \)-SAT will find a satisfying assignment of \( F \). For example, consider the following algorithm:

Pick some arbitrary variable \( x \in \text{vbl}(F) \) and choose \( b \in \{0, 1\} \) as follows: If \( \{x\} \in F \), set \( b = 1 \). If \( \{\bar{x}\} \in F \), set \( b = 0 \). If it contains neither, choose \( b \in \{0, 1\} \) arbitrarily. Set \( F' := F[x \mapsto b] \) and continue recursively with \( F' \).

The algorithm is well-defined, since by assumption \( F \) cannot contain both \( \{x\} \) and \( \{\bar{x}\} \). The algorithm will eventually arrive at the empty formula, having constructed a satisfying assignment: The formula \( F' \) fulfills the assumptions of Proposition 7.9 (\( \Box \not\in F' \) and \( l_{c_1}(F') = 0 \)), and thus the algorithm finds a satisfying assignment by induction. As a second example, consider the following variant of Schöning’s random walk algorithm [Sch99]:

99
Start with some assignment \( \alpha \). As long as \( \alpha \) does not satisfy \( F \), pick an arbitrary unsatisfied clause \( C \), pick a literal \( u \in C \) arbitrarily and flip the value of the underlying variable, obtaining an assignment \( \alpha' \). Formally, \( \alpha' := \alpha[u \mapsto 1] \). Repeat.

To see that this algorithm is correct and terminates, consider \( \alpha, C, \) and \( \alpha' \) as above. Obviously \( \alpha' \) satisfies \( C \). Not so obviously, it also satisfies every clause \( D \) that was satisfied by \( \alpha \). This holds due to the local flipping argument we will use several times in this section.

**The local flipping argument.** To see that \( \alpha' \) satisfies \( D \), observe what happens when we flip \( x \), the underlying variable of \( u \). If \( x \notin \text{vbl}(D) \), the clause \( D \) will not care, and \( \alpha' \) satisfies \( D \) if and only if \( \alpha \) does. If \( u \in D \), then \( D \) will only benefit from setting \( u \) to 1. If \( \bar{u} \in D \), then \( C \) and \( D \) have a conflict. Since they do not have a 1-conflict, there is some literal \( v \neq u \) such that \( v \in C \) and \( \bar{v} \in D \). Now \( \alpha(v) = 0 \), since \( \alpha \) does not satisfy \( C \). Since \( \alpha \) and \( \alpha' \) agree on \( v \), \( \alpha' \) satisfies \( D \). This constitutes another proof of Proposition 7.9.

**Definition 7.10 (Free Literals and Clauses).** Let \( F \) be a CNF formula, \( C \in F \) a clause, and \( u \in C \) some literal. We say \( u \) is free in \( C \) with respect to \( F \) if \( \Gamma^1_F(C, u) = \emptyset \), i.e., if there is no clause \( D \in F \) such that \( C \cap \bar{D} = \{u\} \). Furthermore, we say that \( C \) is free with respect to \( F \) if it contains some literal \( u \) that is free in \( C \) with respect to \( F \).

If the ambient formula is understood, we simply say that \( u \) is free in \( C \) and \( C \) is free.

**Proposition 7.11.** Let \( F \) be a CNF formula and \( C \in F \) some clause. If \( C \) is free, then \( F \) is satisfiable if and only if \( F \setminus \{C\} \) is.

**Proof.** If \( F \) is satisfiable, then \( F \setminus \{C\} \) is satisfiable, too. For the other direction, assume that \( F \setminus \{C\} \) is satisfiable and let \( \alpha \) be an assignment that is defined on every variable of \( \text{vbl}(F) \) and satisfies \( F \setminus \{C\} \). Let \( u \in C \) be a literal that is free in \( C \). If \( \alpha \) satisfies \( C \), we are done. Otherwise, \( \alpha(u) = 0 \) and we define \( \alpha' := \alpha[u \mapsto 1] \). We apply the local flipping argument from above to show that \( \alpha' \) satisfies \( F \): Clearly, \( \alpha' \) satisfies \( C \), and also every clause \( D \in F \) with \( \bar{u} \in D \). This holds because if \( \bar{u} \in D \), then \( |C \cap \bar{D}| \geq 2 \), and \( \alpha \) satisfies at least two literals in \( D \). Therefore, \( \alpha' \) satisfies at least one literal in \( D \). Thus, \( \alpha' \) satisfies \( F \). \( \square \)

We can use Proposition 7.11 to repeatedly remove free clauses in a formula, finally arriving at a formula without free clauses. The algorithm removeFree formalizes this procedure.
Algorithm 11 \texttt{removeFree}(CNF formula $F$)

1: \textbf{while} there is a clause $C \in F$ that is free with respect to $F$ \textbf{do}
2: \hspace{1em} $F := F \setminus \{C\}$
3: \textbf{end while}
4: \textbf{return} $F$

**Proposition 7.12.** Let $F$ be a CNF formula and let $F' := \texttt{removeFree}(F)$. Then $F \equiv_{SAT} F'$.

**Proof.** This follows from Proposition 7.11 and induction on the number of clauses.

Proposition 7.12 yields another proof of Proposition 7.9 on page 99. If $\text{lc}_1(F) = 0$, then every non-empty clause contains a free literal, i.e., every clause $C$ is free in $F$. The algorithm \texttt{removeFree} will remove one by one, finally arriving at the empty formula, which is satisfiable. Here we were using an innocent but crucial fact: If a clause $C$ is free with respect to $F$, then it is also free with respect to every subformula $F' \subseteq F$ for which $C \in F'$.

**Lemma 7.13.** A CNF formula is satisfiable if and only if every connected component of its 1-conflict graph is satisfiable.

Again, this is something we expect from a reasonable notion of conflict. Since each clause in $F$ is itself satisfiable, unless $\Box \in F$, the lemma implies that a formula $F$ with an empty 1-conflict graph and without the empty clause is satisfiable, providing yet another proof of Proposition 7.9.

**Proof.** The one direction is trivial: If $F$ is satisfiable, then all connected components are satisfiable. For the other direction, we assume that every connected component is satisfiable and use induction on the number of connected components. If this number is 1, i.e., if the 1-conflict graph of $F$ is connected, the lemma holds trivially true. Otherwise, $F$ can be written as $F = F_1 \uplus F_2$ such that there is no 1-conflict between $F_1$ and $F_2$, i.e., for all $C \in F_1$ and $D \in F_2$, it holds that $|C \cap \bar{D}| \neq 1$. Furthermore, both $F_1$ and $F_2$ have fewer connected components than $F$ and therefore are satisfiable by induction.

The rest of the proof is a variation of what we have seen above. Since $F_1$ and $F_2$ are satisfiable, we can choose a pair $\alpha_1, \alpha_2$ of assignments to $\text{vbl}(F)$ such that $\alpha_1$ satisfies $F_1$, $\alpha_2$ satisfies $F_2$, and the Hamming distance $d_H(\alpha_1, \alpha_2)$ is minimized. We claim that $\alpha_1$ satisfies $F_2$ as well, and therefore $F$. Suppose for the sake of contradiction that this is not the case. There is a clause $D \in F_2$ such that $\alpha_1$ does not satisfy $D$. Since $\alpha_2$ satisfies $D$, there is a literal $u \in$
such that $\alpha_1(u) = 0$ and $\alpha_2(u) = 1$. Define $\alpha'_1 := \alpha[u \mapsto 1]$. Clearly $d_H(\alpha'_1, \alpha_2) = d_H(\alpha_1, \alpha_2) - 1$. If we can prove that $\alpha'_1$ still satisfies $F_1$, we have arrived at a contradiction to $d_H$ being minimal, and are done. Showing that $\alpha'_1$ satisfies $F_1$ is another application of the local flipping argument: Consider any $C \in F_1$. By the assumptions of the lemma, there is no 1-conflict between $C$ and $D$. Hence either $C \cap \overline{D} = \emptyset$ or $|C \cap \overline{D}| \geq 2$. In the first case, $\alpha_1(C) = \alpha'_1(C) = 1$. In the second case, $\alpha_1$ satisfies at least two literals in $C$, and therefore, $\alpha'_1$ satisfies at least one literal in $C$. This shows that $\alpha'_1$ indeed satisfies $F_1$, contradicting minimality of $d_H(\alpha_1, \alpha_2)$.

### 7.2.2 The Main Result: Trivial – Tractable – Hard

**Theorem 7.14.** The following three statements hold for all $k \geq 0$:

(i) Every $k$-CNF formula $F$ with $\ell c_1(F) \leq k - 1$ is satisfiable.

(ii) There exists an unsatisfiable $k$-CNF formula $F$ with $\ell c_1(F) = k$.

(iii) Satisfiability of $k$-CNF formulas $F$ with $\ell c_1(F) \leq k$ can be decided in polynomial time.

(iv) Deciding satisfiability of $k$-CNF formulas $F$ with $\ell c_1(F) \leq k + 1$ is NP-complete, if $k \geq 3$.

**Proof.** If $F$ is a $k$-CNF formula and $\ell c_1(F) \leq k - 1$, then every clause $C \in F$ is free. Therefore $\text{removeFree}(F) = \{\}$ and by Proposition 7.12, $F$ is satisfiable. This shows point [i]. To prove [ii] consider the complete formula $\text{CF}(x_1, \ldots, x_k)$. It is unsatisfiable, and every clause has exactly $k$ many 1-conflicts.

Let us show point (iii). This is the most interesting point and also the main result of this section. Suppose $F$ is a $k$-CNF formula and $\ell c_1(F) \leq k$. We consider the algorithm $\text{frugalSAT}$. To see correctness of this algorithm, consider lines 5 and 6. The algorithm recurses on $F_1$ and $F_2$ and returns $\text{true}$ if both calls return $\text{true}$. By Lemma 7.13, $F$ is satisfiable if and only if $F_1$ and $F_2$ are both satisfiable individually. The challenging part is to argue that if $\ell c_1(F) \leq k$, then its running time is polynomial in $|F|$. Actually for this we need $F$ to satisfy a stronger property.

**Definition 7.15 (Frugal Literals, Clauses, and Formulas).** Let $F$ be a CNF formula, $C \in F$ a clause $u \in C$ a literal. We say $u$ is frugal in $C$ with respect to $F$ if $|\Gamma^+_F(C, u)| = 1$, i.e., there is exactly one clause $D \in F$ with $C \cap \overline{D} = \{u\}$. If every $u \in C$ is free or frugal with respect to $F$, we say $C$ is frugal with respect to $F$. If every clause $C \in F$ is frugal with respect to $F$, we call $F$ itself frugal.
Algorithm 12 \texttt{frugalSAT}(CNF formula \( F \))

1: \textbf{if} \( \square \in F \) \textbf{then}
2: \hspace{1em} \textbf{return} \ false
3: \textbf{else if} \( F = \{\} \) \textbf{then}
4: \hspace{1em} \textbf{return} \ true
5: \textbf{else if} \( F = F_1 \uplus F_2 \) for some \( F_1, F_2 \neq \{\} \) and \( |C \cap \bar{D}| \neq 1 \) for all \( C \in F_1, D \in F_2 \) \textbf{then}
6: \hspace{1em} \textbf{return} \ \texttt{frugalSAT}(F_1) \land \texttt{frugalSAT}(F_2)
7: \textbf{else}
8: \hspace{1em} \( x \rightarrow \text{vbl}(F) \)
9: \hspace{2em} \( G_1 := \text{removeFree}(F[x \rightarrow 1]) \)
10: \hspace{2em} \( G_0 := \text{removeFree}(F[x \rightarrow 0]) \)
11: \hspace{1em} \textbf{return} \ \texttt{frugalSAT}(G_1) \lor \texttt{frugalSAT}(G_0)
12: \textbf{end if}

Proposition 7.16. If \( F \) is frugal, then so is every \( F' \subseteq F \). If \( u \) is frugal in \( C \) with respect to \( F \), then for all \( F' \subseteq F \) with \( C \in F' \) it holds that \( u \) is frugal in \( C \) with respect to \( F' \).

We claim that if \( F \) satisfies \( \text{l}_{c_1}(F) \leq k \), then every clause in \( F \) is frugal or free: Let \( C \in F \). If \( |\Gamma^1_F(C)| \leq k - 1 \), then \( C \) is free. If \( |\Gamma^1_F(C)| = k \), then there are two cases: Either, it could be that the \( k \) sets \( \Gamma^1_F(C, u) \) for \( u \in C \) all have size 1. In this case, all \( u \in C \) are frugal, and \( C \) is frugal. Or, there is some \( u \in C \) such that \( \Gamma^1_F(C, u) = \emptyset \), i.e., \( u \) is free in \( C \). In this case, \( C \) is free. This shows that every clause in \( F \) is frugal or free. Define \( F' := \text{removeFree}(F) \). By the property of the algorithm \texttt{removeFree}, \( F' \) contains no free clauses. Therefore, every clause in \( F' \) is frugal, thus \( F' \) is frugal. Furthermore, \( F' \) is satisfiable if and only if \( F \) is. Hence from now on we can assume that the input formula is frugal.

The Algorithm \texttt{frugalSAT} on Frugal CNF Formulas

Claim 1. If \( F \) is a frugal CNF formula and has no free clauses, then \( \texttt{frugalSAT}(F) \) runs in polynomial time.

In this claim we do not assume that \( F \) is a \( k \)-CNF formula. Indeed, it is easy to see that \( \texttt{frugalSAT} \) will in general recurse on formulas containing clauses of size smaller than \( k \). The reason why the claim holds is that the calls to \texttt{removeFree} in lines 9 and 10 of \( \texttt{frugalSAT} \) delete many clauses. In fact, we will see that any clause of \( F \) is deleted in at least one of these two calls. For
this, consider the 1-conflict graph of \( F \). In this graph, every clause has a certain degree \( \deg(C) \). Since \( F \) is frugal, \( \deg(C) \leq |C| \) for every \( C \in F \). Furthermore, \( C \) is free in \( F \) if and only if \( \deg(C) < |C| \). The procedure \( \text{removeFree}(F) \) can now be seen as repeatedly deleting all clauses \( C \) for which \( \deg(C) < |C| \). This implies the following fact:

**Proposition 7.17.** Let \( F \) be a frugal CNF formula. Then \( \text{removeFree}(F) \) consists of exactly those clauses of \( F \) that lie in a connected component of the 1-conflict graph of \( F \) that does not contain any clause that is free with respect to \( F \).

Consider what happens in lines [9] and [10] of \( \text{frugalSAT} \): At this point, \( F \) is frugal and its 1-conflict graph is connected. Furthermore, it has no free clauses.

**Proposition 7.18.** Let \( F \) be a frugal CNF formula with a connected 1-conflict graph and without free clauses. Let \( x \in \text{vbl}(F) \). Define \( G_0 := \text{removeFree}(F[x \leftarrow 0]) \) and \( G_1 := \text{removeFree}(F[x \leftarrow 1]) \). Then \(|G_0| + |G_1| \leq |F| \), and \(|G_0|, |G_1| \leq |F| - 1\).

**Proof.** Since \( x \in \text{vbl}(F) \), some clause contains \( x \), and since this clause is not free, some other clause contains \( \bar{x} \). Thus, fixing \( x \) to some truth value deletes at least one clause, and therefore \(|G_0|, |G_1| \leq |F| - 1\). To see that \(|G_0| + |G_1| \leq |F|\), define \( F_x := \{C \in F | x \in C\} \) and \( F_{\bar{x}} := \{C \in F | \bar{x} \in C\} \). See Figure 7.3 for an illustration. Consider some clause \( C \in F \) that contains neither \( x \) nor \( \bar{x} \). Since the 1-conflict graph of \( F \) is connected, there is some path from \( C \) to \( F_x \cup F_{\bar{x}} \). Take a shortest such path: \( C = C_0, C_1, \ldots, C_t \), where \( C_0, \ldots, C_{t-1} \) contain neither \( x \) nor \( \bar{x} \). Without loss of generality, \( C_t \in F_x \). In \( F[x \leftarrow 1] \), clause \( C_t \) disappears, and the degree of \( C_{t-1} \) shrinks by at least 1. The clauses \( C_0, \ldots, C_{t-1} \) do not change their size. Therefore, \( C_{t-1} \) is free in \( F[x \leftarrow 1] \). \( C \) and \( C_{t-1} \) lie in the same connected component of the 1-conflict graph of \( F[x \leftarrow 1] \), and thus by Proposition 7.17, \( C \notin G_1 \). If \( C_t \), the endpoint of the shortest path from \( C \) to \( F_x \cup F_{\bar{x}} \), lies in \( F_{\bar{x}} \), things are similar: In that case, \( C \notin G_0 \). We conclude that some \( C \in F \setminus (F_x \cup F_{\bar{x}}) \) can show up in at most one of \( G_0 \) and \( G_1 \). The same of course holds for \( C \in F_x \cup F_{\bar{x}} \), since either setting \( x \leftarrow 1 \) or \( x \leftarrow 0 \) makes \( C \) disappear. This completes the proof. \( \square \)

Let us go back to the big picture of algorithm \( \text{frugalSAT} \). It recurses either on \( F_1, F_2 \) (line [6]) or on \( G_1, G_0 \) (line [11]). In any case, denote the two formulas on which the algorithm recurses by \( F_1 \) and \( F_2 \). If \( F \) is frugal, then \( F_1 \) and \( F_2 \) are both frugal, and \(|F_1| + |F_2| \leq |F|\). Note that if the algorithm recurses in line [6] this holds with equality, whereas in line [11] the inequality can be strict.
Consider the recursion tree built by the \textit{frugalSAT}. If $F$ is empty, the tree consists of a single node. Otherwise, we claim that it has at most $4m - 1$ nodes, where $m = |F|$ is the number of clauses. We prove this claim by induction on $m$. If $m = 1$, then $F = \{C\}$ for some clause $C$. If $C = \Box$, the algorithm immediately returns false, hence the tree has one node. Otherwise, it branches at some variable. In this case, one checks that $G_0$ is either $\{\Box\}$ or $\{\}$, likewise $G_1$. This holds because a clause $C$ in a formula $\{C\}$ is either the empty clause $\Box$, or it is free, thus removed by the call to \textit{removeFree}. Thus if $|F| = 1$, the recursion tree has at most 3 nodes. If $|F| \geq 2$, then the algorithm recurses on two formulas with $a$ and $b$ clauses, respectively, with $a, b \leq m - 1$ and $a + b \leq m$. By induction the number of nodes in the tree is at most

$$(4a - 1) + (4b - 1) \leq (4a - 1) + (4(m - a) - 1) + 1 = 4m - 1,$$

as claimed. We see that the number of nodes in the recursion tree is linear in $|F|$. Clearly the work required in each call is polynomial in $|F|$: Computing the connected components of a graph and, if the graph is connected, computing $\text{removeFree}(F[x \rightarrow 1])$ and $\text{removeFree}(F[x \rightarrow 0])$. This can be done in linear time if $F$ is given in the right format. This shows how to efficiently decide satisfiability for $k$-CNF formulas $F$ with $\text{lc}_1(F) \leq k$ and finishes the proof of point (iii).

### 7.2.3 Hardness for $\text{lc}_1 \geq k + 1$

Finally, we show point (iv) of Theorem 7.14 on page 102. The proof is similar to the proof by Gebauer et al. [GMSW09] that the number of local conflicts
exhibits a complexity jump. We will give a reduction that takes a $k$-CNF formula $F'$ as input and outputs a $k$-CNF formula $F''$, such that $F' \equiv_{\text{SAT}} F''$, and $\text{lcc}(F') \leq k + 1$, i.e., every clause in $F'$ has a 1-conflict with at most $k + 1$ other clauses. In fact, $F'$ will have a stronger property: Of the $k$ literals in a clause $C \in F'$, $k - 1$ literals generate at most one 1-conflict, i.e., are frugal or free in $C$, and one literal is allowed to have up to two 1-conflicts. If $k \geq 3$, this shows that deciding satisfiability of $k$-CNF formulas with $\text{lcc}(F) \leq k + 1$ is NP-complete.

In a first step, for each variable $x \in \text{vbl}(F)$ we set $d := \text{deg}_F(x)$ and introduce $2d$ new variables $x_1, x_2, \ldots, x_{2d}$. We replace the $d$ occurrences of $x$ by the variables $x_2, x_4, \ldots, x_{2d}$. Skipping the odd indices will prove useful soon. We call the new formula $F_2$. For example, if $x$ appears in three clauses, say

$$F = \{\{x, \bar{y}, \bar{z}\}, \{x, u, v\}, \{\bar{x}, y, \bar{u}\}, \ldots\},$$

then we replace those three occurrences by $x_2$, $x_4$, and $x_6$ and obtain

$$\{\{x_2, \bar{y}, \bar{z}\}, \{x_4, u, v\}, \{\bar{x_6}, y, \bar{u}\}, \ldots\}.$$  

We apply the same procedure to $y, z$, and all other variables. $F_2$ has no conflicts, since each variable appears in only one clause. It is satisfiable, which is not good, because we want a formula $F''$ such that $F' \equiv_{\text{SAT}} F''$. We introduce an equalizer formula for the variables $x_1, x_2, \ldots, x_{2d}$, that is a formula which is satisfied only if one assigns the same value to $x_1, x_2, \ldots, x_{2d}$. This is achieved by the following formula

$$\text{Eq}(x_1, \ldots, x_{2d}) = \{\{\bar{x}_1, x_2\}, \{\bar{x}_2, x_3\}, \ldots, \{\bar{x}_{d-1}, x_d\}, \{\bar{x}_d, x_1\}\}.$$  

$\text{Eq}(x_1, \ldots, x_{2d})$ has exactly two satisfying assignments: One that sets every variable to 1, and one that sets every variable to 0. We obtain $F_3$ by adding an equalizer for every variable $x \in \text{vbl}(F)$:

$$F_3 := F_2 \cup \bigcup_{x \in \text{vbl}(F)} \text{Eq}(x_1, x_2, \ldots, x_{2\text{deg}_F(x)}).$$

The property of the equalizers implies $F \equiv_{\text{SAT}} F_3$. Furthermore, $|\Gamma_{F_3}(C)| \leq |C| + 1$ for every $C \in F_3$: Since each occurrence of a variable in $F$ gets replaced by a fresh copy of this variable, there are no conflicts within $F_2$. Every clause $C \in F_2$ has a 1-conflict with exactly $|C|$ clauses in the equalizer formulas: If $x_i \in C$, then $C$ has a 1-conflict with the clause $\{\bar{x}_i, x_{i+1}\} \in \text{Eq}(x_1, \ldots, x_{2\text{deg}_F(x)})$. Similarly, if $\bar{x}_i \in C$, then $C$ has a 1-conflict with $\{x_{i-1}, x_i\}$. Therefore, $C$ is frugal. Consider a clause $\{\bar{x}_i, x_{i+1}\}$ in an equalizer. Each of the
two literals generates one 1-conflict with another equalizer-clause. Additionally, $x_i$ (if $i$ is even) or $x_{i+1}$ (if $i$ is odd) might generate a 1-conflict with a clause in $F_2$. Thus, $|\Gamma_{F_3}(C)| \leq |C|$ if $C \in F_2$, and $|\Gamma_{F_3}^{-1}(C)| \leq |C| + 1$ if $C$ is an equalizer-clause.

The formula $F_3$ fulfills almost all our needs, except that its clauses are too short: We want to output a $k$-CNF formula. For this reason, we add $k - 2$ new variables to each equalizer clause: We replace $\{\bar{x}_i, x_{i+1}\}$ by

$$\{\bar{x}_i, x_{i+1}, u_3, \ldots, u_k\}.$$ 

We add clauses that force the variables $u_3, \ldots, u_k$ to 0: For each $u_j$, we take a complete formula $C_F(v_1, \ldots, v_k)$ over $k$ new variables $v_1, \ldots, v_k$. In this formula, every clause is frugal. We pick one clause from it, say $\{v_1, \ldots, v_k\}$, and replace it by $\{v_1, \ldots, v_{k-1}, \bar{u}_j\}$. We call this $k$-CNF formula $G(u_j)$. It is satisfiable, but every satisfying assignment sets $u_j$ to 0. Furthermore, every clause in $G(u_j)$ is frugal, and $\bar{u}_j$ is free in $\{v_1, \ldots, v_{k-1}, \bar{u}_j\}$ with respect to $G(u_j)$. We denote by $F'$ the $k$-CNF formula we obtain from $F_3$ by filling up the equalizer-clauses and adding the formulas $G(u_j)$ to it.

For every $x \in \text{vbl}(F)$, we add $2 \deg_{F}(x)$ equalizer clauses, each of which we fill up to a $k$-clause, introducing a total of $(k - 2)2\deg_{F}(x)$ new variables. Finally, we add the formulas $G(u_j)$, consisting of $2^k$ clauses. That is, for each $x \in \text{vbl}(F)$, we add $2 \deg_{F}(x) + 2(k - 2)\deg_{F}(x)2^k$ clauses. A lot, but only a blow-up by a constant factor.

There are three types of clauses in $F'$: First, there are the “original” clauses, those of $F_2$. These clauses are frugal in $F_3$, and they are also frugal in $F'$. Second, there are equalizer clauses $\{\bar{x}_i, x_{i+1}, u_3, \ldots, u_k\}$. Here, every literal is frugal, except possibly $x_i$ (if $i$ is even) or $x_{i+1}$ (if $i$ is odd). Thus this clause has at most $k + 1$ many 1-conflicts. Third, there are clauses in $G(u_j)$. Every clause $C \in G(u_j)$ is frugal in $G(u_j)$, and the literal $\bar{u}_j$ is free in $\{v_1, \ldots, v_{k-1}, \bar{u}_j\}$ with respect to $G(u_j)$. Since $u_j$ occurs in exactly one equalizer clause, this adds exactly one 1-conflict to $\{v_1, \ldots, v_{k-1}, \bar{u}_j\}$. Therefore every $C \in G(u_j)$ is frugal in $F'$. This concludes the proof.

## 7.3 Conflicts Generated by an Individual Variable

In this section we investigate what happens when we restrict the conflict structure of each individual variable. Recall the result from Kratochvíl, Savický, and Tuza [KST93] that every $k$-CNF formula $F$ with $\deg(F) \leq 2^k/(ek)$ is satisfiable. In words, imposing sufficiently strong restrictions on the frequency of each variable in a $k$-CNF formula guarantees that it is satisfiable. In this

107
section, we investigate what happens if both polarities of a variable (positive and negative) are restricted separately. This leads us to several questions:

Is there a real number $a > 1$ such that for every unsatisfiable $k$-CNF formula $F$ there is a variable with $oc_{F}(x) \geq a^k$ and $oc_{F}(\bar{x}) \geq a^k$?

The answer to this question is a very strong “no”:

**Theorem 7.19.** For every $k \in \mathbb{N}_0$, there is an unsatisfiable $k$-CNF formula $F$ with $oc_{F}(\bar{x}) \leq 1$ for all $x \in vbl(F)$.

**Proof.** The proof is by induction. For $k = 0$, take $F_0 = \{\square\}$. For the induction step, we suppose the theorem holds for $k$, and let $F_k$ be an unsatisfiable $k$-CNF formula such that $oc_{F_k}(\bar{x}) \leq 1$ for all $x \in vbl(F_k)$. We construct an unsatisfiable $(k+1)$-CNF formula $F_{k+1}$ as follows: We take $k+1$ disjoint copies $F_k^{(1)}, \ldots, F_k^{(k+1)}$ of $F_k$, i.e., rename the variables in every copy. We introduce $k+1$ new variables $x_1, \ldots, x_{k+1}$. For $1 \leq i \leq k+1$, we obtain $G_i$ by adding $x_i$ to every clause in $F_k^i$, i.e., $G_i := F_k^i \lor x_i$ (recall the definition of the combinatorial OR $\lor$ on page 98). Since $F_k$ is unsatisfiable, every satisfying assignment of $G_i$ sets $x_i$ to 1. Finally, we let

$$F_{k+1} := G_1 \cup \cdots \cup G_{k+1} \cup \{\bar{x}_1, \ldots, \bar{x}_{k+1}\}.$$

This is an unsatisfiable $(k + 1)$-CNF formula, and $deg_{F_{k+1}}(\bar{x}) \leq 1$ for every $x \in vbl(F_{k+1})$. 

Note that achieving $oc_{F}(\bar{x}) \leq 1$ comes at a cost: The formulas in the proof contain variables $x$ with $oc_{F}(x) \geq (k-1)!$. Intuitively, if we want to construct an unsatisfiable $k$-CNF formula $F$ for which $oc_{F}(\bar{x})$ is “very small” for every variable $x$, then we must allow $oc_{F}(y)$ to be “very large” for some variable $y$. The next theorem justifies this intuition.

**Theorem 7.20 (Exponential Tradeoff Between Positive and Negative Degree).** The following two statements hold:

(i) (Upper Bound) For every $1 < a \leq 2$ and $b = \frac{a}{a-1}$ there is a constant $c$ such that for all sufficiently large $k$, there is an unsatisfiable $k$-CNF formula $F$ with $oc_{F}(\bar{x}) \leq ck^2a^k$ and $oc_{F}(x) \leq ck^2b^k$, for all $x$.

(ii) (Lower Bound) Let $1 < a < \sqrt{2}$ and $b = \sqrt{\frac{a^4}{a^2-1}}$. Then every $k$-CNF formula $F$ with $oc_{F}(x) \leq \frac{b^k}{8k}$ and $oc_{F}(\bar{x}) \leq \frac{a^k}{8k}$ is satisfiable.
The point of Theorem 7.20 is to show that some tradeoff exists, even if we cannot quantify it exactly. For example, point (ii) implies that every \( k \)-CNF formula in which \( \text{occ}_F(\overline{x}) \leq 1.25^k/8k \) and \( \text{occ}_F(x) \leq 2.08333^k/8k \) for every variable \( x \) is satisfiable. On the other hand point (i) shows that there are unsatisfiable \( k \)-CNF formulas with \( \text{occ}_F(\overline{x}) \leq c1.25^k k^2 \) and \( \text{occ}_F(x) \leq c5^k k^2 \). It leaves open whether there is an unsatisfiable \( k \)-CNF formula \( F \) with \( \text{occ}_F(\overline{x}) \leq 1.25^k/(8k) \) and \( \text{occ}_F(x) \leq 4^k \) for all \( x \in \text{vbl}(F) \). We suspect that if \( F \) is unsatisfiable, then for some variable \( x \), the product \( \text{occ}_F(x) \cdot \text{occ}_F(\overline{x}) \) is large. For this we have defined

\[
ic(F) := \max\{\text{occ}_F(x) \cdot \text{occ}_F(\overline{x}) \mid x \in \text{vbl}(F)\}
\]

and \( \nic(k) \) to be the greatest integer \( c \) such that every \( k \)-CNF formula \( F \) with \( \nic(F) \leq c \) is satisfiable. We will prove the following theorem:

**Theorem 7.21.** \( \nic(k) \in O(3.01^k) \).

The best lower bound we have is \( \nic(F) \geq f(k) - 1 \): Take a \( k \)-CNF formula \( F \). We can assume that \( \text{occ}_F(x) \geq 1 \) and \( \text{occ}_F(\overline{x}) \geq 1 \) for all \( x \in \text{vbl}(F) \): If for example \( \text{occ}_F(\overline{x}) = 0 \), then \( F' := F[\overline{x} \mapsto 1] \) is a \( k \)-CNF formula as well, and \( \nic(F') \leq \nic(F) \). If \( \text{occ}_F(x) \cdot \text{occ}_F(\overline{x}) \leq f(k) - 1 \), then \( \deg_F(x) = \text{occ}_F(x) + \text{occ}_F(\overline{x}) \leq (f(k) - 1) + 1 \), and \( F \) is satisfiable by definition of \( f(k) \).

We will first prove upper bounds, that is, construct unsatisfiable \( k \)-CNF formulas with certain properties. The goal is to prove Theorem 7.21 and point (i) of Theorem 7.20. Afterwards, we prove the lower bound, i.e., point (ii) of Theorem 7.20.

**Proving the Upper Bounds**

We want to construct very unbalanced unsatisfiable \( k \)-CNF formulas, i.e., formulas in which positive literals are much more frequent than negative literals. For this, we first employ a probabilistic construction of unsatisfiable CNF formulas with clauses of size \( k \) and \( \ell \), for some \( \ell \leq k \). In a second step, we expand all smaller clauses to size \( k \).

**Definition 7.22.** Let \( F \) be a \((\leq k)\)-CNF formula. For each \( C \in F \) we introduce \( k - |C| \) new variables \( y_1^C, \ldots, y_{k-|C|}^C \) and construct the complete \((k - |C|)\)-CNF formula \( G_C := CF(y_1^C, \ldots, y_{k-|C|}^C) \). Note that \( C \cup G_C \) is a \( k \)-CNF formula and, since \( G_C \) is unsatisfiable, \( C \cup G_C \equiv C \). Finally we define

\[
G := \bigcup_{C \in F} (C \cup G_C)
\]

which we call the \( k \)-CNFification of \( F \).
For example, a 3-CNFfication of \{\{x, y\}, \{\bar{x}, y, z\}\} is the 3-CNF formula \\
\\{\{x, y, y_1\}, \{x, y, y_1\}, \{\bar{x}, y, z\}\}. Note that the \(k\)-CNFfication is unique up to \\
the names of the newly introduced variables. By the property of \(\forall\) we see that \\
a CNF formula is equivalent to its \(k\)-CNFfication.

**Definition 7.23.** Let \(\ell, k \in \mathbb{N}_0\). An \((\ell, k)\)-CNF formula is a formula consisting \\
of \(\ell\)-clauses containing only positive literals, and \(k\)-clauses containing only negative literals.

If \(F\) is an \((\ell, k)\)-CNF formula, we write \(F = F^+ \cup F^-\), where \(F^+\) consists of \\
positive \(\ell\)-clauses and \(F^-\) of negative \(k\)-clauses.

**Lemma 7.24.** Let \(\ell \leq k\) and let \(F = F^+ \cup F^-\) be an \((\ell, k)\)-CNF formula. Let \(G\) be \\
the \(k\)-CNFfication of \(F\). Then \(ic(F) \leq \max\{4^{k-\ell}, 2^{k-\ell}|F^+| \cdot |F^-|\}\).

**Proof.** Let \(x\) be any variable in \(vbl(G)\). We want to show that \(occ_G(x) \cdot occ_G(\bar{x})\) is as stated in the proposition. We distinguish two cases: A variable \(x \in vbl(G)\) either appears in \(F\), or it has been introduced in the \(k\)-CNFfication. First, if \\
\(x\) appears in \(F\), then \(occ_G(\bar{x}) = occ_F(\bar{x})\) and \(occ_G(x) = occ_F(x)2^{k-\ell}\), thus \\
\(occ_G(x)occ_G(\bar{x}) \leq 2^{k-\ell}|F^+| \cdot |F^-|\). Second, if \(x\) has been introduced in the \\
\(k\)-CNFfication, then \(occ_G(x) = occ_G(\bar{x}) = 2^{k-\ell-1}\), and \(occ_G(x) \cdot occ_G(\bar{x}) \leq 4^{k-\ell}\).

Our proof strategy now is as follows: We use a probabilistic construction to obtain unsatisfiable \((\ell, k)\)-CNF formulas for some \(\ell \leq k\) that suits our needs. Should this \(\ell\) be smaller than \(k\), we build the \(k\)-CNFfication of it and use \\
Lemma 7.24 to estimate the maximum number of individual conflicts in it.

**Lemma 7.25.** For any \(\rho \in (0, 1)\), there is a constant \(c\) such that for all sufficiently \\
large \(k\) and all \(\ell \leq k\), there exists an unsatisfiable \((\ell, k)\)-CNF formula \(F = F^+ \cup F^-\) \\
with \(|F^-| \leq ck^2\rho^{-k}\) and \(|F^+| \leq ck^2(1 - \rho)^{-\ell}\).

**Proof.** We choose a set \(V = \{x_1, \ldots, x_n\}\) of \(n = k^2\) variables. There are \\
\(\binom{n}{k}\) \(k\)-clauses over \(V\) containing only negative literals. We form \(F^-\) by sampling \\
\(ck^2\rho^{-k}\) of them, uniformly with replacement, and form \(F^+\) by sampling \\
\(ck^2(1 - \rho)^{-\ell}\) purely positive \(\ell\)-clauses, where \(c\) is some suitable constant determined later. Set \(F = F^- \cup F^+\). We claim that with high probability, \(F\) is unsatisfiable. Let \(\alpha\) be any assignment. There are two cases.

**Case 1.** \(\alpha\) sets at least \(\rho n\) variables to \text{true}. For a random negative clause \\
\(C\), \\
\[\Pr[\alpha \not\models C] \geq \frac{\binom{\rho n}{k}}{\binom{n}{k}} \geq c' \rho^k,\]

The last inequality follows from the following fact about the binomial coefficient:
Lemma 7.26. Let $a, b \in \mathbb{N}$ with $b/a \leq 0.75$. Then

$$\frac{a^b}{b!} \geq \binom{a}{b} > \frac{a^b}{b!} e^{-b^2/a}.$$ 

Proof. The upper bound follows from elementary calculations and is true for all $a, b$. The lower bound holds because

$$\binom{a}{b} = \frac{a(a-1) \cdots (a-b+1)}{b!} = \frac{a^b b^{-1}}{b!} \prod_{j=0}^{b-1} \frac{a - j}{a} > \frac{a^b}{b!} e^{-2/a} \sum_{j=0}^{b-1} j \geq \frac{a^b}{b!} e^{-b^2/a},$$

where we used the fact that $1 - x > e^{-2x}$ for $0 \leq x \leq 0.75$. \hfill \Box

Since we select the clauses of $F^-$ independently of each other, we obtain

$$\Pr[\alpha \models F^-] \leq (1 - c' \rho^k) c^k \rho^{-k} < e^{-cc' k^2} \leq e^{-k^2},$$

provided we chose $c$ large enough, i.e., $c \geq \frac{1}{c'}$.

Case 2: $\alpha$ sets at most $\rho n$ variables to true. Now a similar calculation shows that $\alpha$ satisfies $F^+$ with probability at most $e^{-k^2}$.

In any case, $\Pr[\alpha \models F] \leq e^{-k^2}$. Thus, the expected number of satisfying assignments of $F$ is at most $2 k^2 e^{-k^2} \ll 1$ and with high probability $F$ is unsatisfiable. This concludes the proof of Lemma 7.25. \hfill \Box

The bound in Lemma 7.25 is tight up to a polynomial factor in $k$:

Lemma 7.27. Let $F = F^+ \cup F^-$ be an $(\ell, k)$-CNF formula. If there is a $\rho \in (0, 1)$ such that $|F^+| < \frac{1}{2} (1 - \rho)^{-\ell}$ and $|F^-| < \frac{1}{2} \rho^{-k}$, then $F$ is satisfiable.

Proof. Sample an assignment $\alpha$ by setting each variable independently to true with probability $\rho$. For a negative $k$-clause $C$, it holds that $\Pr[\alpha \not\models C] = \rho^k$. Similarly, for a positive $\ell$-clause $D$, $\Pr[\alpha \not\models D] = (1 - \rho)^\ell$. Hence the expected number of clauses in $F$ that are unsatisfied by $\alpha$ is $\rho^k |F^-| + (1 - \rho)^\ell |F^+| < \frac{1}{2} + \frac{1}{2} = 1$. Therefore, with positive probability $\alpha$ satisfies $F$. \hfill \Box

We are now ready to put everything together and prove the promised upper bounds.

Proof of point (i) of Theorem 7.20 on page 108 We construct an unsatisfiable $k$-CNF formula $F$ such that $\text{occ}_F(\overline{x}) \leq ck^2 a^k$ and $\text{occ}_F(x) \leq ck^2 b^k$, for all $x \in \text{vbl}(F)$. In fact, we construct an unsatisfiable monotone $k$-CNF formula $F = F^+ \cup F^-$. Clearly $\text{occ}_F(x) \leq |F^+|$ and $\text{occ}_F(\overline{x}) \leq |F^-|$ for all $x \in \text{vbl}(F)$. We use
Lemma 7.25 to construct such a formula. Since both $F^+$ and $F^-$ must consist of $k$-clauses, we set $\ell = k$. By setting $\rho = \frac{1}{a}$, we obtain

$$|F^-| \leq ck^2\rho^{-k} = ck^2a^k,$$

which implies $\text{occ}_F(x) \leq ck^2a^k$ for all $x \in \text{vbl}(F)$. Similarly,

$$|F^+| \leq ck^2(1 - \rho)^{-k} = ck^2\left(\frac{a - 1}{a}\right)^{-k} = ck^2\left(\frac{a}{a - 1}\right)^k \leq ck^2b^k,$$

where the last inequality comes from the assumption that $b \geq a/(a - 1)$ in point (i) of the theorem. □

Proof of Theorem 7.21 We will describe a probabilistic construction of an unsatisfiable $k$-CNF formula $F$ with $\text{ic}(F) \in O(3.01^k)$. More precisely,

$$\text{occ}_F(x) \cdot \text{occ}_F(\bar{x}) \leq ck^23.009^k,$$

for some constant $c$. We choose some $\ell \leq k$ and some $\rho \in (0, 1)$, to be determined later, and use Lemma 7.25 to construct an unsatisfiable $(\ell, k)$-CNF formula $F' = F^+ \cup F^-$ with $|F^-| \leq ck^2\rho^{-k}$ and $|F^+| \leq ck^2(1 - \rho)^{-\ell}$. Let $F$ be the $k$-CNFification of $F'$. As described above, this is a formula we obtain by introducing $k - \ell$ new variables for every $\ell$-clause $C$, and replacing $C$ by $2^{k-\ell}$ new $k$-clauses, such that any assignment satisfies $C$ if and only if it satisfies the $2^{k-\ell}$ new clauses $k$-clauses. $F$ is an unsatisfiable $k$-CNF formula, and by Lemma 7.24 on page 110, it holds that

$$\text{occ}_F(x) \cdot \text{occ}_F(\bar{x}) \leq \text{max}\{4^{k-\ell}, 2^{k-\ell}c^k\rho^{-k}(1 - \rho)^{-\ell}\} \quad \forall x \in \text{vbl}(F).$$

The constant $c$ depends on $\rho$, but not on $k$ or $\ell$. For fixed $k, \ell > 1$, the term $\rho^{-k}(1 - \rho)^{-\ell}$ is minimized for $\rho = \frac{k}{k+\ell}$. Choosing $\ell = \lceil 0.2055k \rceil$, we get $\rho \approx 0.83$ and $\text{occ}_F(x) \cdot \text{occ}_F(\bar{x}) \in O(3.01^k)$. □

Proving the Lower Bound

Proof of point (ii) of Theorem 7.20 on page 108 (ii) We are given a $k$-CNF formula $F$ with $\text{occ}_F(x) \leq \frac{b^k}{8k}$ and $\text{occ}_F(\bar{x}) \leq \frac{a^k}{8k}$ for every $x \in \text{vbl}(x)$. Further, $1 < a < \sqrt{2}$ and $b = \sqrt{\frac{a^4}{a^2 - 1}}$. We have to show that $F$ is satisfiable. We fix a probability $p := \frac{1}{\sqrt{2}}$ and set every variable of $F$ to $\text{true}$ with probability $p$, independent of each other. This gives a random assignment $\alpha$. Since $a < \sqrt{2}$, it follows that $p > \frac{1}{2}$, which means that under $\alpha$, every variable is more likely
7.3. Conflicts Generated by an Individual Variable

7.3. Conflicts Generated by an Individual Variable 113
to be set to \( \text{true} \) than to \( \text{false} \). We define \( F' \) as follows: For each clause \( C \in F \), if more than half the literals of \( C \) are negative, we remove all positive literals from \( C \) and insert the truncated clause into \( F' \), otherwise we insert \( C \) into \( F' \) without truncating it. Note that \( F' \) is “stricter” than \( F \), in the sense that every assignment that satisfies \( F' \) also satisfies \( F \). We will prove that \( F' \) is satisfiable. We write \( F' = F_k \cup F^- \), where \( F^- \) consists of purely negative clauses of size at least \( \frac{k}{2} \), and \( F_k \) consists of \( k \)-clauses, each containing at least \( \frac{k}{2} \) positive literals. A clause in \( F^- \) is unsatisfied with probability at most \( p^{\frac{k}{2}} \), and a clause in \( F_k \) is unsatisfied with probability at most \( p^{\frac{k}{2}} (1 - p)^{\frac{k}{2}} \). This is because in the worst case, half of all literals are negative: Since \( p > \frac{1}{2} \), negative literals are more likely to be unsatisfied than positive ones. Let \( C \in F' \) be any clause. A positive literal \( x \in C \) causes conflicts between \( C \) and the \( \text{occ}_{F'}(\bar{x}) \leq \frac{a^k}{8k} \) clauses of \( F' \) containing \( \bar{x} \). Similarly, a negative literal \( \bar{y} \in C \) causes conflicts with the at most \( \frac{b^k}{8k} \) clauses of \( F_k \) containing \( y \). Therefore

\[
\sum_{D \in \Gamma_{F'}(C)} \Pr[\alpha \not\models D] \leq \sum_{u \in C} \sum_{D \in \Gamma'(u)} \Pr[\alpha \not\models D] = \sum_{x \in C} \sum_{D \in \Gamma'(x)} \Pr[\alpha \not\models D] + \sum_{\bar{x} \in C} \sum_{D \in \Gamma'(x)} \Pr[\alpha \not\models D] \leq \sum_{x \in C} \sum_{D \in \Gamma'(x)} p^{\frac{k}{2}} + \sum_{\bar{x} \in C} \sum_{D \in \Gamma'(x)} p^{\frac{k}{2}} (1 - p)^{\frac{k}{2}} \leq k \cdot \frac{a^k}{8k} p^{\frac{k}{2}} + k \cdot \frac{b^k}{8k} p^{\frac{k}{2}} (1 - p)^{\frac{k}{2}} = \frac{1}{4},
\]

since \( p = \frac{1}{a^2} \) and \( b = \sqrt{\frac{a^4}{a^2 - 1}} \). By Lemma 7.6 on page 97 (the Lovász Local Lemma), \( F' \) is satisfiable.

**Remark:** Part (ii) of Theorem 7.20 on page 108 can easily be improved by defining a more careful truncation procedure: We remove all positive literals from a clause \( C \) if \( C \) contains less than \( \lambda k \) of them, for some \( \lambda \in [0, 1] \). Choosing \( \lambda \) and \( p \) optimally, we obtain a better result, but the calculations become messy, and it offers no additional insight. The crucial part of the proof is that by removing positive literals from a clause, we can use the fact that \( \text{occ}_{F}(\bar{x}) \) is small to bound the number of clauses \( D \) that conflict with \( C \) and have a large probability of being unsatisfied. We will use this idea again in the next section when proving a lower bound on the total number of conflicts in an unsatisfi-
able \( k \)-CNF formula. However, the truncation process and its analysis there will be more complicated.

## 7.4 Total Number of Conflicts

**Theorem 7.28.** Every unsatisfiable \( k \)-CNF formula contains \( \Omega (2.69^k) \) conflicts. There are unsatisfiable \( k \)-CNF formulas with \( O (3.51^k) \) conflicts.

**Proof of the upper bound in Theorem 7.28.** We want to construct an unsatisfiable \( k \)-CNF formula with \( O (3.51^k) \) conflicts. We re-use parts of the machinery from Section 7.3. We state a modified version of Lemma 7.24 on page 110. Recall the definitions of \((\ell, k)\)-CNF formulas (Definition 7.23 on page 110) and of the \( k \)-CNF-ification (Definition 7.22 on page 109).

**Lemma 7.29.** Let \( \ell \leq k \) and let \( F = F^+ \cup F^- \) be an \((\ell, k)\)-CNF formula. Let \( G \) be the \( k \)-CNF-ification of \( F \). Then \( gc(G) \leq 4^{k-\ell}\cdot |F^+| + 2^{k-\ell}\cdot |F^+| \cdot |F^-| \).

**Proof.** Every edge in the conflict graph of \( F \) runs between a positive \( \ell \)-clause \( C \) and a negative \( k \)-clause \( D \). Thus, \( gc(F) \leq |F^+| \cdot |F^-| \). In \( G \), this edge is replaced by \( 2^{k-\ell} \) edges, since \( C \) is replaced by \( 2^{k-\ell} \) copies. This explains the term \( 2^{k-\ell}\cdot |F^+| \cdot |F^-| \). Replacing \( C \) by \( 2^{k-\ell} \) many \( k \)-clauses introduces at most \( \binom{2^{k-\ell}}{2} \leq 4^{k-\ell} \) new conflicts. This explains the term \( 4^{k-\ell}|F^+| \).

We construct an unsatisfiable \((\ell, k)\)-CNF formula using Lemma 7.25 on page 110 and then take its \( k \)-CNF-ification. We choose some \( \ell \leq k \) and \( \rho \in (0, 1) \), to be determined later. By Lemma 7.25 there is an unsatisfiable \((\ell, k)\)-CNF formula \( F' = F^+ \cup F^- \) with \( |F^-| \leq c^k\cdot \rho^k \) and \( |F^+| \leq c^k(1 - \rho)^{-\ell} \). Let \( F \) be the \( k \)-CNF-ification of \( F' \). This is an unsatisfiable \( k \)-CNF formula, and by Lemma 7.29 it holds that

\[
 gc(F) \leq 4^{k-\ell}c^k(1 - \rho)^{-\ell} + 2^{k-\ell}c^2k^4\rho^k(1 - \rho)^{-\ell} .
\]

For \( \rho \approx 0.6298 \) and \( \ell = \lceil 0.333k \rceil \), we obtain \( gc(F) \in O(3.51^k) \).

**Proof of the lower bound in Theorem 7.28.** Define \( s := -\int_{1/2}^1 \frac{1}{x\ln(1-x)} dx < 0.572 \). We will prove that for every \( \epsilon > 0 \), there is a \( k_0 \) such that for \( k \geq k_0 \), every \( k \)-CNF with \( gc(F) < \frac{2^{(2-s-\epsilon)k}}{128k} \) is satisfiable. Since \( 2^{(2-0.572)} > 2.69 \), this proves the lower bound. The value of \( s \) follows from some calculation at the end of the proof and shall not bother us now.

For the rest of the proof, let \( F \) be a \( k \)-CNF formula with \( gc(F) < \frac{2^{(2-s-\epsilon)k}}{128k} \). In the proof, \( x \) denotes a variable and \( u \) a positive or negative literal. We
assume \( \text{occ}_F(\bar{x}) \leq \text{occ}_F(x) \) for all variables \( x \). We can do so since otherwise we just replace \( x \) by \( \bar{x} \) and vice versa. This changes neither \( \text{gc}(F) \) nor satisfiability of \( F \). Also we can assume that \( \text{occ}_F(x) \) and \( \text{occ}_F(\bar{x}) \) are both at least 1 for all \( x \in \text{vbl}(F) \). For \( x \), we define

\[
p(x) := \max \left\{ \frac{1}{2}, \sqrt{\frac{\text{occ}_F(x)}{16\text{gc}(F)}} \right\}.
\] (7.5)

Note that \( \text{occ}_F(x) \leq \text{gc}(F) \), and therefore \( p(x) \) will never exceed 1. Therefore we can set each variable \( x \) to \( \text{true} \) with probability \( p(x) \) independently of all other variables. This gives a random assignment \( \alpha \). We set \( p(\bar{x}) = 1 - p(x) \).

Proposition 7.30. If \( p(u) < \frac{1}{2} \) for some literal \( u \), then \( u \) is a negative literal \( \bar{x} \), and

\[
p(x) = \sqrt{\frac{\text{occ}_F(x)}{16\text{gc}(F)}} > \frac{1}{2}.
\]

If \( p(u) = \frac{1}{2} \), then both \( \sqrt{\frac{\text{occ}_F(u)}{16\text{gc}(F)}} \leq \frac{1}{2} \) and \( \sqrt{\frac{\text{occ}_F(\bar{u})}{16\text{gc}(F)}} \leq \frac{1}{2} \) hold.

We distinguish two types of clauses: Bad clauses, which contain at least one literal \( u \) with \( p(u) < \frac{1}{2} \), and good clauses, which contain only literals \( u \) with \( p(u) \geq \frac{1}{2} \). With this definition, a bad clause always contains at least one negative literal. However, a good clause can contain negative literals \( \bar{x} \), provided that \( p(\bar{x}) = 1/2 \). We write \( F = B \cup G \), where \( B \) is the set of bad clauses and \( G \) the set of good clauses.

Our goal is to show that if \( F \) is a \( k \)-CNF formula and it holds that \( \text{gc}(F) \leq 2^{(2-s-\epsilon)k}/(128k) \), then \( F \) is satisfiable. One strategy is to apply the Lovász Local Lemma with our random assignment \( \alpha \). As a first step, we prove a lemma concerning only the bad clauses. Note that for a formula \( F \) and literal \( u \), we denote defined \( \Gamma_F(u) = \{ C \in F \mid u \in C \} \) and \( \Gamma'_F(u) = \Gamma_F(\bar{u}) \).

Lemma 7.31. Let \( \alpha \) be the random assignment defined above. That is, we set each variable \( x \in V \) to 1 independently with probability \( p(x) \), as defined in (7.5). Then

\[
\sum_{C \in B} \Pr[\alpha \not\models C] \leq \frac{1}{8}.
\]

Proof. For each clause \( C \in B \), let \( u_C \) be the literal in \( C \) minimizing \( p(u) \), breaking ties arbitrarily. This means

\[
\Pr[\alpha \not\models C] = \prod_{u \in C} p(\bar{u}) \leq p(\bar{u}_C)^k.
\]
Since $C$ is a bad clause, $p(u_C) < \frac{1}{2}$, $u_C$ is a negative literal $\bar{x}_C$, and $p(x_C) = \sqrt[k]{\frac{\text{occ}_F(x_C)}{16gc(F)}}$. Thus

$$\sum_{C \in B} \Pr[\alpha \not\models C] \leq \sum_{C \in B} p(x_C)^k = \sum_{C \in B} \frac{\text{occ}_F(x_C)}{16gc(F)} .$$

(7.6)

For each clause $C \in B$, it holds that $\bar{x}_C \in C$, and therefore $C$ conflicts with the $\text{occ}_F(\bar{x}_C)$ clauses containing $x_C$. Therefore

$$\sum_{C \in B} \text{occ}_F(x_C) \leq 2gc(F) .$$

The factor 2 arises since we count each conflict possibly twice, once from each side. Combining this with (7.6) yields

$$\sum_{C \in B} \text{occ}_F(x_C) \leq 2gc(F) \leq 1/8 ,$$

which proves the lemma. □

To apply the Lovász Local Lemma (Lemma 7.6 on page 97), we have to show that (7.4) holds, i.e., for every $C \in F$,

$$\sum_{D \in \Gamma'_F(C)} \Pr[\alpha \not\models D] \leq 1/4 .$$

We split this sum into two terms, one involving bad clauses, one involving good clauses:

$$\sum_{D \in \Gamma'_F(C)} \Pr[\alpha \not\models D] = \sum_{D \in \Gamma'_B(C)} \Pr[\alpha \not\models D] + \sum_{D \in \Gamma'_G(C)} \Pr[\alpha \not\models D] \leq \frac{1}{8} + \sum_{u \in C} \sum_{D \in \Gamma_G(\bar{u})} \Pr[\alpha \not\models D] .$$

Here we use Lemma 7.31 on the previous page to bound the first sum by $1/8$. To finish the proof, one could try to show that

$$\sum_{D \in \Gamma_G(\bar{u})} \Pr[\alpha \not\models D] \leq \frac{1}{8k} .$$

(7.7)
for every literal \( u \), i.e., that the good clauses containing any given literal do not cause too much trouble. Why should this hold? Well, if the set \( \Gamma_G(\bar{u}) \) is very large, then \( \text{occ}_F(\bar{u}) \) is at least as large. By the property of our random distribution, very frequent literals are satisfied with a probability greater than \( 1/2 \). This in turn decreases the terms \( \Pr[\alpha \neq D] \) for \( D \in \Gamma_G(\bar{u}) \). However, the decrease might not be strong enough to guarantee (7.7): Consider the case that \( u = \bar{x} \) and \( x \) is extremely frequent, say \( \text{occ}_F(x) = 2.6^k \). Furthermore, suppose \( x \) occurs only in good clauses. Then \( |\Gamma_G(x)| = \text{occ}_F(x) = 2.6^k \). By the definition of our distribution, \( p(x) = \sqrt[16]{2^k/16 \cdot 2.6^k} = 0.967 \). Hence \( x \) is very likely to be set to true. However, it could be that in the clauses in \( \Gamma_G(x) \), the literal \( x \) is the only literal satisfied with a probability greater than \( 1/2 \), and therefore for \( D \in \Gamma_G(x) \), it holds that \( \Pr[\alpha \neq D] = 0.967 \cdot 2^{-k+1} \). We calculate

\[
\sum_{D \in \Gamma_G(x)} \Pr[\alpha \neq D] \approx 2.6^k \cdot 0.967 \cdot 2^{-k+1} \gg \frac{1}{8k},
\]

and our proof method fails. We have to do something smarter: We perform a truncation process, i.e., we delete certain literals from certain clauses in \( G \), arriving at a formula \( G' \) such that (7.7) holds for \( G' \). This will imply that \( B \cup G' \) is satisfiable. Since every assignment satisfying \( G' \) also satisfies \( G \), \( F \) is satisfiable, too. On the one hand, deleting literals makes \( F \) “less satisfiable”, which runs against our goal, but it also thins out the conflict structure of \( F \), making it more amenable to the Lovász Local Lemma. If we are lucky, the benefits outweigh the damage.

\section*{Algorithm 13 \texttt{truncate}(CNF formula \( F \))}

\begin{enumerate}
\item \( G' := \{ D \in F \mid p(u) \geq \frac{1}{2}, \forall u \in D \} \)
\item \textbf{while} there is a literal \( u \) such that \( \sum_{D \in \Gamma_G(u)} \Pr[\alpha \neq D] > \frac{1}{8k} \) \textbf{do}
\item \( C := \) a clause in \( G'(u) \) maximizing \( \Pr[\alpha \neq D] \)
\item \( C' := C \setminus \{u\} \)
\item \( G' := (G' \setminus \{C\}) \cup \{C'\} \)
\item \textbf{end while}
\item \textbf{return} \( F' := G' \cup B \)
\end{enumerate}

\section*{Lemma 7.32.} Let \( F' := \texttt{truncate}(F) \). If \( F' \) does not contain the empty clause, then \( F \) is satisfiable.

\textbf{Proof.} We will show that (7.4) applies to \( F' \). Fix a clause \( C \in F' \). After the truncation process, every literal \( u \) fulfills \( \sum_{D \in \Gamma_{G'}(\bar{u})} \Pr[\alpha \neq D] \leq \frac{1}{8k} \). Thus for
a clause \( C \in F' \), it holds that

\[
\sum_{D \in \Gamma'_{G'}(C)} \Pr[\alpha \not\models D] \leq \sum_{u \in C} \sum_{D \in \Gamma'_{G'}(u)} \Pr[\alpha \not\models D] \leq k \cdot \frac{1}{8k} = \frac{1}{8}.
\]

From Lemma 7.31 we conclude that \( \sum_{D \in \Gamma'_{G'}(C)} \Pr[\alpha \not\models D] \leq 1/8 \), and therefore \( \sum_{D \in \Gamma'_{F'}(C)} \Pr[\alpha \not\models D] \leq 1/4 \). By the Lovász Local Lemma (Lemma 7.6 on page 97), \( F' \) is satisfiable, and clearly \( F \) as well.

It remains to show that if \( F \) is as promised, i.e., \( F \) is a \( k \)-CNF formula and \( \text{gc}(F) < 2^{(2-s-\epsilon)k}/(128k) \), then \( \text{truncate} \) does not produce the empty clause. We will prove this by contradiction. Suppose the truncation process produces the empty clause. In this case, we will show that \( \text{gc}(F) > 2^{(2-s-\epsilon)k}/(128k) \).

If \( \text{truncate} \) produces the empty clause, then there is some \( C \in G \) all whose literals are being deleted during the truncation process. Write \( C = \{u_1, u_2, \ldots, u_k\} \), and order the \( u_i \) such that \( \text{occ}_F(u_1) \leq \text{occ}_F(u_2) \leq \cdots \leq \text{occ}_F(u_k) \). One checks that this implies that \( p(u_1) \leq p(u_2) \leq \cdots \leq p(u_k) \).

Consider any \( \ell \in \{1, \ldots, k\} \) and let \( u_j \) be the first literal among \( u_1, \ldots, u_{\ell} \) that is deleted from \( C \). Let \( C' \) denote what is left of \( C \) just before that deletion, and consider the set \( G' \) at this point of time. Then \( \{u_1, \ldots, u_\ell\} \subseteq C' \in G' \). The procedure \( \text{truncate} \) deletes \( u_j \) from the clause \( D \) that maximizes \( \Pr[\alpha \not\models D] \) among all clauses of \( \Gamma G'(u_j) \). Hence \( C' \) maximizes this probability, and

\[
\sum_{D \in \Gamma'_{G'}(u_j)} \Pr[\alpha \not\models D] \leq \Pr[\alpha \not\models C'] \cdot \text{occ}_{G'}(u_j) \leq \Pr[\alpha \not\models C'] \cdot \text{occ}_F(u_j).
\]

Since \( \alpha \) sets every variable \( x \) independently to \text{true} with probability \( p(x) \), it holds that \( \Pr[\alpha \not\models C'] = \prod_{i=1}^{\ell} (1 - p(u_i)) \). Furthermore, we ordered the literals in \( C' \) such that \( \text{occ}_F(u_1) \leq \text{occ}_F(u_2) \leq \cdots \leq \text{occ}_F(u_k) \). Therefore, we conclude that

\[
\Pr[\alpha \not\models C'] \cdot \text{occ}_F(u_j) \leq \text{occ}_F(u_\ell) \cdot \prod_{i=1}^{\ell} (1 - p(u_i)).
\]

Finally, we use the fact that \( \text{truncate} \) removes \( u_j \) from \( C' \). This means that

\[
\sum_{D \in \Gamma'G'(u_j)} \Pr[\alpha \not\models D] > \frac{1}{8k}.
\]
Combining (7.9), (7.10), and (7.11), we conclude that

\[
\frac{1}{8k} < \text{occ}_F(u_\ell) \cdot \prod_{i=1}^{\ell} (1 - p(u_i)). \tag{7.12}
\]

Since \( C \) is a good clause, it holds that \( p(u_i) \geq 1/2 \) for every literal \( u_i \in C \). Hence also \( p(u_\ell) \geq 1/2 \). By Proposition 7.30 this implies that either \( p(u_\ell) = 1/2 \), in which case \( p(u_\ell) \geq k \sqrt{\text{occ}_F(u_\ell)/16\text{gc}(F)} \), or \( p(u_\ell) > 1/2 \), in which case \( u_\ell \) is a positive literal. In the second case it follows by definition of \( p \) that \( p(u_\ell) = k \sqrt{\text{occ}_F(u_\ell)/16\text{gc}(F)} \). In either case, \( p(u_\ell)^k \geq \frac{\text{occ}_F(u_\ell)}{16\text{gc}(F)} \), and therefore

\[
\text{occ}_F(u_\ell) \leq 16\text{gc}(F)p(u_\ell)^k.
\]

Combining this with (7.12) yields

\[
\frac{1}{128k\text{gc}(F)} < p(u_\ell)^k \cdot \prod_{i=1}^{\ell} (1 - p(u_i)). \tag{7.13}
\]

And of course this inequality holds for every \( 1 \leq \ell \leq k \). At this point, we can forget everything about the formula and the truncation process, and simply observe that there exists a sequence \( q_1, \ldots, q_k \) of numbers in \([1/2, 1]\) such that

\[
\frac{1}{128k\text{gc}(F)} \leq q_1^k (1 - q_1)
\]

\[
\frac{1}{128k\text{gc}(F)} \leq q_2^k (1 - q_1)(1 - q_2)
\]

\[
\vdots
\]

\[
\frac{1}{128k\text{gc}(F)} \leq q_\ell^k \prod_{i=1}^{\ell} (1 - q_i)
\]

\[
\vdots
\]

\[
\frac{1}{128k\text{gc}(F)} \leq q_k^k \prod_{i=1}^{k} (1 - q_i)
\]

Suppose now that there is some \( \ell \in \{1, \ldots, k\} \) such that the \( \ell \)th inequality is not tight and \( q_\ell > 1/2 \). What happens when we slowly decrease \( q_\ell \)? The first, second, up to \((\ell - 1)\)st inequality stay satisfied, since they do not involve \( q_\ell \). The \((\ell + 1)\)st up to \( k \)th inequality will be happy: Their respective right sides increase as \( q_\ell \) decreases. We decrease \( q_\ell \) until either \( q_\ell = 1/2 \) or the \( \ell \)th inequality holds.
with equality. In this way, we first decrease $q_1$ as long as possible, then $q_2$, up to $q_k$. In the end, we have sequence $q_1, \ldots, q_k$ of numbers in $[1/2, 1]$ satisfying the inequalities (7.14) such that for each $1 \leq \ell \leq k$, the $\ell$th inequality holds with equality, or $q_\ell = 1/2$ (or both).

We claim that the sequence $q_1, \ldots, q_k$ we obtain is non-decreasing: Suppose otherwise. Then there would be some $1 \leq \ell < k$ such that $q_\ell > q_{\ell+1} \geq 1/2$. Hence $q_\ell > 1/2$, and the $\ell$th inequality would hold with equality. Therefore

$$\frac{1}{128k \text{gc}(F)} \leq q_{\ell+1} \prod_{i=1}^{\ell+1} (1 - q_i) < q_{\ell} \prod_{i=1}^{\ell} (1 - q_i) = \frac{1}{128k \text{gc}(F)},$$

a contradiction. We conclude that the sequence is non-decreasing.

Second, we claim that not all $q_i$ are equal to $1/2$. Suppose they were. The $k$th inequality would yield

$$\frac{1}{128k \text{gc}(F)} \leq 2^{-k} \prod_{i=1}^{k} \frac{1}{2} = 4^{-k},$$

which would imply $\text{gc}(F) \geq 4^k/(128k)$, a clear contradiction to our assumption.

We conclude that there is some $\ell^* = \min\{i \mid q_i > 1/2\}$. That is, $q_1 = q_2 = \cdots = q_{\ell^*-1} = 1/2$, and $q_{\ell^*} > 1/2$. Take a number $j$ with $\ell^* \leq j < k$. Since $q_j > 1/2$ and $q_{j+1} > 1/2$, we conclude that the $j$th and $(j + 1)$st inequalities of (7.14) are tight, and therefore

$$q_j^k \prod_{i=1}^{j} (1 - q_i) = \frac{1}{128k \text{gc}(F)} = q_{j+1}^k \prod_{i=1}^{j+1} (1 - q_i).$$

This is great, because it allows us to solve it for $q_j$:

$$q_j = q_{j+1} \sqrt{1 - q_{j+1}}.$$

We conclude that if we know $q_k$, we can compute $q_1, \ldots, q_{k-1}$, using $q_j = \max(1/2, q_{j+1} \sqrt{1 - q_{j+1}})$. We define

$$f_k(t) := t \sqrt{1 - t},$$

By $f_k^{(j)}(t)$ we denote $f_k(f_k(\ldots(f_k(t))\ldots))$, the $j$-fold iterated application of $f_k(t)$, with $f_k^{(0)}(t) = t$. We observe that $q_j = f_k^{(k-j)}(q_k) > \frac{1}{2}$ for $\ell^* \leq j \leq k$. We list several properties of $f_k$ and $f_k^{(j)}$ below in Proposition 7.34. By point (v) of
Proposition 7.34 $f_k^{(k-1)}(q_k) \leq \frac{1}{2}$, thus $\ell^* \geq 2$. Therefore $q_1 = \cdots = q_{\ell^*-1} = \frac{1}{2}$, and the $(\ell^*-1)^{th}$ inequality reads as

$$\frac{1}{128k \text{gc}(F)} \leq q_{\ell^*-1}^{k-1} \prod_{i=1}^{\ell^*-1} (1 - q_i) = 2^{-k-\ell^*+1}.$$ 

We obtain $\text{gc}(F) \geq \frac{2^{k+\ell^*-1}}{128k}$. Recall that we want to obtain a contradiction to our assumption that $\text{gc}(F) < 2^{(2-s-\epsilon)k}/(128k)$. For this, we should prove a lower bound on $\ell^*$. Define $S_k := \min\{j \in \mathbb{N}_0 \mid f_k^{(j)}(t) \leq \frac{1}{2} \forall t \in [0,1]\}$. By point (v) of Proposition 7.34, $S_k$ is finite. Since it holds that $f_k^{(k-\ell^*)}(q_k) = q_{\ell^*} > \frac{1}{2}$, we conclude that $k - \ell^* \leq S_k - 1$, thus $\text{gc}(F) \geq \frac{2^{2k-S_k}}{128k}$.

**Lemma 7.33.** The sequence $\frac{S_k}{k}$ converges to $s = -\int_{\frac{1}{2}}^{1} \frac{1}{x \ln(1-x)} \, dx < 0.572$.

By Lemma 7.33, it holds for all sufficiently large $k$ that $S_k \leq (s + \epsilon)k$, and therefore

$$\text{gc}(F) \geq \frac{2^{2k-S_k}}{128k} \geq \frac{2^{2k-(s+\epsilon)k}}{128k} = \frac{2(2-s-\epsilon)k}{128k},$$

contradicting our assumption. This finishes the proof.

**Proof of Lemma 7.33**

**Proposition 7.34.** Let $k \in \mathbb{N}$ and $f_k : [0,1] \to [0,1]$ with $f_k(t) = t \sqrt[2k]{1-t}$. For $t \in [0,1]$, the following statements hold.

(i) $f_k(t)$ attains its unique maximum at $t = t_k^* := \frac{k}{k+1}$.

(ii) $f_k(t) \leq t$, and $f_k(t) = t$ if and only if $t = 0$.

(iii) For $\ell \geq 1$, $f_k^{(\ell)}(t) \leq f_k^{(\ell)}\left(\frac{k}{k+1}\right)$.

(iv) For $\ell \geq 0$ and $t \in [0,1]$, $(1-t)^{\ell/k} t \leq f_k^{(\ell)}(t) \leq (1 - f_k^{(\ell)}(t))^{\ell/k} t$.

(v) For $k \geq 2$ and any $t \in [0,1]$, $f_k^{(k-1)}(t) \leq \frac{1}{2}$.

**Proof.** (i) follows from elementary calculus. (ii) holds since $\sqrt[2k]{1-t}$ is less than 1 for all $t > 0$. For $\ell = 1$, (iii) follows from (i), and for greater $\ell$, it follows from (ii) and induction on $\ell$. (iv) holds because each of the $\ell$ applications of $f_k$ multiplies its argument with a factor that is at least $\sqrt[2k]{1-t}$ and at most
Figure 7.4: The graph of $f_k(x)$ for $k = 4$. The function has a unique maximum, and $f_k(x) \leq x$ for all $x \in [0, 1]$.

$\sqrt{1 - f_k^{(t)}}$. Suppose (v) does not hold. Then by (iii) we get $f_k^{(k-1)} \left( \frac{k}{k+1} \right) \geq f_k^{(k-1)}(t) > \frac{1}{2}$, and by (iv), we have

$$\frac{1}{2} < f_k^{(k-1)} \left( \frac{k}{k+1} \right) \leq \left( \frac{1}{2} \right)^{k-1} \frac{k}{k+1}.$$ 

An elementary calculation shows that this does not hold for any $k \geq 1$.

To prove Lemma 7.33, we compute $\lim_{k \to \infty} \frac{S_k}{k}$ (and show that the limit exists). Recall the definition

$$S_k = \min \{ \ell \in \mathbb{N}_0 \mid f_k^{(\ell)}(t) \leq \frac{1}{2} \forall t \in [0, 1] \} ,$$

where $f_k(t) = t \sqrt{1 - t}$. By Part (iii) of Proposition 7.34 we have $S_k = \min \{ \ell \mid f_k^{(\ell)}(t^*_k) \leq \frac{1}{2} \}$, for $t^*_k := \frac{k}{k+1}$. We generalize the definition of $S_k$ by defining for $t \in (0, 1]$,

$$S_k(t) := \min \{ \ell \mid f_k^{(\ell)}(t^*_k) \leq t \} .$$

Further, we set $s_k(t) := \frac{S_k(t)}{k}$. Let $0 < t_2 < t_1 < t^*_k$. We want to estimate $s_k(t_2) - s_k(t_1)$. This should be small if $|t_1 - t_2|$ is small. For brevity, we write
\[ a := S_k(t_1), b := S_k(t_2). \] Clearly \( a \leq b. \) We calculate

\[ t_2 \geq f_k(b)(t_k^*) = f_k(b-a+1)(f_k(a-1)(t_k^*)) \geq f_k(b-a+1)(t_1) \geq (1 - t_1)((b-a+1)/k)t_1, \]

\[ t_2 < f_k(b-1)(t_k^*) = f_k(b-a-1)(f_k(a)(t_k^*)) \leq f_k(b-a-1)(t_1) \leq (1 - t_2)((b-a-1)/k)t_1. \]

Where we used part \((iv)\) of Proposition 7.34. In fact, these inequalities also hold if \( t_1 \geq t_k^* \) when \( a = 0: \)

\[ t_2 \geq f_k(b)(t_k^*) \geq (1 - t_k^*)b/k t_1 \geq (1 - t_1)(b+1)/k t_1, \]

\[ t_2 < f_k(b-1)(t_k^*) = (1 - t_2)((b-1)/k)t_1. \]

One checks that the inequalities even hold if \( t^* \leq t_2 < t_1 \leq 1. \) Note that \( \frac{b-a}{k} = s_k(t_2) - s_k(t_1). \) Solving for \( \frac{b-a}{k}, \) the above inequalities yield

\[
\frac{\log t_2 - \log t_1}{\log(1-t_1)} - \frac{1}{k} \leq s_k(t_2) - s_k(t_1) \leq \frac{\log t_2 - \log t_1}{\log(1-t_2)} + \frac{1}{k},
\] (7.15)

for all \( 0 < t_2 < t_1 < 1. \) The right inequality also holds for \( 0 < t_2 < t_1 \leq 1. \) Multiplying with \(-1, \) we see that it also holds if \( t_2 > t_1. \) If \( t_2 = t_1, \) it is trivially true. Hence this inequality is true for all \( t_1, t_2 \in (0, 1). \)

Suppose \( s(t) = \lim_{k \to \infty} s_k(t) \) exists, for every fixed \( t. \) Inequality (7.15) also holds in the limit. Writing \( t_1 = t \) and \( t_2 = t + h \) and dividing (7.15) by \( h \) gives

\[
\frac{\log(t + h) - \log t}{h \log(1-t)} \leq \frac{s(t + h) - s(t)}{h} \leq \frac{\log(t + h) - \log t}{h \log(1 - t - h)},
\]

When we let \( h \) go to \( 0, \) we obtain \( s'(t) = \frac{1}{t \log(1-t)}, \) thus \( s(t) = s(1) - \int_t^1 \frac{1}{x \log(1-x)} dx. \) Observing that \( \frac{S_k}{k} = s_k(\frac{1}{2}) \) and \( s_k(1) = 0 \) for all \( k \) proves the Lemma.

The above argument shows that if \( s_k(t) \) converges pointwise, then it converges to a continuous function \( s(t) \) on \((0, 1). \) We have to show that \( \lim_{k \to \infty} s_k(t) \) does in fact exist. First plug in \( t_1 = 1 \) into the right inequality of (7.15) to observe that for each fixed \( t_2, \) the sequence \( s_k(t_2) \) is bounded from above. Clearly it is bounded from below by \( 0. \) Hence there exist \( \overline{s}(t) := \limsup s_k(t) \) and similarly \( \underline{s}(t) := \liminf s_k(t). \) We write shorthand \( L(t_1, t_2) := \frac{\log t_2 - \log t_1}{\log(1-t_1)} \) and \( U(t_1, t_2) := \frac{\log t_2 - \log t_1}{\log(1-t_2)}. \) Now (7.15) reads as \( L(t_1, t_2) - \frac{1}{k} \leq s_k(t_2) - s_k(t_1) \leq U(t_1, t_2) + \frac{1}{k}. \) We claim that

\[
L(t_1, t_2) \leq \overline{s}(t_2) - \overline{s}(t_1) \leq U(t_1, t_2), \quad (7.16)
\]

\[
L(t_1, t_2) \leq \underline{s}(t_2) - \underline{s}(t_1) \leq U(t_1, t_2). \quad (7.17)
\]
For sequences \((a_k)_{k \in \mathbb{N}}, (b_k)_{k \in \mathbb{N}}\), \(\limsup a_k - \limsup b_k = \limsup(a_k - b_k)\) does not hold in general, hence the claim is now completely trivial. We will proof that \(\overline{s}(t_2) - \underline{s}(t_1) \leq U(t_1, t_2)\). This will prove one claimed inequality. The other three inequalities can be proven similarly. Fix some small \(\epsilon > 0\). For all sufficiently large \(k\), \(\frac{1}{k} \leq \epsilon\). We have \(s_k(t_2) \geq \overline{s}(t_2) - \epsilon\) for infinitely many \(k\), thus \(s(t_1) - U(t_1, t_2) - \frac{1}{k} \geq \overline{s}(t_2) - U(t_1, t_2) - 2\epsilon\) for infinitely many \(k\). Therefore \(\overline{s}(t_1) \geq s(t_2) - U(t_1, t_2) - 2\epsilon\). By making \(\epsilon\) arbitrarily small, the claimed inequality follows.

We can now apply our non-rigorous argument from above, this time rigorously. Write \(t = t_1, t_2 = t + h\), and divide (7.16) and (7.17) by \(h\), send \(h\) to 0, and we obtain \(\overline{s}'(t) = \overline{s}'(t) = \frac{1}{t \log(1 - t)}\). Since \(\overline{s}(1) = \overline{s}(1) = 0\), we obtain

\[
\overline{s}(t) = s(t) = \int_t^1 \frac{-1}{x \log(1 - x)} \, dx.
\]
Chapter 8

Linear Formulas

Hence, $F_v$ is satisfied. And so are we.

Emo Welzl, personal communication

8.1 Introduction

The results of this chapter appeared in [Sch10]. Well call a CNF formula $F$ linear if $|\text{vbl}(C) \cap \text{vbl}(D)| \leq 1$ for any two distinct clauses $C, D \in F$.

For example, the CNF formula \{\{\bar{x}_1, x_2\}, \{x_1, x_2\}, \{x_3, x_4\}, \{\bar{x}_4, \bar{x}_1\}\} is linear, but the formula \{\{\bar{x}_1, x_2\}, \{x_1, x_2\}, \{x_2, x_3\}\} is not. Are there unsatisfiable linear CNF formulas? Trivially, yes: \{\Box\} is unsatisfiable, and \{\{x\}, \{\bar{x}\}\} is unsatisfiable, too, and both are obviously linear. Okay, but are there linear $k$-CNF formulas? Now it becomes interesting. For $k = 2$, the answer is yes: The formula

\[
\{\{\bar{x}_1, x_2\}, \{x_2, x_3\}, \{\bar{x}_3, x_4\}, \{\bar{x}_4, x_1\}, \{\{x_1, x_3\}, \{\bar{x}_2, \bar{x}_4\}\}
\]

is linear and it is unsatisfiable. To see this, note that the first four clauses are satisfied if and only if either all variables are $0$ or all variables are $1$. However, the last two clauses forbid exactly those assignments. Some case analysis shows that every unsatisfiable linear $2$-CNF formula has at least four variables and at least six clauses. On the other hand, the $2$-CNF formula \{\{x, y\}, \{x, \bar{y}\}, \{\bar{x}, y\}, \{\bar{x}, \bar{y}\}\}, which is not linear, is unsatisfiable and has four clauses. Jumping to conclusions, we observe
Unsatisfiable linear $k$-CNF formulas seem to be larger than general unsatisfiable $k$-CNF formulas.

What about $k = 3$? This takes considerably more effort, but if you try for a while, you will also find an unsatisfiable linear 3-CNF formula. I will not give any example, since the smallest one I have found has 30 clauses. Achieving 48 clauses is rather easy after Theorem 8.1 below, 32 clauses is slightly more tricky, and from there one can save two clauses. As seen above, any unsatisfiable 3-CNF formula has at least 8 clauses, and by some case analysis one gets a lower bound of maybe 13 or 14 for linear 3-CNF formulas. I have no idea what the smallest possible number is. The subject of this chapter will be the existence, size, and structure of unsatisfiable linear $k$-CNF formulas.

It is not obvious whether unsatisfiable linear $k$-CNF formulas exist for every $k$. These questions have been asked first by Porschen, Speckenmeyer and Randerath [PSR06], who also proved that for any $k \geq 3$, if an unsatisfiable linear $k$-CNF formula exists, then deciding satisfiability of linear $k$-CNF formulas is NP-complete. Later, Porschen, Speckenmeyer and Zhao [PSZ09] and, independently, myself [Sch07] gave a construction of unsatisfiable linear $k$-CNF formulas, for every $k \in \mathbb{N}$:

**Theorem 8.1 ([PSZ09], [Sch07]).** For every $k \geq 0$, there exists an unsatisfiable linear $k$-CNF formula $F_k$, with $F_0$ containing one clause and $F_{k+1}$ containing $|F_k|^2 |F_k|$ clauses.

We will prove this and the next theorems in the following sections. We obtain the formulas $F_k$ from Theorem 8.1 by a simple explicit construction. However, from an extremal combinatorist’s point of view, these formulas are disastrous, because their size (measured in number of clauses) is gigantic. In fact, $F_k$ is greater than $2^{2^{2^k}}$, where the height of the tower is $k$. We will find much smaller linear formulas by using a probabilistic construction and also prove that their size is as small as possible, up to a polynomial factor in $k$.

Let us introduce a relaxation of linearity. We call $F$ weakly linear if $|C \cap D| \leq 1$ for any distinct $C, D \in F$. For example, the formula $\{\{x_1, x_2\}, \{x_1, x_2\}, \{x_2, x_3\}\}$ is weakly linear, but not linear, and finally $\{\{x_1, x_2, x_3\}, \{x_1, x_2, \overline{x}_3\}\}$ is not weakly linear (and not linear, either). Our results do not much depend on which notion one uses (linear or weakly linear). Typically we will state the strongest version of each result, i.e., proving the existence of certain unsatisfiable linear formulas and proving lower bounds for weakly linear formulas.

**Theorem 8.2.** All weakly linear $k$-CNF formulas with at most $\frac{4^k}{8e^2(k-1)^2} - \frac{2^k}{4e(k-1)}$ clauses are satisfiable. There exists an unsatisfiable linear $k$-CNF formula with $4k^2 4^k$
8.1. Introduction

It is a common phenomenon in extremal combinatorics that by probabilistic means one can show that a certain object exists (in our case, a “small” linear unsatisfiable $k$-CNF formula), but one cannot explicitly construct it. We have no explicit construction avoiding the tower-like growth in Theorem 8.1. We give some arguments why this is so, and show that small linear unsatisfiable $k$-CNF formulas have a more complex structure than their non-linear relatives. To do so, we speak about resolution.

Resolution Trees

If $C$ and $D$ are clauses and there is unique literal $u$ such that $u \in C$ and $\bar{u} \in D$, then $(C \setminus \{u\}) \cup (D \setminus \{\bar{u}\})$ is called the resolvent of $C$ and $D$. It is easy to check that every assignment satisfying $C$ and $D$ also satisfies their resolvent.

**Definition 8.3.** A resolution tree for a CNF formula $F$ is a rooted tree $T$ whose vertices are labeled with clauses, such that

- each vertex is either a leaf or has two children,
- each leaf of $T$ is labeled with a clause of $F$,
- the root of $T$ is labeled with the empty clause,
- if vertex $a$ has children $b$ and $c$, and these three vertices are labeled with clauses $C_a, C_b, C_c$, respectively, then $C_a$ is the resolvent of $C_b$ and $C_c$.

It is well-known that a CNF formula $F$ is unsatisfiable if and only if it has a resolution tree. Proving lower bounds on the size of resolution trees (and general resolution proofs, which we will not introduce here) has been and still is an area of intensive research. See for example Ben-Sasson and Wigderson [BSW01].

**Theorem 8.4.** Let $k \geq 2$. Every resolution tree of an unsatisfiable weakly linear $k$-CNF formula has at least $2^{2\frac{k}{2} - 1}$ leaves.

We do not know whether $2^{2\frac{k}{2} - 1}$ is asymptotically the correct bound. Neither do we know whether one can prove good lower bounds on the general resolution complexity of linear formulas (general meaning not necessarily treelike). A large ratio between the size of $F$ and the size of a smallest resolution tree is an indication that $F$ has a complex structure. For example, it is well-known that the running time of so-called Davis-Putnam procedures [DP60, DLL62] on a formula $F$ is lower bounded by the size of the
smallest resolution tree of $F$. Such a procedure tries to find a satisfying assignment for a formula $F$ (or to prove that none exists) by choosing a variable $x$, and then recursing on the formulas $F[x \mapsto 0]$ and $F[x \mapsto 1]$, obtained from $F$ by fixing the value of $x$ to 0 or 1, respectively. If $F$ is unsatisfiable, the procedure implicitly constructs a resolution tree.

**Strictly Treelike Formulas**

A CNF formula $F$ is *minimal unsatisfiable* if it is unsatisfiable, and for every clause $C \in F$, $F \setminus \{C\}$ is satisfiable. For example, the complete $k$-CNF formula $CF(x_1, \ldots, x_k)$ is minimal unsatisfiable, and has a resolution tree with $2^k$ leaves, one for every clause. This is as small as possible, since for a minimal unsatisfiable formula, every clause must appear as the label of at least one leaf of any resolution tree.

**Definition 8.5.** A resolution tree strict if no two leaves are labeled by the same clause. A CNF formula $F$ is strictly treelike if it has a strict resolution tree.

In some sense, strictly treelike formulas are the least complex formulas possible. For example, the complete formula $K_k$ and the formulas constructed in the proof of Theorem 8.1 are strictly treelike.

**Theorem 8.6.** For every $\epsilon > 0$, there exists a constant $c$ such that for all $k \in \mathbb{N}$, every strictly treelike weakly linear $k$-CNF formula has at least $\text{tower}_2(k-c)$ clauses, where $\text{tower}_a(n)$ is defined by $\text{tower}_a(0) = 1$ and $\text{tower}_a(n + 1) = a^{\text{tower}_a(n)}$.

Strictly treelike formulas appear in other contexts, too. Consider MU(1), the class of minimal unsatisfiable formulas whose number of variables is one less than the number of clauses. A result of Davydov, Davydova and Kleine Büning ([DDB98], Theorem 12) implies that every MU(1)-formula is strictly treelike. Also, MU(1)-formulas serve as “universal patterns” for unsatisfiable formulas: Szeider [Sze03] shows that a formula $F$ is unsatisfiable if and only if it can be obtained from a MU(1)-formula $G$ by renaming the variables of $G$ (in a possibly non-injective manner). It is not difficult to show that a strictly treelike linear $k$-CNF formula can be transformed into a linear MU(1)-formula with at most as many clauses.

MU(1)-formulas appear in another familiar context. Recall the extremal parameter $f(k) := \max(\deg, k$-CNF), which is the greatest integer $d$ such that any $k$-CNF formulas $F$ with $\deg(F) \leq d$ is satisfiable. The best upper bounds on $f(k)$ come from MU(1)-formulas constructed by Gebauer, Szabó and Tardos [GST10]. We define $f_1(k)$ to be the greatest integer $d$ such that there is no MU(1)-formula $F$ with $\deg(F) \leq d$ (note that MU(1)-formulas are always
unsatisfiable). This function was defined first by Hoory and Szeider \cite{HS05} when investigating $f(k)$ for small values of $k$. Clearly $f(k) \leq f_1(k)$, and all values of $k$ for which we know the value of $f(k)$ exactly (which is up to $k = 5$), it holds that $f(k) = f_1(k)$. To summarize: Hoory and Szeider \cite{HS05} show $f(k) = f_1(k)$ for $k \leq 5$, and Gebauer, Szabó and Tardos \cite{GST10} show that $f(k)$ and $f_1(k)$ have the same asymptotic behavior, i.e., $f(k)/f_1(k)$ converges to 1 as $k$ grows.

MU(1)-formulas work fine in providing tight upper bounds for $f(k)$. Unfortunately, in the context of linear formulas, the upper bounds achieved by MU(1)-formulas perform horribly, as we show in Theorem \ref{cor:MU1}.

While interest in linear CNF formulas is rather young, linear hypergraphs have been studied for quite some time. A hypergraph $H = (V, E)$ is linear if $|e \cap f| \leq 1$ for any two distinct hyperedges $e, f \in E$. A $k$-uniform hypergraph is a hypergraph where every hyperedge has cardinality $k$. We ask when a hypergraph is 2-colorable, i.e., admits a 2-coloring of its vertices such that no hyperedge becomes monochromatic. Bounds on the number of edges in such a hypergraph were given by Erdős and Lovász \cite{EL75} (this is the paper in which the Local Lemma appeared first). They show that there are non-2-colorable linear $k$-uniform hypergraphs with $ck^4/A_k$ hyperedges, but not with less than $c'k^4/A_k$. The proof of the lower bound directly translates into our lower bound for linear $k$-CNF formulas. For the number of edges in linear $k$-uniform hypergraphs that are not 2-colorable, the currently best upper bound is $ck^2/A_k$ by Kostochka and Rödl \cite{KR}, and the best lower bound is $k^{-\epsilon}4^k$, for any $\epsilon > 0$ and sufficiently large $k$, due to Kostochka and Kumbhat \cite{KK}.

### 8.2 Existence and Size

**Proof of Theorem 8.1 on page 126.** Choose $F_0$ to be the formula consisting of only the empty clause. Suppose we have constructed $F_k$, and want to construct $F_{k+1}$. Let $m = |F_k|$. We create $m$ new variables $x_1, \ldots, x_m$, and let $K_m = \{D_1, D_2, \ldots, D_{2^m}\}$ be the complete $m$-CNF formula over $x_1, \ldots, x_m$. It is unsatisfiable, but not linear. We take $2^m$ variable disjoint copies of $F_k$, denoted by $F^{(1)}_k, F^{(2)}_k, \ldots, F^{(2^m)}_k$. For each $1 \leq i \leq 2^m$, we build a linear $(k + 1)$-CNF formula $\tilde{F}^{(i)}_k$ from $F^{(i)}_k$ by adding, for each $1 \leq j \leq m$, the $j^{th}$ literal of $D_i$ to the $j^{th}$ clause of $F^{(i)}_k$. Note that every assignment satisfying $\tilde{F}^{(i)}_k$ also satisfies $D_i$. Finally, we set $F_{k+1} := \bigcup_{i=1}^{2^m} \tilde{F}^{(i)}_k$. This is an unsatisfiable linear $(k + 1)$-CNF formula with $m2^m$ clauses. 

Using induction, it is not difficult to see that the formulas $F_k$ are strictly
Having proved the existence of unsatisfiable linear \( k \)-CNF formulas, we will now prove the upper bound stated in Theorem 8.2 on page 127. We give a probabilistic construction of a comparably small unsatisfiable linear \( k \)-CNF formula. Our construction consists of two steps. First, we construct a linear \( k \)-uniform hypergraph \( H \) that is “dense” in the sense that \( \frac{m}{n} \) is large, where \( m \) and \( n \) are the number of hyperedges and vertices, respectively, and then transform it randomly into a linear \( k \)-CNF formula \( F \) that is unsatisfiable with high probability. We explain the second step first, because it is conceptually simpler.

**Lemma 8.7.** If there is a linear \( k \)-uniform hypergraph \( H \) with \( n \) vertices and \( m \) edges such that \( \frac{m}{n} \geq 2^k \), then there is an unsatisfiable linear \( k \)-CNF formula with \( m \) clauses.

**Proof.** Let \( H = (V, E) \). By viewing \( V \) as a set of variables and \( E \) as a set of clauses (each containing only positive literals), this is a (satisfiable) linear \( k \)-CNF formula. We replace each literal in each clause by its complement with probability \( \frac{1}{2} \), independently in each clause. Let \( F \) denote the resulting (random) formula. For any fixed assignment \( \alpha \), it holds that \( \Pr[\alpha \text{ satisfies } F] = (1 - 2^{-k})^m \). Hence the expected number of satisfying assignments of \( F \) is

\[
2^n (1 - 2^{-k})^m < 2^n e^{-2^{-k}m} = e^{\ln(2)n - 2^{-k}m} \leq 1,
\]

where the last inequality follows from \( \frac{m}{n} \geq 2^k \). Hence some formula \( F \) has fewer than one satisfying assignment, i.e., none. \( \square \)

How can we construct such a dense linear hypergraph? We use a construction by Kuzjurin [Kuz95]. Our application of this construction is motivated by Kostochka and Rödl [KR], who use it to construct linear hypergraphs of large chromatic number.

**Lemma 8.8.** For every prime power \( q \) and every \( k \in \mathbb{N} \), there exists a \( k \)-uniform linear hypergraph with \( kq \) vertices and \( q^2 \) edges.

With \( n = kq \), this hypergraph has \( n^2/k^2 \) hyperedges. This is almost optimal, since any linear \( k \)-uniform hypergraph on \( n \) vertices has at most \( \binom{n}{2}/\binom{k}{2} \) hyperedges: The \( n \) vertices provide us with \( \binom{n}{2} \) vertex pairs. Each hyperedge occupies \( \binom{k}{2} \) pairs, and because of linearity, no pair can be occupied by more than one hyperedge.

**Proof.** Choose the vertex set \( V = V_1 \uplus \cdots \uplus V_k \), where each \( V_i \) is a disjoint copy of the finite field \( GF(q) \). The hyperedges consist of all \( k \)-tuples \( (x_1, \ldots, x_k) \)
with \( x_i \in V_i, 1 \leq i \leq k \), such that

\[
\begin{pmatrix}
1 & 1 & \cdots & 1 & 1 \\
1 & 2 & \cdots & i & \cdots & k \\
1 & 4 & \cdots & i^2 & \cdots & k^2 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
1 & 2^{k-3} & \cdots & i^{k-3} & \cdots & k^{k-3}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_i \\
\vdots \\
x_k
\end{pmatrix} = 0.
\] (8.1)

Consider two distinct vertices \( x \in V_i, y \in V_j \). How many hyperedges contain both of them? If \( i = j \), none. If \( i \neq j \), we can find out by plugging the fixed values \( x, y \) into (8.1). We obtain a (possibly non-uniform) \((k-2) \times (k-2)\) linear system with a Vandermonde matrix, which has a unique solution. In other words, \( x \) and \( y \) are in exactly one hyperedge, and the hypergraph is linear. Since every pair \((x, y) \in V_1 \times V_2\) is contained in exactly one hyperedge, there are exactly \( q^2 \) hyperedges.

\[\square\]

**Proof of the upper bound in Theorem 8.2 on page 127.** We choose a prime power \( q \in \{k2^k, \ldots, 2k2^k - 1\} \). By Lemma 8.8, there is a linear \( k \)-uniform hypergraph \( H \) with \( n = qk \) vertices and \( m = q^2 \) hyperedges. Since \( \frac{m}{n} = \frac{q}{k} \geq 2^k \), Lemma 8.7 shows that there is an unsatisfiable linear \( k \)-CNF formula with \( q^2 \leq 4k^2 4^k \) clauses.

\[\square\]

Let us prove the lower bound of Theorem 8.2 on page 127. For a literal \( u \) and a CNF formula \( F \), we write \( \text{occ}_F(u) := |\{C \in F \mid u \in C\}| \), the degree of the literal \( u \). Thus \( d_F(x) = \text{occ}_F(x) + \text{occ}_F(\overline{x}) \). We write \( \text{occ}(F) = \max_u \text{occ}_F(u) \). In analogy to \( f(k) \), we define \( f_{\text{occ}}(k) \) to be the largest integer \( d \) such that any \( k \)-CNF formula \( F \) with \( \text{occ}(F) \leq d \) is satisfiable. Clearly \( f_{\text{occ}}(k) \geq \left\lfloor \frac{f(k)}{2} \right\rfloor \), and thus from [KST93] it follows that \( f_{\text{occ}}(k) \geq \left\lfloor \frac{2^k}{2ek} \right\rfloor \). Actually, an application of the Lopsided Lovász Local Lemma [ES91, AS00, LS07] yields \( f_{\text{occ}}(k) \geq \left\lfloor \frac{2^k}{ek} - 1 \right\rfloor \).

**Lemma 8.9.** Let \( F \) be a weakly linear \( k \)-CNF formula in which there are most \( 1 + f_{\text{occ}}(k - 1) \) literals of degree at least \( 1 + f_{\text{occ}}(k - 1) \). Then \( F \) is satisfiable.

**Proof.** Transform \( F \) into a \((k-1)\)-CNF formula \( F' \) by removing in every clause in \( F \) a literal of maximum degree. We claim that \( \text{deg}_{F'}(u) \leq f_{\text{occ}}(k - 1) \) for every literal \( u \). Therefore \( F' \) is satisfiable, and \( F \) is, as well.

For the sake of contradiction, suppose there is a literal \( u \) such that \( t := \text{occ}_{F'}(u) \geq 1 + f_{\text{occ}}(k - 1) \). Let \( C'_i, i = 1, 2, \ldots, t, \) be the clauses in \( F' \) containing
u. $C'_i$ is obtained by removing some literal $v_i$ from some clause $C_i \in F$. By construction of $F'$, $\text{occ}_F(v_i) \geq \text{occ}_F(u) \geq f_{\text{occ}}(k - 1) + 1$ for all $1 \leq i \leq t$. The $v_i$ are pairwise distinct: If $v_i = v_j$, then $\{u, v_i\} \subseteq C_i \cap C_j$. Since $F$ is weakly linear, this can only mean $i = j$. Now $u, v_1, v_2, \ldots, v_t$ are $t + 1 \geq 2 + f_{\text{occ}}(k - 1)$ variables of degree at least $1 + f_{\text{occ}}(k - 1)$ in $F$, a contradiction.

We see that an unsatisfiable weakly linear $k$-CNF formula has at least $f_{\text{occ}}(k - 1) + 2 \geq \left\lceil \frac{2^k}{2e(k - 1)} + 1 \right\rceil$ literals of degree at least $f_{\text{occ}}(k - 1) + 1 \geq \left\lfloor \frac{2^k}{2e(k - 1)} \right\rfloor$. Double counting yields $k|F| = \sum_u \text{occ}_F(u) \geq (f_{\text{occ}}(k - 1) + 2)(f_{\text{occ}}(k - 1) + 1) \in \Omega(4^k/k^3)$. By a more careful argument, we can improve this by a factor of $k$.

We call a hypergraph $(j, d)$-rich if at least $j$ vertices have degree at least $d$. The following lemma is due to Welzl [Wel].

**Lemma 8.10.** For $d \in \mathbb{N}_0$, every linear $(d, d)$-rich hypergraph has at least $\binom{d + 1}{2}$ edges. This bound is tight for all $d \in \mathbb{N}_0$.

**Proof.** We proceed by induction over $d$. Clearly, the assertion of the lemma is true for $d = 0$. Now let $H = (V, E)$ be a linear $(d, d)$-rich hypergraph for $d \geq 1$. Choose some vertex $v$ of degree at least $d$ in $H$ and let $H' = (V, E')$ be the hypergraph with $E' := E \setminus \{e \in E \mid e \ni v\}$. We have (i) $|E| \geq |E'| + d$, (ii) $H'$ is linear, since this property is inherited when edges are removed, and (iii) $H'$ is $(d - 1, d - 1)$-rich, since for no vertex other than $v$ the degree decreases by more than 1 due to the linearity of $H$. It follows that $|E| \geq \binom{d}{2} + d = \binom{d + 1}{2}$. The complete $2$-uniform hypergraph (graph, so to say) on $d + 1$ vertices shows that the bound given is tight for all $d \in \mathbb{N}_0$. □

**Proof of the lower bound in Theorem 8.2 on page 127.** Let $F$ be a weakly linear $k$-CNF formula. $F$ is a linear $k$-uniform hypergraph, with its literals as vertices and its clauses as hyperedges. If $F$ is unsatisfiable, then by Lemma 8.9 it is $(f_{\text{occ}}(k - 1) + 1, f_{\text{occ}}(k - 1) + 1)$-rich. By Lemma 8.10, $F$ has at least $(f_{\text{occ}}(k - 1) + 2) > \frac{4^k}{8e^2(k - 1)^2} - \frac{2^k}{4e(k - 1)}$ clauses. □

### 8.3 Resolution Complexity

Let $F$ be an unsatisfiable weakly linear $k$-CNF formula, and let $T$ be a resolution tree of minimum size of $F$. We want to show that $T$ has a large number of nodes. It is not difficult to see that a resolution tree of minimum size is regular, meaning that no variable is resolved more than once on a path from a leaf to the root. See Urquhart [Urq95], Lemma 5.1, for a proof of this fact. We fix some natural number $\ell$, to be determined later. Consider a random walk of
8.3. Resolution Complexity

Figure 8.1: A resolution tree, with its edges labeled in the obvious way. The assignment defined by the edges on the path from a node $u$ to the root unsatisfies the clause with which $u$ is labeled.

length $\ell$ in $T$ starting at the root, in every step choosing randomly to go to one of the two children of the current node. If we arrive at a leaf, we stay there. We claim that if $\ell \leq \sqrt{2^{k-2}}$, then with probability at least $\frac{1}{2}$, our walk does not end at a leaf. This means that there are at least $2^{\ell-1}$ distinct paths of length $\ell$ at start at the root. The endpoints of these paths are all distinct inner nodes, hence the tree has at least $2^{\ell-1}$ inner nodes, thus at least $2^\ell$ nodes.

As illustrated in Figure 8.1, we label each edge in $T$ with an assignment. If $C$ is the resolvent of $D_1$ and $D_2$, $x \in D_1$ and $\bar{x} \in D_2$, we label the edge from $C$ to $D_1$ by $x \mapsto 0$ and from $C$ to $D_2$ by $x \mapsto 1$. Each path from the root to a node gives a partial assignment $\alpha$. If that node is labeled with clause $C$, then $C$ evaluates to false under $\alpha$. In our random walk, let $\alpha_i$ denote the partial assignment associated with the first $i$ steps. Then $\alpha_0$ is the empty assignment, and $\alpha_i$ assigns exactly $i$ variables (if we are not yet at a leaf). We set $F_i := F[\alpha_i]$, i.e., the formula obtained from $F$ by fixing the variables according to the partial assignment $\alpha_i$. For a formula $G$, we define the weight $w(G)$ to be

$$w(G) := \sum_{C \in G, |C| \leq k-2} 2^{k-|C|}.$$  

(8.2)

Since $F$ is a $k$-CNF formula, $w(F) = 0$. If some formula $G$ contains the empty clause, then $w(G) \geq 2^k$. In our random walk, $w(F_i)$ is a random variable.

**Lemma 8.11.** $E[w(F_{i+1})] \leq E[w(F_i)] + 4i$.

Since $w(F_0) = 0$, this implies $E[w(F_\ell)] \leq 4\binom{\ell}{2} \leq 2\ell^2$. If our random walk ends at a leaf, then $F_\ell$ contains the empty clause, thus $w(F_\ell) \geq 2^k$. Therefore

$$2\ell^2 \geq E[w(F_\ell)] \geq 2^k \Pr[\text{the random walk ends at a leaf}],$$
and consequently \( \Pr[\text{the random walk ends at a leaf}] \leq 2\ell^2/2^k \). Choosing \( \ell^* = \sqrt{2^{k-2}} \), we conclude that at least half of all paths of length \( \ell \) starting at the root do not end at a leaf. Thus \( T \) has at least \( 2\ell^{*-1} \) internal nodes at distance \( \ell^* \) from the root, and thus at least \( 2\ell^{*} \) leaves, which proves Theorem 8.4. It remains to prove the lemma.

**Proof of the lemma.** For a formula \( G \) and a variable \( x \), let \( d_{k-1}(x, G) \) denote the number of \((k - 1)\)-clauses containing \( x \) or \( \bar{x} \). Since \( F_0 \) is a \( k-CNF \) formula, \( d_{k-1}(x, F_0) = 0 \), for all variables \( x \). We claim that \( d_{k-1}(x, F_{i+1}) \leq d_{k-1}(x, F_i) + 2 \) for every variable \( x \). To see this, note that in step \( i \), some variable \( y \) is set to \( b \in \{0, 1\} \), say to 0. At most one \( k \)-clause of \( F_i \) contains \( y \) and \( x \), and at most one contains \( y \) and \( \bar{x} \), since \( F_i \) is weakly linear, thus \( d_{k-1}(x, F_{i+1}) \leq d_{k-1}(x, F_i) + 2 \). It follows immediately that \( d_{k-1}(x, F_i) \leq 2i \).

Consider \( w(F_i) \), which was defined in (8.2). \( F_{i+1} \) is obtained from \( F_i \) by setting some variable \( y \) randomly to 0 or 1. Let \( C \) be some clause. How does its contribution to (8.2) change when setting \( y \)? If (i) \( y \notin \text{vbl}(C) \) or \( |C| = k \), it does not change. If (ii) \( y \in \text{vbl}(C) \) and \( |C| \leq k-2 \), then with probability \( \frac{1}{2} \) each, its contribution to (8.2) doubles or vanishes. Hence on expectation, it does not change. If (iii) \( y \in \text{vbl}(C) \) and \( |C| = k - 1 \), then \( C \) contributes nothing to \( w(F_i) \), and with probability \( \frac{1}{2} \), it contributes 4 to \( w(F_{i+1}) \). On expectation, its contribution to (8.2) increases by 2. Case (iii) applies to at most \( d_{k-1}(y, F_i) \leq 2i \) clauses. Hence \( \mathbb{E}[w(F_{i+1})] \leq \mathbb{E}[w(F_i)] + 4i \). □

### 8.4 Linear MU(1)-Formulas

Recall the definition of strict resolution trees and strictly treelike CNF formulas (Definition 8.5 on page 128). Let \( F \) be a strictly treelike weakly linear \( k-CNF \) formula \( F \), and let \( T \) be a strict resolution tree of \( F \). We want to prove Theorem 8.6 on page 128. Loosely speaking, we want to show that the number of clauses in \( F \) is a tower function in \( k \). Equivalently, we can show that the size of \( T \) is a tower function in \( k \). The proof is somewhat technical and requires some notation.

Letters \( a, b, c \) denote nodes of \( T \), and \( u, v, w \) denote literals. Every node \( a \) of \( T \) is labeled with a clause \( C_a \). We define a graph \( G_a \) with vertex set \( C_a \), connecting \( u, v \in C_a \) with an edge if \( u, v \in D \) for some clause \( D \in F \) that occurs as a label of a leaf in the subtree of \( T \) rooted at \( a \). Since \( T \) is a strict resolution tree and \( F \) is weakly linear, every edge in \( G_a \) comes from a unique leaf of \( T \). If \( a \) is a leaf, then \( G_a = K_k \). Since the root of a resolution tree is labeled with the empty clause, we have \( G_{\text{root}} = (\emptyset, \emptyset) \). Resolution now has a
simple interpretation as a "calculus on graphs", see Figure 8.2. The complete graph $K_k$ is the axiom, and for a derivation step, we take two graphs $G_1$ and $G_2$, overlay them in an edge-disjoint manner and remove one vertex of the resulting graph. The goal of this calculus is to derive the graph $(\emptyset, \emptyset)$.

**Definition 8.12.** For a graph $G$, let $\kappa_i(G)$ denote the minimum size of a set $U \subseteq V(G)$ such that $G - U$ contains no $i$-clique. Here, $G - U$ is the subgraph of $G$ induced by $V(G) \setminus U$.

For example, $\kappa_1(G) = |V(G)|$, and $\kappa_2(G)$ is the size of a minimum vertex cover of $G$. For the complete graph $K_k$, $\kappa_i(K_k) = k - i + 1$. We write $\kappa_i(a) := \kappa_i(G_a)$. The tuple $(\kappa_1(a), \ldots, \kappa_k(a))$ can be viewed as the complexity measure for $a$. We observe that if $a$ is a leaf, then $\kappa_i(a) = k - i + 1$, and $\kappa_i(\text{root}) = 0$, for all $1 \leq i \leq k$.

**Proposition 8.13.** Let $a$ be an inner node in $T$ and let $b$ be one of the two children of $a$. Then $\kappa_i(b) \leq \kappa_i(a) + 1$ for $1 \leq i \leq k$.

**Proof.** Let $c$ be the other child of $a$. There is some vertex $u \in G_b$ such that $G_a$ is obtained by removing $u$ from $G_b$, removing $\bar{u}$ from $G_c$, and taking the union of the two remaining graphs. Thus $G_b - u$ is a subgraph of $G_a$. Now let $U \subseteq V(G_a)$ be a set such that $|U| = \kappa_i(a)$ and $G_a - U$ has no $i$-clique. Since $G_b - u$ is a subgraph of $G_a$, we conclude that $G_b - (U \cup \{u\})$ contains no $i$-clique either, thus $\kappa_i(b) \leq |U \cup \{u\}| = \kappa_i(a) + 1$. 

If $a$ is an ancestor of $b$ in $T$, let $\text{dist}(a, b)$ denote the number of edges in the $T$-path from $a$ to $b$. Applying Proposition 8.13 $\text{dist}(a, b)$ times, we obtain the following proposition:

**Proposition 8.14.** If $b$ is a descendant of $a$ in $T$, then $\kappa_i(b) \leq \kappa_i(a) + \text{dist}(a, b)$.
We will now show that if the values $\kappa_i(a)$ are small for some node $a$ in the tree, then the subtree of $a$ is big, where the exact value of “big” depends in $i$ and $\kappa_i(a)$. Since $\kappa_i(\text{root}) = 0$, this implies that the whole tree is very big. We have to define what small and big actually means in this context. We fix a value $1 \leq \ell \leq k$ and define $\nu_i$ and $\theta_i$ for $1 \leq i \leq \ell$ as follows:

$$\theta_\ell := \left\lfloor \frac{k - \ell + 1}{2} \right\rfloor - 1$$
$$\nu_\ell := 1$$

$$\theta_i := \left\lfloor \frac{2^{\nu_{i+1}}\theta_{i+1} - 2}{\theta_{i+1}} \right\rfloor - 1, \quad 1 \leq i < \ell$$

$$\nu_i := \frac{\nu_{i+1}\theta_{i+1} - 1}{\theta_i} \left\lfloor \frac{\theta_i}{\theta_{i+1}} \right\rfloor, \quad 1 \leq i < \ell.$$  

These expressions look more terrifying than they are. They are simply chosen in way that makes the induction go through. We claim that the values $\theta_{\ell-i}$ grow very fast in $i$. For example, choosing $\ell = k - 260$, one can prove by induction that $\theta_i \geq \text{tower}_3(\sqrt{k} - 260)$. This implies that $\theta_i \geq \text{tower}_2(k - c)$ for some constant $c$ depending on $\epsilon$, but not on $k$. The technical details are not very enlightening and are therefore omitted. The following theorem is a more precise version of Theorem 8.6.

**Theorem 8.15.** Let $F$ be a strictly treelike weakly linear $k$-CNF formula. Then $F$ has at least $2^{\nu_1 \theta_1}$ clauses.

**Proof.** A node $a$ in $T$ is $i$-extendable if $\kappa_j(a) \leq \theta_j$ for each $i \leq j \leq \ell$. By definition, if $a$ is $i$-extendable, it is also $(i+1)$-extendable. For $i = \ell + 1$, the condition is void, so every node is $(\ell + 1)$-extendable. Also, the root is 1-extendable, since $\kappa_1(\text{root}) = 0$.

**Definition 8.16.**

(i) A set $A$ of nodes in $T$ is an antichain if no vertex in $A$ is an ancestor of any other vertex in $A$.

(ii) If $A$ is an antichain in $T$, $a$ is a node in $T$, and every node in $A$ is a descendant of $a$, then $A$ is an antichain of $a$.

(iii) If furthermore $\text{dist}(a, b) \geq d$ for every $b \in A$, then $A$ is an antichain of $a$ at distance at most $d$.

(iv) If $A$ is an antichain in $T$ and every $b \in A$ is $i$-extendable, then $A$ is an $i$-extendable antichain.

**Lemma 8.17.** Let $1 \leq i \leq \ell$, and let $a$ be a node in $T$. If $a$ is $i$-extendable, then there is an $(i+1)$-extendable antichain $A$ of $a$ at distance at most $\theta_i$ such that $|A| = 2^{\nu_i \theta_i}$.  

136
8.4. Linear MU(1)-Formulas

Figure 8.3: Illustration of the claim in the proof of Lemma 8.17. If node $a$ is $i$-extendable, and $b$ is a close $(i + 1)$-extendable descendant of $a$, then $b$ itself has many close descendants $A$, at least half of which are $(i + 1)$-extendable themselves.

Proof. We use induction on $\ell - i$. For the base case $i = \ell$, we have $\kappa_\ell(a) \leq \theta_\ell$, as $a$ is $\ell$-extendable. Since each leaf $b$ of $T$ has $\kappa_\ell(b) = k - \ell + 1 \geq 2\theta_\ell + 2$, Proposition 8.14 tells us that every leaf in the subtree of $a$ has distance at least $\theta_\ell + 2$ from $a$. Hence there are $2^{\theta_\ell}$ different paths of length $\ell$ starting at $a$ and going to descendants of $a$. These descendants of $a$ form an antichain $A$ of $a$ at distance $\ell$. Since every node is $(\ell + 1)$-extendable by definition, the base case holds. For the step, let $a$ be $i$-extendable, for $1 \leq i < \ell$.

Claim: Let $b$ be a descendant of $a$ with $\text{dist}(a, b) \leq \theta_i$. If $b$ is $(i + 1)$-extendable, then there is an $(i + 1)$-extendable antichain $A$ of $b$ at distance at most $\theta_{i+1}$ of size $2^{\nu_{i+1}\theta_{i+1}-1}$.

Proof of the claim. By applying the induction hypothesis of the lemma to $b$, there is an $(i + 2)$-extendable antichain $A$ of $b$ at distance at most $\theta_{i+1}$ of size $2^{\nu_{i+1}\theta_{i+1}}$. Let $A_{\text{good}} := \{c \in A \mid \kappa_{i+1}(c) \leq \theta_{i+1}\}$. This is an $(i + 1)$-extendable antichain. If $A_{\text{good}}$ contains at least half of $A$, we are done. See Figure 3 for an illustration. Write $A_{\text{bad}} := A \setminus A_{\text{good}}$ and suppose for the sake of contradiction that $A_{\text{bad}} > 2^{\nu_{i+1}\theta_{i+1}-1}$. Consider any $c \in A_{\text{bad}}$. On the path from $c$ to $b$, in each step some literal gets removed (and others may be added). Let $P$ denote the set of the removed literals. Then $C_c \setminus \{P\} \subseteq C_b$, and $G_c - P$ is a subgraph of $G_b$. Node $c$ is not $(i + 1)$-extendable, thus $\kappa_{i+1}(c) \geq \theta_{i+1} + 1$. Since $|P| = \text{dist}(b, c) \leq \theta_{i+1}$, the graph $G_c - P$ contains at least one $(i + 1)$-clique, which is also contained in $G_b$. This holds for every $c \in A_{\text{bad}}$, and by weak linearity, $G_b$ contains at least $|A_{\text{bad}}|$ edge disjoint $(i + 1)$-cliques. Since $b$ is $(i + 1)$-extendable, we have $\kappa_{i+1}(b) \leq \theta_{i+1}$, thus there exists a set $U \subseteq V(G_b), |U| \leq \theta_{i+1}$ such that $G_b - U$ contains no $(i + 1)$-clique. Each of the
\[ |A_{\text{bad}}| \text{ edge-disjoint } (i + 1)\text{-cliques of } G_b \text{ contains some vertex of } U, \text{ thus some vertex } v \in U \text{ is contained in at least } \frac{|A_{\text{bad}}|}{|U|} \geq \frac{2^{\nu_i+1}\theta_{i+1}-1}{\theta_{i+1}} \geq 2\theta_i + 1 \text{ edge-disjoint } (i + 1)\text{-cliques. Two such cliques overlap in no vertex besides } v, \text{ hence } G_b \text{ contains at least } 2\theta_i + 1 \text{ vertex-disjoint } i\text{-cliques, thus } \kappa_i(b) \geq 2\theta_i + 1. \text{ By Proposition } 8.14, \kappa_i(a) \geq \kappa_i(b) - \text{dist}(a, b) \geq \theta_i + 1. \text{ This contradicts the assumption of Lemma } 8.17 \text{ that } a \text{ is } i\text{-extendable. We conclude that } |A_{\text{bad}}| \leq \frac{1}{2}|A|, \text{ which proves the claim.} \]

Let us continue with the proof of the lemma. If \( A \) is an \((i + 1)\)-extendable antichain of \( a \) at distance \( d \leq \theta_i \), then by the claim for each vertex \( b \in A \) there exists an \((i + 1)\)-extendable antichain of \( b \) at distance at most \( \theta_{i+1} \), of size \( 2^{\nu_i+1}\theta_{i+1}-1 \). Their union is an \((i + 1)\)-extendable antichain \( A' \) of \( a \) at distance at most \( d + \theta_{i+1} \), of size \( |A|2^{\nu_i+1}\theta_{i+1}-1 \). Hence we can “inflate” \( A \) to \( A' \), as long as \( d \leq \theta_i \). Starting with the \((i + 1)\)-extendable antichain \( \{a\} \) and inflate it \( \left\lfloor \frac{\theta_i}{\theta_{i+1}} \right\rfloor \) times, and obtain a final \((i + 1)\)-extendable antichain of \( a \) at distance at most \( \theta_i \) of size at least \( (2^{\nu_i+1}\theta_{i+1}-1)\left\lfloor \frac{\theta_i}{\theta_{i+1}} \right\rfloor = 2^{\nu_i}\theta_i \). \[ \square \]

Applying Lemma 8.17 to the root of \( T \), which is 1-extendable, we obtain an antichain \( A \) of size \( 2^{\nu_i}\theta_i \). Since \( T \) has at least \( |A| \) leaves, this proves the theorem. \[ \square \]
Bibliography


[KK] A. V. Kostochka and M. Kumbhat. Coloring uniform hypergraphs with few edges, manuscript.

[KR] A. V. Kostochka and V. Rödl. Constructions of sparse uniform hypergraphs with high chromatic number, manuscript.


[Rol03] Daniel Rolf. 3-sat in rtime(o(1.32793^{N})) - improving randomized local search by initializing strings of 3-clauses. Electronic Colloquium on Computational Complexity (ECCC), (054), 2003.


[Wel] E. Welzl. personal communication.


Curriculum Vitae

Dominik Scheder

born August 28, 1980 in Nürnberg, Germany

1990 - 1999 Labenwolf-Gymnasium Nürnberg, Germany
Degree: Abitur

1999 - 2003 Studies of Computer Science
Universität Erlangen-Nürnberg, Germany

2003 - 2005 Studies of Computer Science
University of Colorado at Boulder, USA
Degree: Master of Science

since October 2005 Ph.D. student at ETH Zürich, Switzerland