DISS. ETH NO. 20668

EXACT ALGORITHMS FOR CONSTRAINT SATISFACTION PROBLEMS

A dissertation submitted to
ETH ZÜRICH

for the degree of
DOCTOR OF SCIENCES

presented by

ROBIN ALEXANDER MOSER
MSc ETH
born 14.08.1983
citizen of Inkwil (BE), Switzerland

accepted on the recommendation of
Prof. Dr. Emo Welzl, examiner
Prof. Dr. Uwe Schöning, co-examiner
Prof. Dr. Gábor Tardos, co-examiner

2012

Abstract

The Boolean satisfiability problem (SAT) and its generalization to variables of higher arities – constraint satisfaction problems (CSP) – can arguably be called the most "natural" of all NP-complete problems. The present work is concerned with their algorithmic treatment. It consists of two parts.

The first part investigates CSPs for which satisfiability follows from the famous Lovász Local Lemma. Since its discovery in 1975 by Paul Erdős and László Lovász, it has been known that CSPs without dense spots of interdependent constraints always admit a satisfying assignment. However, an iterative procedure to discover such an assignment was not available. We refine earlier attempts at making the Local Lemma algorithmic and finally present a polynomial time algorithm able to make almost all known applications constructive.

In the second part, we leave behind the class of polynomial time tractable problems and instead investigate the randomized exponential time algorithm devised and analyzed by Uwe Schöning in 1999, which solves arbitrary clause satisfaction problems. Besides some new interesting perspectives on the algorithm, the main contribution of this part consist of a refinement of earlier approaches at derandomizing Schöning's algorithm. We present a deterministic variant which losslessly reaches the performances of the randomized original.

Zusammenfassung

Die Erfüllbarkeitsprobleme SAT und CSP dürfen mit Fug als die "natürlichsten" aller NP-vollständigen Probleme bezeichnet werden. Die vorliegende Arbeit befasst sich mit deren algorithmischen Behandlung. Sie besteht aus zwei Teilen.

Der erste Teil befasst sich mit Erfüllbarkeitsproblemen, deren Lösbarkeit aus dem bekannten Lovász Local Lemma folgt. Während seit dessen Entdeckung im Jahre 1975 durch Paul Erdős und László Lovász feststeht, dass Erfüllbarkeitsprobleme mit einer nirgends zu dichten Konzentration an Klauseln immer eine erfüllende Belegung zulassen, war ein algorithmisches Verfahren zur tatsächlichen Bestimmung dieser Lösung lange nicht bekannt. Wir verfeinern frühere Ansätze, das Local Lemma algorithmisch zu machen und präsentieren schliesslich einen Polynomialzeitalgorithmus, der für beinahe alle bisher bekannten Anwendungen des Local Lemma einen konstruktiven Beweis liefert.

Im zweiten Teil verlassen wir die Klasse der in polynomieller Zeit lösbaren Probleme und betrachten stattdessen den von Uwe Schöning im Jahre 1999 vorgeschlagenen und analysierten randomisierten Exponentialzeitalgorithmus für allgemeine Klauselerfüllungsprobleme. Als Hauptbeitrag nebst weiteren Aspekten verfeinern wir frühere Ansätze, diesen Algorithmus zu derandomisieren und präsentieren sodann die erste deterministische Variante, welche gegenüber dem Zufallsalgorithmus nicht an Effizienz einbüsst.

Acknowledgements

First and foremost, I thank my supervisor Emo Welzl for everything he has done for me during the past years. I entered research in general and the field of satisfiability in particular mainly because I was impressed with his passionate and inspired way of teaching and his deep insights. Emo was the one to directly introduce to me all the topics considered in this work. Later, being his student – a 'Gremo' – was an extraordinarily enriching experience. More than anyone else I have met, he understands to encourage and motivate people, to inspire free and creative thinking, to establish a culture of trust and of well-being, to always deal very promptly and professionally with any upcoming problems and issues and to create a working environment where people can thrive and which could not possibly be better in any respect imaginable.

Besides Emo, many other people have contributed to this thesis:

I am grateful to Gábor Tardos for being an excellent co-author, collaborator, a great host and guide during my stay in Vancouver and finally a co-referee of my thesis who has provided a large number of valuable corrections and suggestions greatly improving the quality of this work.

I thank Uwe Schöning for publishing his milestone paper [Sch99] which inspired a large part of this thesis and kept me busy for many years, for being a good host and showing me around the beautiful city of Ulm and finally for co-refereeing and helping to improve my thesis.

I am indebted to Dominik Scheder for being an insightful and motivated co-author, somebody to whose office I could always go if I had math questions exceeding my abilities, a great speaker in lectures and seminars (I still regret having missed your defense!), a fine fellow-traveller to many places inside and outside of Switzerland and for organising many legendary parties in his flat in Zurich.

My thanks go to Timon Hertli for being a co-author and collaborator, for most valuable help with organising various lectures, inventing exercises, advising students and doing corrections and for his extraordinary brilliance which did not only advance the field of SAT but also helped me with overcoming many a mathematical difficulty I would have otherwise struggled with and for attending philosophy lectures with me.

I am grateful to Yves Brise for his very valuable help with designing the book cover for this thesis, for his support during the final stages of publishing this work, for being a fine mate during our simultaneous quest for new jobs, for his and his family's hospitality at his place in Allschwil and for the awesome photographs he produces, some of which now decorate our living room.

I thank Thomas Holenstein for advising me on several issues in direct connection with this thesis and for always being available to answer all kinds of math and cs questions.

I moreover thank all current and former Gremos who made my years inside and outside of the CAB building a most enjoyable time. Without your presence, working on this thesis would not have been half the fun. In particular I thank

Robert Berke for being my teacher when I was a student of Emo's SAT class and for helping me with some of my very early work as a PhD student;

Tobias Christ for many inspiring discussions about jobs and career,

money, politics, philosophy and life and in advance for the guided tour around Basel which he promised me;

Andrea Francke for being so much fun to share an office with, for collaborations in teaching, for organising events at the opera and in restaurants in Paris, for inviting me to her costume parties and for attending mine;

Bernd Gärnter for teaching me about computational geometry and for inspiring discussions at GWOPs and after-GWOPs;

Heidi Gebauer for fruitful collaborations on SAT publications and GWOP projects, for organizing our teaching duties, for reading parts of my thesis and for helping me with setting up my research plan;

Anna Gundert for proofreading parts of the thesis, being a very fine officemate, for many an inspiring discussion whether or not related to mathematics, for GWOP collaborations, for taking Dutch classes with me, and for reminding me of my Foamino duties as well as of my responsibilities to our precious environment;

Michael Hoffmann for proofreading, playing Robo-Rally and dozens of other exciting board games with us at which he always won, for buying the Foamino, for running our servers and for solving all the computer problems which are far beyond the average computer science PhD;

Martin Jaggi for checking parts of the thesis, for being a great head TA in APC, for his fun and relaxed attitude towards basically every aspect of both work and life, for inviting me to awesome barbecue parties in his place in Zurich and for joining us for luxury dining in Paris;

Vincent Kusters for reading and checking parts of my thesis, for being a reliable member of the APC team, for being a fellow campaigner for longer lunches and coffee breaks, and for the many hours we spent together enjoying travel, food and drinks, squash and board games;

Gabriel Nivasch for our collaborations on combinatorial problems and for the fruitful discussions about satisfiability;

Andreas Razen for being an awesome MiSe organizer, a reliable attendee of SAT seminars and student talks, the only certified top reviewer in the Gremo group, for many pleasant dinners and nights out and for inspiring discussions on life and career planning;

Andrea Salow for being a great secretary making all our lives so much easier, helping with copying hundreds of pages of student papers, for the delicious MiLuDi meals, the introduction of the fruit bowl and for being forgiving when her office smelled of burnt milk because of negligence on my part;

Eva Schuberth for leaving me some nice posters when cleaning up her desk;

Sebastian Stich for proofreading parts of my thesis, test solving student assignments, for advising me on optimization problems, for maintaining our group webpage and helping with the setup of the automatic MiSe system;

Marek Sulovský for being a great MiSe organizer, for attending probability classes with me, for introducing me to squash and to the local wine fair, for organising exciting travels to Paris and Prague and for advising me on my job search and career choices;

Tibor Szabó for being an awesome teacher in graph theory, thereby contributing to my interest in theoretical cs, for inviting us to dinner at his place in Bonstetten (even though he forced us to run for it beforehand), for his hospitality in Montreal and for many pleasant discussions at GWOPs and seminars;

Patrick Traxler for his hospitality during my stay in Vienna, GWOP collaborations, for many interesting discussions about both research and life and for organizing awesome Austrian comedy events;

Hemant Tyagi for answering my machine learning questions, test solving student problems, for interesting discussions over lunch, coffee and dinner and for going out with me in Zurich;

Uli Wagner for organizing our reading seminars and introducing me to many interesting topics I would otherwise not have come across and for invitations to parties in his flat featuring delicious food and legendary dancing performances;

and Philipp Zumstein for being a very pleasant office-mate, for helping me during my very first days in the GREMO group, for proofreading some of my work and for invitations to his exciting parties.

Moreover, I wish to thank Eyal Lubetzky for being my mentor during my stay at Microsoft Research in Redmond, and all the members of the Theory Group I had the pleasure to meet, work and go out with and I also thank Joel Spencer for being an enabler of my stay in Washington.

I thank the Master's and Bachelor's students whose theses I had the honor to be an advisor for, Andrei Giurgiu, Stefan Schneider, May Szedlák and Sebastian Millius, for their insightful and brilliant contributions to the field of satisfiability, some of which have found a place in this thesis too, as well as all other studens whom I had the pleasure to work with and whose ideas and challenging questions in class contributed to my research.

I thank all the members of academic institutions around the world who have organized the great theory conferences and workshops I had the pleasure to attend.

Last but not least, there is no way I can sufficiently express my gratitude to my parents, my family, my flatmates and all my friends who have made me who I am and are making my daily life worthwile. Thanks to all of you!

Preface

I suspect that PhD theses, and specifically those from the domain I am concerned with, are among the least read books there are. I am grateful to have a supervisor and co-referees who are reading mine and checking its content, and I am equally indebted to all the numerous friends and colleagues of mine upon whom I exercised a non-negligible amount of pressure so that they would read it as well and verify my arguments and calculations. And if on top of that, in the future, once every one or two academic terms, there is here or there around the globe some student preparing a thesis of his or her own or a contribution to a seminar involving any of the topics I set out to discuss and he or she stumbles accross this write-up and can make any use whatsoever of it, then it has undoubtedly fulfilled its purpose.

But I will be left not only with such hopes but also with an overly large pile of hardcopies. And since, as I have understood, it is a century-old tradition to distribute these amongst all of the author's family and friends, I *will* distribute them amongst my family and friends and I am dearly looking forward to their puzzled faces and, depending on how much they remember of what I told them the work I am doing was about, any or all of the questions "*What's this about?*" and "*You do not seriously expect me to read this?*". This preface is devoted particularly to them and their justified questions.

Let me answer the latter one first. I definitely do not and you have my explicit and irrevocable permission to hide this thing in the darkest corner of your home before you even reach the first page labelled in Arabic numerals. But before that, I ask you to do me the honor of

reading the following short paragraphs in which I did my very best to give a hint accessible to everyone at what kind of questions are being studied in the remainder of this work.

For once, let us suppose that we are standing in front of a small library featuring roughly four hundred books which I tidily shelved, ordered according to their authors' last names. And I ask you to find me a novel of *Dickens*. This is an *easy* task. You check out where the 'D' starts, then you look for the 'i', you see the book, grab it and hand it over to me. You are done in no time. As computer scientists, we would say this is a task of *logarithmic complexity* – it basically means that the task is so easy it will take you far less time than it took me to put the books on the shelf, because although there are many books by *Jane Austen* which *I* had to place there, *you* can skip them as you know the one you are looking for does not start with an 'A'.

Now suppose I ask you to find me a book of which the text starts with the word 'it'. And let us make the improbable assumption that you are not sufficiently literate to know by heart that *A Tale of Two Cities* does. Then your only choice will be to grab each and every of the books, open it where the story starts and read the first word. If you are unlucky – and according to Murphy's law that is what you *will* be – you will have to look at all the books until you find the single one matching my request. This task is *tedious*, it is a nuisance, but it is *straightforward* as well, because there will be no surprises: you know exactly what you have to do to achieve what I am asking of you. The computer scientist would say this is a task of *linear complexity*, which means that it will take you roughly as long as it took me to put the books on the shelf.

Finally, let us suppose that I ask you to make a selection of several books in such a way that the total weight of the books you chose is exactly 20 kilograms. Now we have a *real* problem on our hands. You can start picking some of the novels at will and put them on your scale. You see that you have 18 kilograms, so you add another thick

one, then you have 20.5 kilograms, you remove a thin one and are left with 19.8 kilograms. You would like to add one that weighs an exact 0.2 kilograms but you find no such book in the library. You keep trying and at some point you give up. The thing is: if there are four hundred books on this shelf, then the number of possible choices you could make is far larger than the number of atoms there are in the known universe. And if you wanted to try each one, you would be busy – well, basically forever.

Luckily, nowadays we have *computers* which can do quite a lot of work for us and they can do it really quick: a cheap modern machine from the retail store can easily carry out one billion elementary operations per second. This is particularly helpful with the *tedious* tasks. Instead of opening every book in the library and checking its first word, a computer on which the books are stored electronically can search for the word 'it' at the beginning of all novels and output *A Tale of Two Cities* in a fraction of a second. Even if there were not four hundred books but all the twenty million which a respectable library like the *Library of Congress* in Washington D.C. features, still it would be a matter of seconds to list all of which 'it' is the first word. The reason why this is not a problem and why the simple solution will scale to libraries of any size is because the problem has *linear complexity*.

But what about filling a box with 20 kilograms of books? We can weigh each book separately and store its weight in the computer. But if the computer is to check all ways how they may possibly be combined, none of today's most powerful computers, not even all of them jointly, have the slightest hope of ever finishing the task. And arguably, if the number of atoms in the known universe is smaller than the number of possibilities we have to take into account, no computer mankind could ever hope to build will. And even worse than that, if there *were* an amazing machine which could examine all possibilities of selections from 400 books in, say, a day, then that machine would be totally useless again for another library counting 420 books because with *every*

single book we add, the number of possibilities to be checked *doubles*.

Although it may seem this is where computer science *ends*, in fact *this* is where computer science *starts*. The question that we should have asked was not whether we can build a bigger and faster computer to solve the problem, but whether there is a more *clever way* of solving it. Nobody told us that we *had to* try all the combinations of books blindly and maybe there is a cunning and highly complex strategy or calculation that will bring forward a solution more quickly. The truth is, in this particular case and on the day I am writing this preface, nobody knows. This is both disappointing and thrilling at the same time, as you can here and now start doing research. Develop a strategy which is *guaranteed* to work for any collection of books and which finds such a selection with reasonable effort. I promise that if you find one, you will be famous, appear in the newspapers and be hired by your favorite university at conditions *you* determine.

All problems I have mentioned so far have one important thing in common. Once you have *found* a solution, it is most easy to *verify* that this solution is right. You hand me the novel of Dickens, I read the cover and am satisfied. You give me any novel starting with the word 'it', I open it on the first page, check and am satisfied. You give me any choice of books that weigh an exact 20 kilograms, I put them on my scale and am satisfied. Not *all* the problems have this property, but there are infinitely many more riddles which do.

A standard example are the *Sudoku* problems you recently find in many newspapers and magazines. *Finding* the solution of a Sudoku may at times be extremely hard, but if I *show* you the solution of one, you can most easily *check* that there are no misplaced numbers.

One of the great advances of theoretical computer science was made by Cook and Levin [Coo71, Lev73] who discovered that all such problems are basically the same, identical problem. It may sound hilarious to you but it is in fact true: if I am facing a library and I am supposed to assemble a choice of books weighing an exact amount of

kilograms, I do not know of any way to solve this problem within my lifetime. But I *do* know a way, although it is fairly complicated, of writing down the weights of all the books, then doing some calculations and from those to produce a huge Sudoku problem. A Sudoku problem which *I* do not know how to solve, but if *you* are a genius and find a solution to my Sudoku problem, then from this solution, from the numbers you filled into your squares, I can do another calculation and then come up with a selection of books having the prescribed weight. Both the production of the Sudoku and the reconstruction of the solution are lengthy, *tedious* processes. But they are *straightfoward*, I can follow a determined recipe, much like with the search problem. If I even have a computer at my disposition, I am done with this in no time. The same is true for millions of comparable problems.

So if you find a viable way of solving large Sudoku problems, not only did you solve Sudoku, you solved the book selection problem as well and in fact you solved arguably *all reasonable* difficult problems there are. Since many intelligent people have tried, and tried hard, most researchers have given up hope that there might be a simple answer to these questions and are in fact currently trying to do the opposite: to find a *proof* for the fact that no good solution exists. Another thing we do is to identify characteristics which distinguish simple from hard problems. Take the bookshelf example: if every book weighed exactly one kilogram, getting together any exact number of kilograms would be easy because we could take any choice of n books to get n kilograms. It's a trivial example but it generalizes. You can ask: *what exactly* is it that makes this situation so much different?

In this thesis there are three chapters. The first is an introduction where the exact problem we discuss, the problem of *constraint satisfaction* (CSP), is formally described. A CSP is a more general concept of riddles very closely resemblant of the Sudoku problem. It is a combination of *variables*, in the Sudoku case the squares where you can *vary* the numbers you want to fill in, and *constraints* which are the

rules of the game, like 'no two identical numbers in a row', and so forth. Recall that since all the problems are essentially equivalent, it does not overly matter *which* one we work on. CSPs are a natural choice because very many problems can be translated to CSPs easily and directly with much less effort than it takes to find the Sudoku corresponding to the bookshelf.

In the second chapter, we look at a type of CSPs which turn out to be solvable easily using a simple strategy. If the constraints do not *contradict* each other too heavily, then what you can do is use a dice to fill in random numbers and then check whether there are violated constraints and if there is one, you erase the numbers violating the constraint and use your dice again to fill in new numbers. It is by no means clear that this strategy will produce something in any given amount of time, but we give a proof here that under certain favorable conditions, it does.

The third chapter is devoted mainly to the very question of dices. Do dices help with solving problems? This is another huge unsolved question. Most researchers think they do not, in the sense that if there is an efficient strategy to resolve a riddle and the strategy involves the use of dices, then there should also be a comparably efficient strategy which can do without. People are trying to find a proof for this conjecture – and so far struggle. In the third chapter we look at one instance of this question, an algorithm by Uwe Schöning for solving certain types of CSPs which involves the use of dices. We demonstrate that there is a simple alternative way of doing the same thing with basically the same amount of effort and without a dice.

Both are tiny contributions which in the end may or may not turn out to serve as little pieces of a giant puzzle. A puzzle which, if nothing else, might help to make feel slightly more humbled a mankind who has flown to the moon and is constructing nanoscopic robots while at the same time unable to resolve a simple brainteaser.

Zurich, August 2012, Robin Moser

Contents

1 Introduction

In a *constraint satisfaction problem* (CSP), we are confronted with a series of *choices* and some *constraints* telling us which combinations of choices are undesirable. The goal is to determine if all constraints can be met.

As so many hard computational problems map to CSPs transparently, they arguably form *the* prototype of a hard combinatorial problem and any insights into their properties transfer seamlessly to a vast variety of domains.

In this chapter, we set out to motivate our investigation of particular aspects of CSPs, fix a formal framework and mention some closely related topics which are beyond the scope of this thesis.

1.1 Motivation

In the early seventies, Stephen Cook [Coo71] as well as Leonid Levin [Lev73] independently discovered the concept of *NP-complete* problems. A computational problem is *NP-complete* if it has two characteristic traits. For one, the problem must be in *NP*. Intuitively, NP is the class of all problems a given solution to which is efficiently verifiable. Secondly, it must be *complete* in the sense that *all* instances of problems in NP can be efficiently translated to an instance of the complete problem. This means that NP-complete problems are – in a well-defined sense – all equivalent. And – in a not so well-defined sense – they all seem to be hard as they have so far resisted all attempts at efficient resolution.

With the concept of NP-completeness, Cook and Levin delivered a proof that the *Boolean satisfiability problem* falls into this class. In a SAT problem, you are given a formula of propositional logic, that is a formula built from *Boolean variables* taking the values 'true' or 'false' and the logical operations conjunction, disjunction and negation and you are to determine whether there is an assignment of values to the variables such that the whole formula evaluates to 'true'. *Constraint satisfaction problems* or CSPs are the immediate generalization of SAT where the variables can take more than two values.

Proving SAT as the first ever problem to be NP-complete involved an elaborate argument over all possible algorithms in the computational model of *Turing machines*. Once established for one problem, it is being demonstrated for other problems via reductions. By today, a sheer unlimited supply of computational problems has been proven to be NP-complete via a reduction either directly or ultimately from SAT.

The reason why SAT (or CSP) has such an outstanding role is far more than historical. Pick any example of an NP-complete problem and chances are that its reduction to SAT (or CSP) is far more imme-

diate than the converse. In GRAPH 3-COLORING, we are supposed to color the vertices of a graph with three colors such that no two adjacent vertices receive the same color. The reduction to a CSP is straightforward: instantiate one variable per vertex and constraints for the edges. The converse reduction involves sophisticated graph gadgets representing variables and clauses. In BINARY SUBSET SUM, we want to determine whether among a set of integers, there exists a subset summing up to a prescribed number. The reduction to SAT is tedious but straightforward: it is necessary to model the addition of numbers by a standard adder circuit. The converse however, representing a SAT formula by means of numbers and integer addition, is highly non-trivial. Despite being arbitrary examples, these are representative of most NP-complete problems.

SAT, and more generally CSP, are hence arguably the most 'natural' NP-complete problems there are and any insight into them is highly likely to translate to many other domains.

Before we outline which paths research has taken in the field of CSPs, let us agree on a formal problem statement.

1.2 Formal Problem Setting and Notation

The specific problem that we will consider in this thesis is quickly described. In its description we build closely upon the formal framework which Welzl uses in [Wel12].

Clause Satisfaction Problems. Suppose that we are given a finite set

$$V_n := \{x_1, x_2, \ldots, x_n\}$$

of *n variables* and for each variable $x \in V_n$ a finite *alphabet* L_x of *values* the variable can take. An *assignment* is a total function

$$\alpha : V_n \to L := \bigcup_{x \in V} L_x$$

that sends each variable x to a specific value $\alpha(x) \in L_x$. The *solution space* in this setting is the set

$$\mathcal{S} := \{\alpha : V_n \to L \mid \alpha(x) \in L_x\}$$

of all possible assignments, where we use the common notation $\{X \to Y\}$ to denote the set of all functions from set X to set Y. This space has the shape of a high-dimensional cube or grid and there is a natural metric on it induced by the *Hamming distance* of two assignments α, β defined as

$$\text{dist}(\alpha, \beta) := |\{x \in V_n \mid \alpha(x) \neq \beta(x)\}|.$$

Now suppose that from the solution space we want to exclude certain assignments which exhibit undesired combinations of values. We do this by prescribing a list $F := \{C_1, C_2, \ldots, C_m\}$ of m clauses, where a *clause* is a finite set

$$C_i = \{(x_{i_1} \neq v_1), (x_{i_2} \neq v_2), \ldots, (x_{i_k} \neq v_k)\}$$

of variable-value pairs $(x_{i_j} \neq v_j)$ called *literals*, where $v_j \in L_{x_j}$ for $1 \leq j \leq k$ and the variables are pairwise distinct, i.e. $i_j \neq i_{j'}$ for $1 \leq j < j' \leq k$. k is said to be the *size* or the *length* of the clause and we use the standard notation $|C_i|$ for it. A clause containing only one literal is called a *unit clause*, if there is no literal in it it is the *empty clause*.

The semantics associated with a clause are that we do *not* want to simultaneously assign x_{i_j} the value v_j for all $j = 1 \ldots k$. An assignment $\alpha \in \mathcal{S}$ is said to *violate* clause C_i if $\forall j : \alpha(x_{i_j}) = v_j$ and to *satisfy* C_i, otherwise. Note that the empty clause is always violated according to this definition. We call a finite set F of clauses a *clause satisfaction problem*, abbreviated *ClSP*, or, interchangeably, a *formula* or *ClSP formula*. α is said to *satisfy* the ClSP F if it satisfies all clauses.

The question that we are interested in is always going to be: given a formula F, *is there* an assignment $\alpha \in \mathcal{S}$ satisfying F and if so, *how*

can we find one algorithmically. If there exists such an assignment, F is said to be *satisfiable* and otherwise it is *unsatisfiable*.

Note that for an instance of this computational problem to be well-defined, its description must be a triple $\langle V, \{L_x\}_{x \in V}, F \rangle$ describing the variables, the admissible values and the set of clauses. Throughout this thesis, by a slight abuse of formal notation, we will simply write "a ClSP F" to either mean just the set of clauses or the whole instance description depending on the context. If F is the input to an algorithm, it is understood that a complete instance description must be input. On the other hand, we will write $C \in F$ to mean a clause contained in the formula. We also stress that we use the term "a ClSP" to refer to an instance of the computational problem, not to a class or to a language.

The *size* of a ClSP, in writing $\text{size}(F)$, is defined to be the number of literal occurrences, i.e.

$$\text{size}(F) := \sum_{C \in F} |C|.$$

In all reasonable cases, the binary representation of a formula is linear or quasilinear in its size. There may be degenerate cases where this is not the case, for instance if the lists are enourmous in size or if there is a disproportionally large set of variables which are unused. However, it is obvious that such formulas can be simplified by trivial operations (remove variables which do not occur, remove values from lists if the values are not used in any literal over the corresponding variable) until size and encoding length match in order of magnitude. The size of a formula is relevant in cases where we are interested in polynomial-time algorithms operating on it.

If all clauses in the given formula feature exactly the same number k of literals and each variable can take the same number $|L_x| = d$ of values, then we call the formula a (d, k)-ClSP. Without loss of generality, we may simply use $L_x = [d] := \{1..d\}$ for all lists. In this case, the solution space is $\mathcal{S}_{n,d} := \{V_n \to \{1..d\}\}$. We extend this nomenclature

naturally to $(\leq d, \leq k)$-ClSP, $(\leq d, k)$-ClSP, $(d, \geq k)$-ClSP and all further combinations to mean formulas where all variables have lists of size at most (or at least) d and/or all clauses have size at most (or at least) k.

In the special case of $d = 2$, thus where every variable can take exactly two values, one calls the problem a *Boolean satisfiability Problem*, or *SAT Problem* for short and F is then called a *formula in Conjunctive Normal Form* or *CNF formula* for short. A k-CNF is a $(2, k)-$CSP and $(\leq k)$-CNF and $(\geq k)$-CNF are defined analogously. Since it is customary in the case of Boolean formulas to consider the underlying alphabet to be $\{0, 1\}$ (more closely resembling the logical meanings 'false' and 'true' and their encoding in binary), we will follow the convention of using $\{0, 1\}$ whenever we are speaking about CNF formulas exclusively, and $\{1..d\}$ whenever we are speaking about the more general ClSP case.

The relevant literature is not consistent with respect to this notation and often, one can find the term "k-CNF" denoting either what we here call a k-CNF or a $(\leq k)$-CNF or a $(\geq k)$-CNF. In this thesis, a k-CNF has clauses of size exactly k exclusively. Note however that in many cases, a result or argument generalizes trivially to include larger or smaller clauses. Which of the two is the case depends on the context. In Chapter 2, we examine families of formulas which are always satisfiable by virtue of meeting certain combinatorial criteria. Having a satisfiable formula at hand, making the clauses larger cannot hurt as this renders the formula even more easily satisfiable. Neither can it hurt the effectivity of any algorithm tailored for such a class as we could simply truncate clauses before feeding the problem into it. The same holds for enlarging variable's domains. In this context, therefore, statements, even if formulated for (d, k)-ClSP, usually trivially extend to the $(\geq d, \geq k)$-ClSP case. In Chapter 3, we consider exponential time algorithms for solving arbitrary (d, k)-ClSP. Such algorithms, usually being improvements over exhaustive enumeration

of possibilities, profit from variables having smaller domains because fewer assignments need to be enumerated and from smaller clauses because fewer possibilities for satisfying a clause have to be branched on. And in particular, if some algorithm relied on the homogeneity of clause and alphabet sizes, we could simply enlarge alphabets and pad clauses with auxiliary variables, adding some extra clauses so as to preserve the set of satisfying assignments. In this context, therefore, most results on (d, k)-ClSP trivially extend to the $(\leq d, \leq k)$-ClSP case. Of course, these are only intuitive explanations, not proofs, so the issue has to be borne in mind when going through the actual material. At times, we will however deliberately leave this burden to the reader as assuming the (d, k)-ClSP case incurs no loss of generality whilst inhomogeneous clause and alphabet sizes make notation and arguments unnecessarily clumsy.

A *partial assignment* as the natural relaxation of an assignment is a partial function

$$\alpha : V_n \rightharpoonup L$$

such that $\alpha(x) \in L_x$ for all $x \in V_n$ on which α is defined. Note that we use the standard notation $\rightharpoonup$ to denote partial functions.

ClSP formulas can be manipulated by assigning values to variables. If F is a formula and x some variable, then assigning value $c \in L_x$ to variable x has the natural effect that all clauses containing the literal $x \neq c'$ for some $c' \neq c$ are satisfied and disappear from the formula. On the other hand, the literal $x \neq c$ cannot be satisfied anymore and thus all clauses containing it are truncated to contain only the remaining literals. We write $F^{[x \mapsto c]}$ to denote the formula so derived.

In general, for any partial assignment α, we write $F^{[\alpha]}$ to denote the formula arising from substituting all given values for the variables assigned in α. If u is a literal over a Boolean variable x, i.e. $L_x = \{0, 1\}$, then writing $F^{[u \mapsto 1]}$ is customary to say that we assign

$x \mapsto 0$ if $u = (x \neq 1)$ or $x \mapsto 1$ in case $u = (x \neq 0)$, i.e. writing $u \mapsto 1$ means to assign the unique value satisfying u to its underlying variable. Writing $u \mapsto 0$ means to assign the underlying variable the value which violates the literal. Note that $u \mapsto 0$ makes sense even if u is a literal over a variable of an arity larger than two. On the other hand, $u \mapsto 1$ is undefined in this case. We extend this notation to assignments as follows. If α is any (total or partial) assignment, then we write $\alpha[x \mapsto c]$ to denote the assignment arising from (re-)assigning variable x to value c. If u is a Boolean literal, $\alpha[u \mapsto 0]$ and $\alpha[u \mapsto 1]$ are defined accordingly. More generally, if β is any (total or partial) assignment, then $\alpha[\beta]$ denotes the assignment α with all assignments in β added to it, overwriting existing ones.

Let us call a ClSP *non-degenerate* if it does not contain any empty clauses, unit clauses, no variables with a singleton domain, no unused variables and no unused values in any lists. Note again that all degeneracies can be trivially removed from any ClSP: if an empty constraint is discovered, the ClSP is trivially unsatisfiable. The same is true if some variable's domain is empty. If a unit constraint is discovered, we may remove the corresponding value from the variable's domain (resulting in the disappearance of the unit clause and possibly other clauses containing the same literal). If a variable has a singleton domain, we can substitute the corresponding value for the variable (resulting in the truncation of some clauses). Unusued variables can be trivially deleted. If any variable has an unused value in its domain, we can substitute this value for the variable, satisfying all clauses where the variable occurs. Once none of these rules apply anymore, the result of this polynomial time simplification process will either be a ClSP which is non-degenerate or the discovery that the ClSP is unsatisfiable. We stress that the empty formula is non-degenerate.

Constraint Satisfaction Problems. *Constraint satisfaction problems*, abbreviated by *CSP*, are a generalization of clause satisfaction prob-

lems. In a CSP, clauses are replaced by constraints which can be arbitrary predicates over the variables involved. That is formally, we can define a constraint satisfaction problem as a triple $\langle V, \{L_x\}_{x \in V}, F \rangle$, where F is a set of constraints, each *constraint* being a pair $\langle U, P \rangle$ where $U \subseteq V$ and P is a set

$$P \subseteq \{\ \beta : U \to L \mid \forall x \in U : \beta(x) \in L_x\ \}$$

describing the combinations of values for the variables in U which are admissible. Such a CSP is *satisfied* by an assignment $\alpha \in \mathcal{S}$ if for all constraints $\langle U, P \rangle$, the projection $\alpha|_U$ is admitted by the constraint, i.e. $\alpha|_U \in P$. Here, we use the common notation $f|_A$ to denote the restriction of a function f to a subset A of its domain.

A clause satisfaction problem is the special case of a constraint satisfaction problem where each constraint is a clause, that is each constraint $\langle U, P \rangle$ has a relation P containing all except exactly one combination of values. While every CSP can be converted into an equivalent ClSP by *constraint splitting*, the crucial difference is a matter of description. To make a general CSP a well-defined computational problem, we have to prescribe a way in which the constraints are being represented in an input encoding. One choice would be to represent the predicates P in the constraints by listing all forbidden tuples. Then, the CSP *is* a ClSP.

Another choice would be to list all allowed tuples, corresponding to a formula in so-called *Disjunctive Normal Form (DNF)* in the Boolean case. This is already a crucial difference. A ClSP containing large clauses may explode exponentially in size when converting between the two representations.

A more natural choice may be to give the constraints as *circuits* or as efficient algorithms which, on input a combination of values for the variables in U, determine whether the combination is admissible or not. This allows for compact representations of a large variety of constraints but may shift some of the difficulty of solving the CSP towards

solving the constraints themselves. In general, the choice of representation may be crucial when determining the efficiency of an algorithm as a function of the description size of the formula, at least in cases where constraint grow with the problem size.

It may also be crucial in other contexts. In Chapter 2 of this thesis, we will look at a class of formulas which can be solved efficiently, where the class is characterized by relations between metrics like the number of constraints, the number of dependencies among constraints and the number of value tuples a constraint forbids. While these metrics do not depend on the input encoding, they depend heavily on whether constraints are clauses or more general predicates. In each case, we will first concentrate on ClSPs and then discuss issues that may come up with the generalization to CSPs and even more general settings.

Let us stress that in this thesis, we will consider "a CSP" to be a problem instance, the same way we did with ClSPs. The literature is not consistent in this respect. Sometimes, the term is used to describe the corresponding class of instances, oftentimes it is even used to describe a family of such classes, which turns into a computational problem only once a specific set of admissible constraints is prescribed. For a prominent case, Feder and Vardi [FV98] use such terminology to formulate their well-known *Dichotomy Conjecture*, stating that whenever a finite set of admissible predicates is prescribed, then the corresponding computational problem is always either polynomial-time solvable or NP-hard.

In this thesis, we are however not concerned with predicate types beyond the distinction between clauses and general efficiently computable predicates. We usually formulate our investigations in terms of ClSPs but where necessary comment on how to generalize concepts to more general CSPs.

1.3 Topics Discussed

Being so fundamental, SAT and CSPs have become famous, widely studied problems and research into them has branched into a vast number of directions around computational and non-computational aspects.

In this thesis, we focus on algorithmic issues. When investigating into the algorithmic treatment of SAT and CSP – and with a less ambitious goal than demonstrating P=NP in mind – there are two main perspectives one can take as we will do in this thesis.

Classifying instances. The first thing one can do is to identify subclasses of the full problem which admit efficient solutions so as to separate 'easy' from 'hard' instances with the aim of gaining more insight into what makes a problem 'easy' or 'hard'.

As one prominent example, 2-SAT, the Boolean problem where every clause contains exactly (or at most) two literals, is well-known to be solvable in time linear in the size of the formula [APT79].

Other examples include *Horn-SAT* or formulas of *bounded deficiency* [DG84, Sze04].

In Chapter 2 of this thesis, we will study one particular other such class: the family of CSP problems for which the famous Lovász Local Lemma guarantees the existence of a satisfying assignment.

Unlike the aforementioned restrictions, this is a set of formulas for which a simple combinatorial criterion ensures *existence* of a solution and instead of considering the decision problem, the main issue will lie in *finding* the actual assignment efficiently. While in the case of most other polynomially tractable subclasses of SAT there is a simple reduction from the search to the decision problem, this is not true for Local Lemma instances.

We first review all combinatorial aspects of the Local Lemma which we consider relevant to the topic as well as previous attempts at making the lemma algorithmic. We then go on to exhibit an algorithmic version which is able to capture the full power of the Lovász Local Lemma and seems to translate easily even to problems beyond SAT and ClSP.

Exponential Time Algorithms. The other perspective one can take is that algorithms able to solve *all* instances of SAT and ClSP/CSP and not just *some* of them are of great interest. And even if it is widely believed not to be possible to find one which runs in polynomial time, we can still try to determine how fast an algorithm is possible.

To our present knowledge, it is possible that no algorithm can solve k-SAT in general in less than exponential time, i.e. a running time of the form $\lambda^{n+o(n)}$. The hunt is currently for the smallest possible base λ.

To date, there are two algorithms which are considered 'competitive' in the domain of exponential time algorithms for SAT and ClSP: the one-pass satisfiability decoding algorithm named *PPSZ* after its inventors Paturi, Pudlák, Saks and Zane [PPSZ05] and the random local search algorithm by Uwe Schöning [Sch99]. With its most recent analysis due to Hertli [Her11], PPSZ holds the current record for k-SAT among *randomized* algorithms and a generalization to the (d,k)-ClSP case is being investigated [Sze11, Mil12].

On the other hand, Schöning's algorithm in its derandomized version is to date the fastest known *deterministic* k-SAT and (d,k)-ClSP algorithm. This derandomization is the main contribution of Chapter 3. Besides, we will discuss a series of further interesting aspects of this simple algorithm and its elegant analysis.

1.4 Topics Not Discussed

A discussion of any aspects of SAT and CSP would be distinctly incomplete without at least mentioning the following two important research fields which take perspectives on the problem other than the one we will take in the remainder of this work.

SAT Solving Software. Maybe the most popular of these branches is concerned with the actual *implementation* of software able to take an encoding of a SAT formula and to determine whether it admits a satisfying assignment.

Such software is of utmost importance to certain scientific and industrial domains. In the design of digital circuits, to name just one example, a model that is to serve as a blueprint for the production of actual hardware needs to be checked for logical faults. The sheer size and complexity of such designs make it necessary that the checking take place in a streamlined, automated way.

Inherently being objects of binary logic, it is not hard to imagine that the design of a circuit along with a specification of what would be its correct, intended behavior can be converted into a (huge) formula of propositional logic (a SAT formula). Most frequently, a formula encodes the internals of the circuit along with the description of a possible faulty state and SAT solving software is then invoked to check if there is a satisfying assignment to that formula, thus if there is any configuration of the input gates that can lead the chip to exhibit that particular unintended behavior.

Again not surprisingly, formulas that occur in such model checking applications are composed of tens of thousands of variables and hundreds of thousands of clauses. The current state of theory, with the fastest known algorithm (for the simplest variant, i.e. 3-SAT) having a running time of $\mathcal{O}(1.308^n)$ where n is the number of variables,

would predict such formulas to be straight-out intractable. However, modern SAT solvers are being applied and break such instances routinely on a daily basis.

This near paradoxical situation on one hand highlights how poorly understood SAT is *both* theoretically and practically and on the other hand how small a fraction of all possible constraint problems actually occur in practice. Researchers concerned with practical SAT solving software taylor their software to the instances they see, use their statistical properties and gather together a vast body of heuristics to speed up the search for satisfying assignments. Such heuristics usually do not make their way into theoretical research, as experience demonstrates that virtually every heuristic admits *some* cleverly crafted counterexample defeating its purpose, even if that particular counterexample would never be seen in a practical application. In theory however, we are interested in algorithms that perform well on *all* formulas or at least all formulas of a well-defined class.

Another reason for this considerable gap between theory and practice lies in the level of sophistication of modern SAT solvers which span thousands of lines of optimized code while in contrast, theoretical research struggles to prove tight running time bounds on even the simplest imaginable few-line algorithms. For complex strategies involving all sorts of optimizations and heuristics, both proving theoretical running time bounds and crafting counterexamples to demonstrate their limits at solving the general problem are so distinctly beyond the current state of research that the question of what *really* is the fastest SAT solving strategy is most likely to remain unanswered for decades to come.

In the present thesis, the issue of practical SAT solving will not further be discussed as we restrict our interest to algorithms which have provable bounds on their running times and which work on *all* formulas of a mathematically well-definable type.

Random SAT. Another approach at the problem is to consider algorithms that are efficient on *almost all* formulas where the notion of *almost all* is being made precise by imposing some distribution on the space of all formulas and then drawing one at random. The quest is then for algorithms that decide satisfiability efficiently with high probability over the input distribution.

This approach could arguably be regarded as just another restriction to some subclass of formulas which are computationally tractable. But the particular way in which this subclass is defined *does* make it special.

It is well-known that for any fixed d and k, if a (d,k)-ClSP over n variables is generated at random by uniformly picking a subset of size m of the roughly $(nd)^k$ possible clauses, then there is a constant $c_{d,k}$ such that the resultant formula is satisfiable w.h.p. if $m \ll c_{d,k}n$ and unsatisfiable w.h.p. if $m \gg c_{d,k}n$. And there is another constant $c'_{d,k} < c_{d,k}$ such that for $c'_{d,k}n \ll m \ll c_{d,k}n$, while being w.h.p. satisfiable, the satisfying assignments appear in tiny fragments scattered all around the solution space. Below density $c'_{d,k}n$, they tend to stick together in a large clump [ACO08].

At the corresponding *constraint densities* where these very changes occur, there are *phase transitions* which have been intensively studied and which resemble the behavior of certain physical systems. The latter has attracted the attention of statistical physicists who have made numerous contributions to the field over the past years.

Within the satisfiable regime, one is interested to have an algorithm that takes a random formula as input and delivers a satisfying assignment with high probability. To the author's knowledge, the current state of research is that algorithmic treatment is usally possible efficiently up to the *shattering transition* at density $c'_{d,k}$, one of the more recent advances by Coja-Oghlan [CO10].

Although very fascinating, in this thesis, we do not concern our-

selves with questions of random satisfiability but only with algorithms which have provable running time bounds in the *worst* case on all formulas or all formulas from a specific combinatorially characterized class.

2

The Lovász Local Lemma

As the problem of deciding whether a prescribed ClSP formula admits a satisfying assignment is widely believed to be computationally very hard in general, one branch of research concentrates on identifying special classes of formulas for which the problem becomes tractable.

In this chapter, we study one large class of formulas for which this is prominently the case, namely the class of formulas 'without dense spots' to which the famous *Lovász Local Lemma* (*LLL*) is applicable. Its statement is that if there are sufficiently few interdependencies between the clauses in a formula, then the formula is always satisfiable. *Deciding* satisfiability of such inputs is trivial. Additionally *searching* for a satisfying assignment is not and will be the main concern in this chapter.

2.1 Introduction[1]

Warmup. Before we go on to state the Local Lemma, as a warm-up, let us look at simpler classes of formulas for which the same conclusion holds. For instance, formulas with only very few clauses can be seen to be satisfiable very easily.

Theorem 2.1. *A (d,k)-ClSP formula F with $|F| < d^k$ clauses is satisfiable.*

A proof for this theorem is a simplest showcase example for the *probabilistic method* pioneered by Erdős which has since been used very widely in many areas of combinatorics.

Proof of Theorem 2.1. Let α be an assignment chosen u.a.r. from $\{V_n \to \{1..d\}\}$. Each clause $C \in F$ is violated with probability d^{-k}. The expected number of violated clauses in F is therefore strictly smaller than $d^k d^{-k} = 1$ and thus with non-zero probability, all clauses are satisfied. This implies that there exists a satisfying assignment. □

Of course, there is nothing in this proof inherently different from a simple counting argument that juxtaposes the total number of assignments in $\{V_n \to \{1..d\}\}$ with the number of assignments each separate clause can forbid. But as arguments get more complicated, thinking in terms of probabilities makes matters considerably more intuitive.

A somewhat more flexible version of this warm-up statement follows immediately from the same proof. Let F be a ClSP. For any $C \in F$, define the *weight* of a clause C as the probability that an assignment selected uniformly at random violates this clause, that is

$$w(C) := \prod_{x \in \mathrm{vbl}\, C} |L_x|^{-1}.$$

[1] Parts of the write-up in this section have appeared in [MW11].

Then, we obtain the following.

Theorem 2.2. *A ClSP formula F for which*

$$\sum_{C \in F} w(C) < 1$$

is satisfiable.

Theorems 2.1 and 2.2 can easily be seen to be tight. If we simply stack d^k clauses on top of each other, i.e. d^k distinct clauses over the same k variables, then those necessarily forbid all d^k possible assignments over these variables, rendering the formula unsatisfiable.

This example of tightness however leaves us with the feeling that there might be more to say. After all, concentrating all the clauses of a formula over the same few variables might not make for interesting examples and we might conjecture that if the same number of clauses were more loosely distributed, the resultant formula might again admit satisfying assignments. The other end of the scale, at the least, placing d^k independent clauses over disjoint sets of variables, yields a formula on kd^k variables with many, namely $(d^k - 1)^{d^k}$, satisfying assignments. But what is the threshold situation and what amount of interleaving is necessary to build an unsatisfiable formula?

The Symmetric Local Lemma. In 1975, Erdős and Lovász [EL75] gave an answer to this question. They argued[2] that building an unsatisfiable (d, k)-ClSP requires not only a *total* of at least d^k clauses but it requires roughly this number of clauses to be concentrated in one place, much like in our simple example of tightness. More precisely, it is necessary that there be a clause $C \in F$ which overlaps with at least d^k/e other clauses.

[2]We are silently translating to ClSP language here; Erdős and Lovász were talking about the colorability of hypergraphs, a less general but almost equivalent problem.

For $C \in F$, let us define the *(exclusive) neighborhood* of C as

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

and the *inclusive neighborhood* as

$$\Gamma_F^+(C) := \Gamma_F(C) \cup \{C\}.$$

Then Erdős and Lovász proved

Theorem 2.3 ([EL75] Symmetric Lovász Local Lemma for ClSPs)**.** *Let F be a (d,k)-ClSP formula such that for all $C \in F$ we have $|\Gamma_F(C)| \leq d^k/e - 1$. Then F is satisfiable.*

We will explain the original proof (of a stronger, generalized version of this theorem) by Erdős and Lovász in Section 2.2. The question whether this bound is tight or whether stronger statements of a similar type are possible is considerably more difficult than in the case of Theorem 2.1. We will discuss this question briefly in Section 2.10 for completeness, but it is not our main concern.

Much more prominently, we will consider the second question that immediately arises with such an existential theorem: now that we know, from just counting a simple metric in the formula, that there *exists* a satisfying assignment, is it also possible to *find* one using an efficient algorithm? The answer to this question is the main contribution in this chapter and will be our concern for the remainder of sections.

The neighborhood restriction can be thought of in terms of a graph describing the interdependencies between the clauses in the formula. For any ClSP F, we define the *dependency graph G_F of F* to be the graph $G_F = (F, E)$ in which the vertices are the clauses of F and any two distinct vertices are connected by an edge if the corresponding clauses share at least one variable. The notion of neighborhood of a clause now coincides with its neighborhood in the dependency graph, that is if the dependency graph has maximum degree at most $d^k/e - 1$, then the formula is satisfiable.

Inhomogeneous Formulas. The statement of the Lovász Local Lemma can be generalized so as to apply to clause satisfaction problems in which the clauses and the domains of the variables vary in size. In this case, we say that a ClSP *F satisfies the local condition* (or synonymously the *local hypothesis*) if there exists a mapping $\mu : F \to (0,1)$ which associates a number with each clause in the formula such that

$$\forall C \in F \; : \; w(C) \leq \mu(C) \prod_{D \in \Gamma_F(C)} (1 - \mu(D)). \tag{2.1}$$

To shorten notation, we define, for a given mapping μ, the right hand side of the condition on a clause C to be the *μ-weight* of the clause, in writing

$$w_\mu(C) := \mu(C) \prod_{D \in \Gamma_F(C)} (1 - \mu(D)).$$

The local condition thus asks for a mapping μ such that each clause's weight is no larger than its μ-weight.

Theorem 2.4 ([Spe77] General Lovász Local Lemma for ClSPs)**.** *Any ClSP satisfying the local hypothesis is satisfiable.*

At first sight, the local hypothesis may appear very intransparent. What does a formula having this property look like? Looking at the inequality more closely reveals that the condition is at least qualitatively intuitive: the smaller the probability $w(C)$ that a clause is violated, the easier it is to satisfy it. The easier it is to satisfy C, the smaller we can make $\mu(C)$ and/or the more other clauses we can allow to interfere with it, that is we can allow more neighbors in $\Gamma_F(C)$ without making the μ-weight too small. But not just the *number* of neighbors is important, but the impact of each neighbor can be of differing severity. The more difficult it is to satisfy $D \in \Gamma_F(C)$, the larger we will have to choose its μ-value (because of the condition we get instantiating (2.1) for D) and the stronger will be the effect of multiplying by $(1 - \mu(D))$ in the condition for C.

The simpler symmetric version of Theorem 2.3 follows from this general version as a simple corollary. The details of this derivation can be found in Appendix A.7.

Applying the generalized statement of Theorem 2.4 requires us to establish the existence of an assignment of numbers to the clauses satisfying a nontrivial hypothesis. Usually, assigning numbers in the order of $\Theta(d^{-|C|})$ for each $C \in F$ is a good idea. Doing this allows for an intermediate formulation that is much easier to apply while still allowing for inhomogeneous clause sizes.

Theorem 2.5. *Let F be a non-degenerate ClSP. If for all clauses $C \in F$, we have*

$$\sum_{D \in \Gamma_F(C)} w(D) \leq \frac{1}{4},$$

then F is satisfiable.

The derivation of this theorem from Theorem 2.4 can be found in Appendix A.8. It is noteworthy how nicely this formulation compares to the warm-up statement of Theorem 2.2. The constant $1/4$ may be improvable.

Constraint Satisfaction Problems. As we will see, none of the proofs of the Local Lemma make inherent use of the fact that the constraints in the problem are clauses. If a general CSP F is given and if we still consider $w(C)$ for every $C \in F$ to be the probability that a uniformly random assignment violates the constraint C, whatever shape it may have, and if we then define *satisfying the local condition* the very same way we did it for ClSPs, then one shows seamlessly the following.

Theorem 2.6 ([Spe77] General Lovász Local Lemma for CSPs)**.** *Any CSP satisfying the local hypothesis is satisfiable.*

However, each proof as we present it later must be redone with the fact in mind that general CSPs are under consideration. Because,

although every constraint satisfaction problem can be converted into an equivalent clause satisfaction problem by *clause splitting* as mentioned in Section 1.2, this might not be a good idea in the present case. It is namely possible for a CSP to satisfy the local condition prior to splitting and to lose this property after the conversion.

For a trivial example, consider a CSP F over k Boolean variables, consisting of one single constraint C which is satisfied if and only if all variables take value one. The constraint C has a weight of $w(C) = 1 - 2^{-k}$. Assigning value $\mu(C) := w(C)$ establishes the local condition for F and the – in the present case trivial – conclusion is that F be satisfiable.

Now let us split this very hard constraint into the necessary $2^k - 1$ clauses to form a ClSP F'. Each clause $D' \in F'$ has a weight of 2^{-k} and thus $\mu(D') \geq 2^{-k}$ must hold in any choice of values trying to establish the local condition. Suppose there was an association of μ-values which worked. Without loss of generality, all clauses $D' \in F'$ must receive the same value $\mu_{D'} = \gamma(k) \cdot 2^{-k}$ as otherwise we may just regularize them using the value of that clause where the local condition is tightest. However, the value $\gamma(k)$ may depend on k of course. In this case, the local condition at every clause now reads

$$2^{-k} \leq 2^{-k} \cdot \gamma(k) \cdot (1 - 2^{-k} \cdot \gamma(k))^{2^k - 2}$$

from which (using Lemma A.1),

$$1 \leq \gamma(k) \cdot e^{-\gamma(k) + 2^{-k+1}\gamma(k)} \leq \gamma(k) e^{-0.9 \cdot \gamma(k)}$$

follows for sufficiently large k. By differentiation, it can most easily be checked that the right hand side never exceeds $1/2$, establishing a contradiction.

This is in addition to the fact that if k is large, the number of constraints produced from splitting may be enormous and make the problem intractable computationally.

We conclude that generalized constraints have to be kept in mind throughout the chapter and we will consider them where necessary.

Beyond Constraint Satisfaction and Dependency Digraphs. It is possible to generalize the Local Lemma even further, completely leaving the language of constraint satisfaction problems and formulating the lemma in terms of probability theory. Let Ω be a probability space and let furthermore $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be arbitrary events in this space. We say that a graph G on the vertex set $\mathcal{A}$ is a *dependency graph* for $\mathcal{A}$ if each event is mutually independent of all events to which it is not adjacent, i.e.

$$\forall A \in \mathcal{A} : \forall S \subseteq \mathcal{A} \setminus \Gamma_G^+(A) : \Pr\left[A \middle| \bigcap_{B \in S} \overline{B}\right] = \Pr[A]. \tag{2.2}$$

The Local Lemma on general events can be stated as follows.

Theorem 2.7 ([Spe77] Lovász Local Lemma for General Events)**.** *Let Ω be a probability space and $\mathcal{A}$ a finite set of events. Let G be a dependency graph for $\mathcal{A}$. If there exists a mapping $\mu : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} \ : \ Pr[A] \leq \mu(A) \cdot \prod_{B \in \Gamma_G(A)} (1 - \mu(B)), \tag{2.3}$$

then $Pr\left[\bigcap_{A \in \mathcal{A}} \overline{A}\right] > 0$.

It is trivial to see that Theorem 2.4 follows from Theorem 2.7 by considering the random experiment of drawing an assignment uniformly, with the events corresponding to violating the different clauses. Clearly, the dependency graph of the formula is a dependency graph for the probabilistic setting as any clause C being violated is clearly independent of any set of events in which no variable of C occurs. The more general version can however be applicable also in situations where Ω does not have the variables-with-finite-domains

product structure that would be required to model the task in hand as a clause or even constraint satisfaction problem.

A way of yet strengthening this statement is to define *directed dependency graphs* (as for instance done in [AS00]). If D is a digraph on $\mathcal{A}$, then it is called a *dependency digraph* if each event is mutually independent of all events which are its non-outneighbors. Then we obtain

Theorem 2.8 ([AS00] Directed Lovász Local Lemma for Events)**.** *Let Ω be a probability space and $\mathcal{A}$ a finite set of events. Let D be a dependency digraph for $\mathcal{A}$. If there exists a mapping $\mu : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} \ : \ Pr\left[A\right] \leq \mu(A) \cdot \prod_{B \in \Gamma_D(A)} (1 - \mu(B)), \tag{2.4}$$

then $Pr\left[\bigcap_{A \in \mathcal{A}} \overline{A}\right] > 0$.

As is natural, $\Gamma_D(A)$ is in this case understood to be the set of outneighbors of vertex A in D.

Using directed graphs adds no value to the case of clause satisfaction problems. If two clauses are dependent because their underlying variable sets intersect, then this dependence is symmetric: if a dependency digraph for a clause satisfaction problem contains a single directed arc between two clauses, either that arc is superfluous because the clauses are strictly disjoint, or the converse arc is also necessary and thus it was not a proper dependency digraph in the first place. We will therefore not study the case of digraphs intensively as it is not applicable to our problem of interest.

Lopsidependency. Contrary to the case of digraphs, another strengthening of Theorem 2.7 is possible which does add value to the case of clause satisfaction problems. This strenghtening arises from relaxing the requirement for dependency graphs in such a way that we do not require mutual independence of events from their non-neighbors, but

only that the correlation goes the right direction. More specifically, a graph G is called a *lopsidependency graph* for the set of events $\mathcal{A}$ if no event is more likely in the conditional space defined by intersecting the complement of any subset of its non-neighbors, i.e.

$$\forall A \in \mathcal{A} : \forall S \subseteq \mathcal{A} \setminus \Gamma_G^+(A) : \Pr\left[A \middle| \bigcap_{B \in S} \overline{B}\right] \leq \Pr[A]. \tag{2.5}$$

The Lopsided Local Lemma on general events then reads just as the standard one but with the condition on G relaxed to be a lopsidependency graph only.

Theorem 2.9 ([ES91] Lopsided Version of Lovász Local Lemma)**.** *Let Ω be a probability space and $\mathcal{A}$ a finite set of events. Let G be a lopsidependency graph for $\mathcal{A}$. If there exists a mapping $\mu : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} \ : \ Pr[A] \leq \mu(A) \cdot \prod_{B \in \Gamma_G(A)} (1 - \mu(B)), \tag{2.6}$$

then $Pr\left[\bigcap_{A \in \mathcal{A}} \overline{A}\right] > 0$.

When applied to clause satisfaction problems, the relaxation to lopsidependency graphs corresponds to having to consider two clauses to be neighbored not if they share a common variable, but only if they *conflict* in at least one literal. Two clauses C, D are *conflicting* if there is a variable x and two distinct values $v_1, v_2 \in L_x$ such that $(x \neq v_1) \in C$ and $(x \neq v_2) \in D$.

Suppose now that F is a ClSP formula and consider the graph on the clauses of F such that there is an edge between any two clauses C and D iff they conflict in some literal. We call this graph the *conflict graph* G_F^* of F.

Then, in the random experiment where we draw assignments uniformly at random, G is clearly a lopsidependency graph: let $C \in F$ be any clause and $\mathcal{D} \subseteq F$ a set of clauses such that no clause in $\mathcal{D}$

conflicts with C. We are interested in the probability that C becomes violated conditioning on all of $\mathcal{D}$ being satisfied. From the fact that if an assignment α satisfies $\mathcal{D}$ and violates C, *all* assignments arising from α by changing any values for $\mathrm{vbl}(C)$ still satisfy $\mathcal{D}$, follows that the probability that C is violated given that $\mathcal{D}$ is satisfied is still at most d^{-k}. Therefore G_F^* is a lopsidependency graph.

This yields the following strengthening of Theorem 2.4. For any $C \in F$, we define the *lopsineighborhood*

$$\Gamma_F^*(C) := \{D \in F \mid \exists x \in V :$$

$$\exists v_1, v_2 \in L_x : v_1 \neq v_2, (x \neq v_1) \in D, (x \neq v_2) \in C\}$$

to be the set of all neighbors of C in the conflict graph for F, i.e. all clauses which have a conflict with C. Then we have obtained

Theorem 2.10 (Lopsided Lovász Local Lemma for ClSP)**.** *Let F be a ClSP. If there exists a mapping $\mu : F \to (0,1)$ that associates a number with each clause in the formula such that*

$$\forall C \in F \ : \ w(C) \leq \mu(C) \cdot \prod_{D \in \Gamma_F^*(C)} (1 - \mu(D)), \tag{2.7}$$

then F is satisfiable.

The lopsided variant extends seamlessly also to the setting of directed graphs as discussed in the previous paragraph. For aforementioned reasons, we are not overly interested in this case and forgo stating this as yet another variant.

Variable Occurrences. The lopsided version of the LLL has been successfully applied for example by Berman et al. [BKS03] in order to prove better bounds on the number of occurrences per variable we can allow in a k-SAT formula while still guaranteeing satisfiability.

Indeed, an immediate consequence of Theorem 1 is the following.

Corollary 2.11 ([KST93]). *Whenever in a (d,k)-ClSP formula F, every variable occurs in at most d^k/ek clauses, then F is satisfiable.*

Berman et al. [BKS03] have refined this result for the Boolean case $d = 2$, eventually leading to the following asymptotic result proved by [GST11].

Theorem 2.12 ([GST11]). *If in a k-SAT formula F, every variable occurs a total of at most $\frac{2}{e} \cdot \frac{2^k}{k} - 1$ times, then F is satisfiable.*

Heidi Gebauer has demonstrated in her award paper [Geb09] that asymptotically, this bound is tight. Her result was refined (with the constant improved) in [GST11], yielding the following.

Theorem 2.13 ([GST11]). *For every $\epsilon > 0$ and k sufficiently large there exists a k-SAT formula where every variable occurs at most $\left(\frac{2}{e} + \epsilon\right) \frac{2^k}{k}$ times and which is unsatisfiable.*

Both theorems were formulated with the case of Boolean satisfiability in mind, but a standard generalizing construction[3] produces unsatisfiable (d,k)-ClSP formulas where every variable occurs at most

$$\left(\frac{2}{e} + \epsilon\right) \frac{1}{k} \left(2 \cdot \left\lceil \frac{d}{2} \right\rceil\right)^k$$

times. For even d, this demonstrates tightness up to a constant factor independent of d and k. For odd d, there is an error due to parity issues which can most probably be resolved with a more careful approach. We do not include any proofs here, as this topic is not our main concern.

An interesting observation is that for very small values of d and k, the Local Lemma does not yield interesting bounds. For example if

[3] Where you take an unsatisfiable k-SAT formula and replace each clause by $\lceil d/2 \rceil^k$ clauses over d-ary variables.

$d = 2$ and $k = 3$, i.e. the prominent case of 3-SAT, Corollary 2.11 and Theorem 2.12 both yield trivially small bounds on the number of permissible occurrences. For such cases, better bounds can be obtained without the Local Lemma.

Indeed, for any given ClSP F, consider the bipartite graph I_F in which one partite set is formed by the clauses of F, the other partite set is formed by the variables of F and there is an edge between a clause and a variable if this variable occurs in that clause. I_F is called the *clause-variable incidence graph* of F. This graph contains more detail than the dependency graph G_F. In fact, G_F is obtained from I_F by taking the square of I_F and then dropping the variable vertices as well as all loops. We call a formula F *matched* if there exists a matching in I_F which exhausts the clause vertices.

Observation 2.14 ([Tov84],[KST93])**.** *Matched ClSP formulas are satisfiable.*

This follows trivially because if a clause-saturating matching is given, each clause has a variable which can be declared solely responsible for satisfying this clause. Using the Marriage Theorem by Philip Hall [Hal35], one can prove that if every variable has degree at most k whilst every clause contains exactly k variables, then the clause-variable incidence graph always admits a clause-saturating matching.

Corollary 2.15 ([Tov84],[KST93])**.** *Every (d, k)-ClSP in which every variable occurs at most k times is matched and hence satisfiable.*

The investigation of the function $f(k)$ defined as the largest integer s with the property that every Boolean k-SAT formula in which every variable occurs at most s times is satisfiable dates back to work by Tovey [Tov84] who demonstrated (using the Marriage Theorem as outlined) that 3-SAT formulas with at most 3 occurrences per variable are satisfiable whilst the problem of deciding satisfiability of 3-SAT formulas with up to 4 occurrences per variable is NP-complete.

Kratochvíl, Savický and Tuza [KST93] further observed that for arbitrary k, allowing more than $f(k)$ variables made the decision problem NP-complete and they were the first to obtain an exponential bound of type $f(k) \geq 2^k/(ek)$ from the Local Lemma (cf. Corollary 2.11). Further insights into this area have finally been provided by Dubois [Dub90], by Hoory and Szeider [HS05, HS06] and finally by Gebauer, Szabó and Tardos [GST11] as mentioned above.

Outline of the Chapter. So far, we have introduced various flavors of the Lovász Local Lemma and we have illustrated how it may be applied to clause and constraint satisfaction problems in various ways. The strongest LLL variant we have stated is the one in Theorem 2.9 and all other variants – with the sole exception of the digraph variant which we do not study further – follow from this strongest version. Next, in Section 2.2, we will go on to show how Erdős, Lovász (and Spencer) originally proved this theorem using the probabilistic method. This proof will be most elegant and short, but it will not disclose any way in which satisfying assignments to problems for which satisfying assignments can be thus guaranteed can be actually discovered using an efficient algorithmic procedure. In Section 2.3, we will start to concern ourselves with the problem of providing an algorithmic variant of the LLL. After analyzing the history of the problem we will present a randomized iterative procedure extraordinarily suited to the purpose. The subsequent sections are to discuss the analysis of that algorithm. After discussing the basic idea in Secion 2.4, Section 2.5 will give a first simple bound on the running time of this algorithm in simplified homogeneous scenarios comparable to the statements of Theorem 2.3 and Theorem 2.5 using a nice and intuitive proof strategy illustrating the philosophy behind all arguments presented thereafter. From Section 2.6, we will study a more refined proof technique capable of making almost all known applications of the Local Lemma algorithmic. In Sections 2.7, 2.8 and 2.9, we will finally look into parallel and deterministic variants of our algorithm.

2.2 Nonconstructive Proof[4]

We now explain the way Erdős, Lovász and Spencer originally proved Theorem 2.9 using a probabilistic argument.

Let Ω be a probability space and $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be any set of events. Let G be a lopsidependency graph for $\mathcal{A}$ and let an association $\mu : \mathcal{A} \to (0, 1)$ be given such that the lopsided local condition (that is Inequality (2.6)) is satisfied.

We prove that there is a positive probability $p > 0$ that none of the events in $\mathcal{A}$ occur. Even though this probability may be extremely small, any non-zero value readily certifies that there must exist *some* point $\omega \in \Omega$ avoiding all the events – which corresponds to a satisfying assignment in the ClSP or CSP case.

In order to show $p > 0$, we first prove the following auxiliary claim.

Lemma 2.16 ([EL75])**.** *Let $A \in \mathcal{A}$ be any fixed event and $H \subseteq \mathcal{A}$ any subset. Then*

$$Pr\left[A \;\middle|\; \bigcap_{D \in H} \overline{D} \right] \leq \mu(A).$$

Proof. The proof is by induction on the size $|H|$ of the set of events we condition on.

The base case for $H = \emptyset$ is easy. In this case we have

$$\Pr\left[A \;\middle|\; \bigcap_{D \in H} \overline{D} \right] = \Pr[\, A \,] \leq \mu(A),$$

where the last inequality uses the local condition.

[4]The material in this section is from [EL75] and [Spe77] and appears in many standard textbooks. The write-up presented here has already appeared similarly in [MW11].

Now suppose the claim is true for all sets H' smaller than H, in particular for all sets $H' \subset H$.

Now we distinguish three cases. For the first one, suppose $A \in H$. Then the statement is trivial because clearly, the probability of having A while at the same time having $\overline{A}$ is clearly nonexistent, so

$$\Pr\left[A \;\middle|\; \bigcap_{D \in H} \overline{D} \right] = 0 < \mu(A).$$

From now on we may assume $A \notin H$.

For the second case, if in H, there is *no* event $D \in H$ such that $D \in \Gamma_G(A)$. In this case, the statement is again trivial and using the induction hypothesis is not necessary, as the definition of the lopside-pendency graph and the local hypothesis entail that

$$\Pr\left[A \;\middle|\; \bigcap_{D \in H} \overline{D} \right] \leq \Pr[\, A \,] \leq \mu(A).$$

From now on we are allowed to assume that $H \cap \Gamma_G(A) \neq \emptyset$.

Then split[5] the set of events H into the two parts $H = H_A \dot{\cup} H_0$ where $H_A := H \cap \Gamma_G(A)$ are the events potentially interferring negatively with A and the remainder $H_0 := H \backslash H_A$ are the ones that do not. Now we use the definition of conditional probability to write

$$\Pr\left[A \;\middle|\; \bigcap_{D \in H} \overline{D} \right] = \frac{\Pr\left[A \cap \bigcap_{D \in H_A} \overline{D} \;\middle|\; \bigcap_{D \in H_0} \overline{D} \right]}{\Pr\left[\bigcap_{D \in H_A} \overline{D} \;\middle|\; \bigcap_{D \in H_0} \overline{D} \right]} \tag{2.8}$$

and then bound the numerator and the denominator of this fraction as follows.

[5]Note that the previous two cases could technically be subsumed into this last one, but we consider the two preliminary cases to be an illustrative preparation for the general case.

For the numerator, note that the probability that A occurs and all of H_A do not is certainly smaller, in any conditional space, than just the probability that A occurs, i.e.

$$\Pr\left[A \cap \bigcap_{D \in H_A} \overline{D} \;\middle|\; \bigcap_{D \in H_0} \overline{D} \right] \leq \Pr\left[A \;\middle|\; \bigcap_{D \in H_0} \overline{D} \right] \leq \Pr[\, A \,], \quad (2.9)$$

where the last equality uses the lopsidependency graph hypothesis, i.e. the fact that the events in H_0 are at most negatively correlated with A. For the denominator, we explicitly list

$$H_A = \{D_1, D_2, \ldots, D_r\}$$

and then use once more the definition of conditional probability, i.e. for any events $\{E_i\}_{i \in \{1..r\}}$,

$$\Pr\left[\bigcap_{i=1}^{r} E_i \right] = \prod_{i=1}^{r} \Pr\left[E_i \;\middle|\; \bigcap_{j=1}^{i-1} E_j \right],$$

to write

$$\Pr\left[\bigcap_{D \in H_A} \overline{D} \;\middle|\; \bigcap_{D \in H_0} \overline{D} \right] = \prod_{i=1}^{r} \underbrace{\Pr\left[\overline{D_i} \;\middle|\; \bigcap_{j=i+1}^{r} \overline{D_j} \cap \bigcap_{D \in H_0} \overline{D} \right]}_{=: p_i}.$$

Now we use the induction hypothesis for each

$$H_i := \{D_{i+1}, D_{i+2}, \ldots, D_r\} \cup H_0$$

for $1 \leq i \leq r$, each of which is a strict subset of H and thus $p_i \geq 1 - \mu(D_i)$. This yields

$$\Pr\left[\bigcap_{D \in H_A} \overline{D} \;\middle|\; \bigcap_{D \in H_0} \overline{D} \right] \geq \prod_{i=1}^{r} (1 - \mu(D_i)) \geq \prod_{D \in \Gamma_G(A)} (1 - \mu(D)). \quad (2.10)$$

Finally, we plug (2.9) and (2.10) into (2.8) and obtain that

$$\Pr\left[A \;\middle|\; \bigcap_{D\in H} \overline{D} \right] \leq \frac{\Pr[\, A \,]}{\prod_{D\in\Gamma_G(A)}(1-\mu(D))} \leq \mu(A)$$

where the last inequality follows from Hypothesis (2.1). This finishes the induction and proves the lemma. □

Having the lemma at our disposal, the proof of the famous Lovász Local Lemma now becomes simple.

Proof of Theorem 2.9. Explicitly list all events

$$\mathcal{A} = \{A_1, A_2, \ldots, A_m\}.$$

Then use the definition of conditional probability to see that

$$p = \Pr\left[\bigcap_{i=1}^{m} \overline{A_i} \right] = \prod_{i=1}^{m} \Pr\left[\overline{A_i} \;\middle|\; \bigcap_{j=i+1}^{m} \overline{A_i} \right] \geq \prod_{i=1}^{m} (1-\mu(A_i)),$$

where the inequality is by Lemma 2.16. Since $\mu(A) < 1$ for all $A \in \mathcal{A}$, the expression on the right is non-zero and thus $p > 0$. □

Let us finally also note that Lemma 2.16 gives a nice intuitive interpretation of the values $\mu(\cdot)$. Roughly speaking, these should bound the probability of the corresponding events in the conditional space where all neighbors are being avoided.

2.3 Algorithmization

However elegant the proof of (Theorem 2.9 and thus of) Theorem 2.4 we have seen, it is deeply non-constructive and does not disclose any way in which a satisfying assignment to a given ClSP satisfying the

local condition could be discovered systematically. In the sequel, we will demonstrate that an efficient randomized iterative procedure for solving such formulas can be devised quite easily.

Nontriviality. To see that this is a nontrivial question, we dismiss the simple attempt consisting of just sampling assignments uniformly at random and hoping they satisfy the clause satisfaction problem as follows.

Consider the (d,k)-ClSP over n variables which consists of n/k independent subformulas, each of which consists of an arbitrary selection of d^k/e clausess over the same k variables. Since every connected component of this formula is satisfied by a random assignment with probability $1/e$, the whole formula is satisfied with probability $\exp(-n/k)$, an exponentially small probability.

If this example is considered too artificial, let $\{F_n\}_{n\in\mathbb{N}}$ be *any* family of (d,k)-ClSP formulas where for all n, F_n is over n variables and, let us just say, kn clauses, and satisfies the local condition. Such a rich class contains as many non-artificial examples as the Local Lemma can be applied to. Now produce randomized copies $\{H_n\}_{n\in\mathbb{N}}$ obtained by randomly changing the value of each literal in the formula independently and uniformly at random. Since the hypothesis of the Local Lemma ignores the values of literals, all random formulas so produced are satisfiable. However, the expected number of satisfying assignments of H_n is very small: for a fixed $\alpha : V_n \to \{1..d\}$, the probability that α survives the randomization of all the literals is at most $(1-d^{-k})^{kn}$ which tends to zero exponentially quickly. At the same time, this is the expected probability with which a uniformly random assignment satisfies H_n (where the expectation is w.r.t. H_n and the probability w.r.t. the random assignment).

Thus families of formulas which satisfy the hypothesis of the Local Lemma where uniform sampling has an exponential expected running

time are easily constructed and we find that we have to be more clever to devise an algorithmic solution.

Method of Conditional Expectations. The same is not true for simple applications of the probabilistic method as for example (d, k)-ClSP formulas F satisfying the precondition of Theorem 2.1. For such formulas, random sampling is successful: having at most $d^k - 1$ clauses, each one forbidding a d^{-k}-fraction of all assignments, a uniformly random assignment satisfies the formula with probability at least d^{-k}. Now either the size of F is in the same range, let us say larger than $(d^k)/2$ making this approach polynomial in $|F|$, or if F has at most $(d^k)/2$ clauses, then the probability of hitting a satisfying assignment grows beyond $1/2$, making the approach polynomial as well.

There is a very well-known derandomization of this technique due to Erdős and Selfridge [ES73] which is an integral component of the probabilistic method and is known as the *method of conditional expectations*. To solve F deterministically rather than through random sampling, an algorithm can explicitly calculate the expected number of clauses violated by a (hypothetical) uniformly random assignment. This number must be below one for the existential argument to apply. Now, the algorithm can pick an arbitrary variable x and cycle through all possible values L_x that can be assigned to it. For each value $d \in L_x$, it considers the formula $F^{[x \mapsto d]}$ arising from substitution of d for x in F and calculates once more the expected number of violated clauses if the remaining formula were to be subjected to random sampling. For at least one of the possible values, this number must stay well-below one since we know that the average was below one. Pick this value and assign it to x permanently, then iterate. Since the expected number of violated clauses stays below one all the time while the set of unassigned variables shrinks, at some point, the expected number of violated clauses must drop to zero and the satisfying assigment has been discovered.

To the more intricate Local Lemma scenario, such simple approaches do not apply, although the method of conditional expectations will be a vital tool in the derandomization which we will investigate much later in Section 2.9.

Problem History and the Beck-Alon Approach. The first successful attempt at making the Local Lemma constructive was made by József Beck in [Bec91] and subsequently simplified by Noga Alon in [Alo91]. The Beck-Alon approach can be summarized as follows. Consider the Boolean case $d = 2$ and thus a k-SAT formula F.

We pick an initial assignment α_0 uniformly at random. Next, we apply a fixed *freezing rule* $\mathcal{R}$ which takes the current assignment α_0 and F and determines a set of variables $V_f := \mathcal{R}(F, \alpha_0)$ which are to be frozen, that is, assigned for good according to α_0. F is then replaced as

$$F' := F^{[\alpha_0|_{V_f}]}$$

and F', the remaining formula over variables $\mathrm{vbl}(F) \setminus V_f$, is then subsequently solved by means of a brute-force algorithm which identifies the connected components of F' (i.e. of $G_{F'}$) and enumerates exhaustively all solutions to each component. The hope is that if the freezing rule is suitable, then all connected components will, with high probability, be small enough for the procedure to have a polynomial running time.

The freezing rule used in [Alo91] is as simple as letting $\mathcal{R}(F, \alpha_0)$ be all variables not occurring in any clause with fewer than $k/8$ literals satisfied by α_0. After freezing these variables, each clause $C \in F$ can be in one of three states: either C is *safe* because $\alpha_0|_{V_f}$ contains an assignment satisfying C so that C is no longer represented in F'. Then, C can be *bad* which means that more than $7k/8$ literals in C are violated by α_0 and in this case, all variables in C are kept non-frozen by the freezing rule so that $C \in F'$ gets another chance to become satis-

fied. The remaining case occurs when C contains at least $k/8$ satisfied literals, but all of those are over variables that are kept non-frozen by the freezing rule. Such clauses are considered *dangerous*. It is easy to see that all bad clauses contain k and all dangerous clauses contain at least $k/8$ literals over non-frozen variables, so they get a chance to be satisfied in the subsequent reassignment phase.

The freezing rule, in particular the fraction of $k/8$, is chosen in such a way that one can prove two things: firstly that with high probability, the dependency graph of F' contains connected components no larger than logarithmic in size exclusively, to which all possible assignments can be enumerated in polynomial time. Secondly, for the procedure to go through successfully, we need that a solution *exists* in each connected component. Such a guarantee is what the Local Lemma can provide via its non-constructive proof as outlined in Section 2.2, at least as long as the hypothesis on the neighborhood sizes is strong enough to also cope with clauses of size $k/8$. This is obviously the case if we ask the neighborhoods to be as small as the 8^{th} root of what is required for the Local Lemma to work. Hence what Alon, following up Beck, proved was that

Theorem 2.17 ([Alo91])**.** *There exists a constant c such that there is a randomized expected polynomial time algorithm, which on input F a k-SAT formula for which we have $|\Gamma_F(C)| \leq c2^{k/8}$ for all clauses $C \in F$, outputs a satisfying assignment.*

Without going into the technical details which we encourage the reader to look up in [Alo91], we would like to sketch the main idea of the proof, as we will use very similar ideas to prove our stronger version later.

Central is the introduction of the concept of *witness trees*, which are certificates for unlucky executions of the algorithm, in the following sense. Imagine a *witness tree of size s* to be any subtree of the dependency graph G_F of F containing at least s vertices. What the hypothesis

of the theorem entails is that each vertex in the dependency graph has a degree no larger than $c2^{k/8}$. This limits as well the number of distinct witness trees of size s there can be which is easily shown (see [Knu73]) to be limited by $(ec2^{k/8})^s \cdot |F|$. Now suppose that all clauses of a fixed witness tree are bad or dangerous after picking α_0. Then there exists a connected component in $G_{F'}$ containing all clauses in the witness tree, so the algorithm will have to enumerate solutions over a set of variables of this magnitude. Conversely, whenever there is a connected component in $G_{F'}$ of a certain size s, this component has a spanning tree and this spanning tree is a witness tree inside the dependency graph of F of size s all of whose vertices are bad or dangerous. One can now compute the probability that for a fixed witness tree, all the vertices become bad or dangerous and one will find, in very rough terms, that this probability is exponentially smaller than the inverse of the number of possible witness trees, leading to the conclusion that for sufficiently large $s = \mathcal{O}(\log n)$, with high probability, no tree of size s will at all exist all of whose vertices are bad or dangerous after the preassignment phase and thus no component of this size will remain in F' to be treated for the reassignment phase.

The basic idea is thus to juxtapose the number of possible ways in which something can go wrong in this algorithm (where a "way something can go wrong" is a large connected component left after the preassignment and freezing stage, and each such way is represented by a witness tree) with the probability that each single bad situation can occur (i.e. the probability with which all of a witness tree's vertices turn bad and dangerous) when picking α_0 at random. Then we find that for well-chosen parameters, the total probability measure accumulated by bad outcomes is small and conclude that thus most of the time, the algorithm must succeed. This idea will stay the same in all approaches we present hereafter.

One can also take an information-theoretic viewpoint on this situation: instead of talking about the "number of possible ways" some-

thing can go wrong, we can as well talk about the algorithm producing a certificate (in the present case a witness tree) so as to justify why it had to surrender inconclusively and then juxtapose the number of bits we need to encode a bad outcome (a witness tree) with the number of random bits (in the choice of α_0) that were contributing to (and thus could have prevented if chosen differently) this bad outcome. Then we find that for well-chosen parameters, the number of bits needed to encode a bad outcome is smaller than the number of random bits needed to steer the algorithm into the corresponding undesirable state. We conclude that most of the times the algorithm must succeed, as otherwise we would have too good a chance to find a compact representation for the random input bits. Both of these views will be taken later when analyzing an improved version of this constructive method.

The main issue with this approach is the derivation of a bound on the probability with which all vertices of a witness tree can turn bad or dangerous. Since these vertices are connected, there are dependencies between the clauses involved. Beck and Alon circumvented this problem by considering subsets of the vertex set of such a tree, all of whose members are at a certain distance from one another. *All* vertices cannot turn bad unless at least those chosen vertices do and these events then become independent by virtue of being at a sufficient distance in the dependency graph. If a vertex set is given all of whose vertices are bad, then this information allows to resconstruct partially the random bits used at the corresponding clauses. This yields a viable bound on the probabilities, but the introduction of distance produces a loss and calls for the reduction of the neighborhood sizes we can allow.

To alleviate this loss, one has to find a way of representing, in an information-theoretic sense, the "bad outcomes" more effectively and to find witness trees from which more information about the preliminary assignment can be reconstructed than just parts of the values of random samples at vertices at a constant distance. In [Mos06], we made a first such attempt, the idea of which was to decrease the dis-

tance between the estimate-supporting vertices at places where this is possible. This makes estimates more precise and allows us to adapt the freezing-rule to freeze fewer variables. The yield was that a neighborhood bound approximately $2^{k/6}$ would suffice for k-SAT. Later and independently, Srinivasan investigated different strategies of a similar spirit and could cope with neighborhood bounds proportional to $2^{k/4}$ [Sri08]. Srinivasan's work also included other improvements making the algorithmization more widely applicable. In a similar spirit, Molloy and Reed [MR98] had previously provided a general framework as to when a Beck-Alon type algorithm could be provided for an application of the Local Lemma. For a next stage, we provided an improved construction of denser witness trees catering for neighborhood sizes of up to roughly $2^{k/2}$ [Mos08]. The idea was both to yet again make the witness structures denser, and also to use the information encoded in the edges of witness trees for reconstruction of more random bits, rather than just the information contained in the collection of vertices they contain. As many witness trees can be built over the same set of vertices and the tree's structure thus contains additional bits, this again allows for a tighter freezing rule.

Still, there remained a gap for an inherent reason: all algorithms considered dwelled on the principle of having a *preassignment phase*, then freezing and truncation of clauses and a *reassignment phase*. As long as we stick to this two-phased basic method, two things stand in our way: for one, we need the Local Lemma to work even after truncation of the clauses, necessarily asking for smaller neighborhood sizes. And secondly, as long as we use witnesses to certify bad outcomes of one single preassignment only, there will *always* be dependencies between different clauses turning bad or dangerous as these events depend on the same random samples and so we will *never* be able to use *all* information a witness can possibly encode for the reconstruction of what random choices lead to the occurrence of that witness. All information must however be used if a tight result is to be achieved.

But there is an easy way out: if instead of doing one preassignment and one reassignment, we iteratively reassign values to currently violated clauses and then build a witness tree representing the succession of reassignments that have taken place, then all random samples partaking in this succession of bad events as represented by the witness trees are mutually independent: each time a sample becomes relevant by making a clause violated, it is immediately replaced by a new, yet unknown random sample. In a witness tree representing the sequence of clauses violated and resampled, even if over dependent clauses, each vertex allows for the reconstruction of all randomness previously assigned to the clause it represents because this randomness has been used once only and is replaced before the next vertex will be embedded into the tree. This is the intuition behind the strategy we are going to follow hereafter.

Iterative Corrections. Consider Algorithm 1, which in each iteration, picks a violated clause and then replaces the assignments to the variables in that clause with fresh random samples. Let us argue for correctness and termination of this procedure first.

If the algorithm terminates, then by virtue of the termination criterion of the loop, the output clearly constitutes a satisfying assignment.

To see that the algorithm always terminates, just fix an assignment α^* satisfying F and then note that at any point in time, there is a non-zero probability that the algorithm only samples values according to α^* during the next step. Since in each step, at least one of the k values in the C selected disagrees with α^* (after all C is satisfied by α^* but violated by the current assignment), this results in at least one more variable correctly set in each step and thus at any point in time, there is a non-zero probability depending only on F that the algorithm terminates within the next at most n iterations. If we iterate arbitrarily often, this is bound to happen at some point and so the algorithm runs indefinitely with probability zero. For the same reason, the expected running time must also be finite, albeit for all we know right now, it

could be exponential.

Algorithm 1 LocalSolver(F)

Require: A satisfiable ClSP formula F.
Ensure: Output is a satisfying assignment.

1: $\alpha \leftarrow$ a uniformly random assignment from[6] $\mathcal{S}$
2: **while** $\exists C \in F : \alpha$ violates C **do**
3: $\quad C \leftarrow$ any clause violated by α
4: $\quad$ **for** $x \in \mathrm{vbl}(C)$ **do**
5: $\quad\quad \alpha(x) \leftarrow$ new uniformly random value from L_x
6: $\quad$ **end for**
7: **end while**

Note that this argument works for *all* satisfiable formulas F. This is the type of argument that Uwe Schöning [Sch99] has used for analyzing the (very similar) random walk algorithm that we will discuss in the next chapter. There, in Section 3.2, we will use a more refined variant to derive an exponential bound on the running time of Algorithm 1 which holds for *all* formulas.

In the present chapter however, we are concerned with formulas satisfying the local condition and as it will turn out, Algorithm 1 is particularly well-suited to this type of formulas. More precisely, we will prove the following.

Theorem 2.18. *Let F be a ClSP formula satisfying the local condition with mapping μ. Then Algorithm 1 on input F terminates after at most*

$$\mathcal{O}\left(\sum_{C \in F} \frac{\mu(C)}{1-\mu(C)}\right)$$

correction steps on expectation.

[6] Recall that we use the notation $\mathcal{S} := \{\alpha : V_n \to L \mid \alpha(x) \in L_x\}$ for convenience.

Whether this is a good running time depends of course on the mapping μ. However, we can easily check that all reasonable formulas satisfying the hypothesis admit an assignment μ such that the bound becomes polynomial in the size of the formula.

Lemma 2.19. *There exists a universal constant $\Lambda \in (0,1)$ with the following property. If F is any non-degenerate ClSP formula satisfying the local condition with some mapping μ, then there exists a mapping μ' also satisfying the hypothesis such that $\forall C \in F : \mu'(C) \leq \Lambda$.*

The proof is merely technical and is therefore being carried out in the appendix[7].

If we use the mapping μ' guaranteed to exist for F by Lemma 2.19, then none of the values $\mu'(C)$ for $C \in F$ exceed the universal constant Λ such that, by virtue of $x/(1-x)$ being monotone in x, each summand in the bound established by Theorem 2.18 is bounded by a constant. Therefore, the algorithm's expected running time (measured in terms of the number of resamplings) is linear on all non-degenerate formulas.

Corollary 2.20. *If F is a non-degenerate ClSP satisfying the local condition, then Algorithm 1 terminates after an expected $\mathcal{O}(|F|)$ correction steps.*

In particular, of course, any formula satisfying the requirements of the symmetric Theorem 2.3 or the simplified asymmetric Theorem 2.5 is solved after an expected linear number of iterations this way. We also note that the corollary can easily be extended to degenerate cases (if it is considered to be of interest). As we decribed in Section 1.2, there are trivial manipulations which remove all degeneracies from a ClSP. All such manipulations preserve the local condition. Empty domains and constraints cannot occur because the formula is satisfiable.

[7] Lemma A.5 makes a more general statement and since in a non-degenerate ClSP, all weights are at least $1/2$, the claim follows.

Substituting values for variables with a singleton domain does not influence any of the inequalities, substituting an unusued value and thereby satisfying some clauses only relaxes them. The only somewhat non-trivial case arises when we remove a unit clause, thereby shrinking the contained variable's domain. But, as can easily be checked, this, too, preserves all conditions at the neighboring clauses because the effect of removing the clause and the effect of shrinking the domain nicely cancel out.

Finally, we may ask the question whether the algorithm generalizes to constraint satisfaction problems in which the constraints are not clauses. There is nothing which prevents the algorithm from running on *any* such instance. As long as there is *any* efficient way of finding a violated constraint, everything works out.

Theorem 2.21. *Let any CSP F be given along with an algorithm A which identifies, for any non-satisfying assignment, a constraint which is violated. And suppose F satisfies the local condition. Then Algorithm 1 (where we now allow a general CSP as input) on input F terminates after at most*

$$\mathcal{O}\left(\sum_{C \in F} \frac{\mu(C)}{1-\mu(C)}\right)$$

resampling steps (and calls to A) on expectation.

At the end of Section 2.6, it will be discussed what is additionally needed to prove this generalized version.

2.4 Journaling and Reconstruction

In the following, it will be our goal to prove Theorem 2.18 using a variety of tools. In Section 2.5, we will prove a weaker variant of the theorem using simpler arguments that capture the key philosophy nicely. In Section 2.6, we will then proceed to full generality. The main idea

will be the same both times: to find a concise way of representing the actions the algorithm has taken and then - in case of an unexpectedly long running time - juxtaposing the amount of randomness that has led to the large number of corrections, proving that long running times are unlikely. It is the same philosophy on which already the Beck-Alon approach bases.

In the present section we will study the key concept common to all later proofs: the fact that it is sufficient to record the succession of clauses being corrected by the algorithm, as this allows to reconstruct all random values assigned to variables during the process.

Journaling. We introduce formal notation for what we are recording. For a complete account of everything which happens during a run of the algorithm, let $X_1^{(0)}, X_2^{(0)}, \ldots, X_n^{(0)}$ denote the values of the variables in the beginning, that is $X_i^{(0)}$ is the assignment initially chosen for variable $x_i \in V$, for all $1 \leq i \leq n$. Note that these are random variables, and according to how the algorithm works, $X_i^{(0)}$ is distributed u.a.r. among all possible values in L_{x_i}. Now suppose the assignment does not satisfy F, then there will be a clause, call it C_1, selected for correction in the first step. Note that C_1 is a random variable too, distributed somehow among all clauses in F. Next, the algorithm forgets about the values of all variables in C_1 and replaces them with new random samples. Suppose $\mathrm{vbl}(C_1) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$. Then denote by $X_{i_1}^{(1)}, X_{i_2}^{(1)}, \ldots, X_{i_k}^{(1)}$ the new random values these variables receive now. Note again that each of these is a random variable where X_{i_j} is distributed u.a.r. among all possible values in $L_{x_{i_j}}$. In particular, these new random values are completely independent of the values we used before.

Let us continue in this way and in general write C_t to denote the clause corrected in the t^{th} step and $X_i^{(j)}$ to denote the random variable that represents the value which the variable x_i received when it was

deleted and resampled for the j^{th} time in the process. Note that each variable has its own counter, and those counters are not synchronous. In total, not every variable is resampled for the same number of times. Overall, we now have a finite or infinite[8] sequence $\mathcal{C} := \langle C_1, C_2, \ldots \rangle$ of selected clauses which we call the *journal* of the algorithm. Additionally, for every variable $x_i \in V$, we have a finite or infinite[8] sequence $\mathcal{X}_i := \langle X_i^{(0)}, X_i^{(1)}, \ldots \rangle$ of values assigned to x_i during the process which we call the *history* of x_i.

Writing down all the values of all the variables gives a complete account of all actions the algorithm has taken in one particular run.

Reconstruction. We now make the observation that in order to be able to follow the execution history of the algorithm, it is sufficient to remember the journal $\mathcal{C}$. The histories $\mathcal{X}_i$ of all the variables are then not additionally needed because they are already determined (almost, up to the very last assignments on termination) completely by $\mathcal{C}$.

For the purpose of formalization, we represent the information that some piece of information gives us about histories of variables by sets. A *reconstruction set* is a set $R \subseteq \{1..n\} \times \mathbb{N}_0 \times L$ of triples such that for any $i \in \{1..n\}$, $j \in \mathbb{N}_0$, there is at most one $v \in L$ such that $(i, j, v) \in R$. The semantics of this are that a triple (i, j, v) is supposed to represent the information that $X_i^{(j)} = v$. Therefore, we will say that

[8]Formally, we have to allow infinities here because there are of course easily constructible points in the probability space where the journal becomes infinite. Note that we already know from the simple considerations in Section 2.3 that in the case of a satisfiable formula, the running time is infinite with probability zero, so these points form a nullset. And due to the non-constructive version of the Local Lemma, we know that all our input formulas are satisfiable. However, we would here like to present an independent proof which does *not* make use of the non-constructive version of the Local Lemma anymore, implying the existential statement via the algorithmic analysis. To this end, we ignore our previous knowledge and for the time being simply allow for infinite journals.

a reconstruction set R is *true* for a given run of the algorithm if

$$\forall (i, j, v) \in R \ : \ X_i^{(j)} = v.$$

We follow this by a lemma telling us that any given prefix of the algorithm's execution journal allows for the reconstruction of a corresponding portion of the variable histories. We use the common notation F^* to denote the set of all finite strings over the alphabet F (with the clauses as its characters), that is ordered lists of any finite length, the entries of which are clauses from F.

Lemma 2.22. *There exists a reconstruction map*

$$\mathcal{R} : F^* \to 2^{[n] \times \mathbb{N}_0 \times L}$$

such that for all fixed lists of clauses $\mathcal{J} \in F^*$, the following holds. Let $r_i(\mathcal{J})$ be the number of clauses in $\mathcal{J}$ that contain variable x_i, for all $1 \leq i \leq n$. For every execution of the algorithm with journal $\mathcal{C}$ and histories $\{\mathcal{X}_i\}_{1 \leq i \leq n}$, if $\mathcal{J}$ is a prefix of the journal $\mathcal{C}$, then

$$\mathcal{R}(\mathcal{J}) = \{ \ (i, j, X_i^{(j)}) \mid 1 \leq i \leq n, \ 0 \leq j < r_i(\mathcal{J}) \ \}.$$

Note that this means that knowing any prefix $\mathcal{J}$ of the execution journal, we can reconstruct the values of all samples of variable values that were discarded (i.e. replaced with new samples) during the first $|\mathcal{J}|$ correction steps. The only thing which we cannot reconstruct, is the most recent value for each variable, that is, the value it holds after the $|\mathcal{J}|^{th}$ step. This information is not present in the journal.

Proof of Lemma 2.22. The proof is by induction on the length of $\mathcal{J}$. If $\mathcal{J}$ is empty, then $\mathcal{J}$ is always a prefix of the journal. But since $r_i(\mathcal{J}) = 0$ for all i in that case, we can define $\mathcal{R}(\mathcal{J}) = \emptyset$ and satisfy the condition. So much for the base case.

Now suppose the statement is true for all prefixes shorter than $\mathcal{J}$ and conclude that it holds for $\mathcal{J}$. Let $C \in F$ be the last entry in $\mathcal{J}$ and

let $\mathcal{J}'$ be $\mathcal{J}$ with the last entry removed. By virtue of the induction hypothesis, we can reconstruct

$$\mathcal{R}(\mathcal{J}') = \{\ (i, j, X_i^{(j)}) \mid 1 \leq i \leq n,\ 0 \leq j < r_i(\mathcal{J}')\ \},$$

meaning that we already know all samples that have been deleted and replaced up to the correction step numbered $|\mathcal{J}'|$. The only ones we miss in order to complete the induction are the triples $(i, j, X_i^{(j)})$ for $j = r_i(\mathcal{J}')$ and all i such that $x_i \in \mathrm{vbl}(C)$. But those are exactly the values that were held by the variables in the clause C right before the algorithm decided to fix C in the $|\mathcal{J}|^{th}$ correction step. And we know what those values are: if $C = \{(x_{i_1} \neq v_1), \ldots, (x_{i_k} \neq v_k)\}$, then the only reason why the algorithm decided to repair C in the $|\mathcal{J}|^{th}$ step can be because C was violated before that step, which uniquely determines the values of all variables in C at that time. Formally, x_{i_q} must have been assigned the value v_q, for $1 \leq q \leq k$, at that point in time. We conclude that we can adjoin the triples

$$\mathcal{R}(\mathcal{J}) := R(\mathcal{J}') \cup \{(i_q, r_{i_q}(\mathcal{J}), v_q) \mid 1 \leq q \leq k\}$$

and finish the induction. □

We summarize. In order to document the execution of the algorithm, it is enough to write down the journal, that is the sequence of clauses $C_1, C_2, \ldots$ which are selected for correction in each step. The information contained in the first t entries of that list allows for the reconstruction of each variable's history up to (but excluding) the very most recent value of each variable (after the t-th correction step).

Now we will proceed to inspecting ways of how to record the journal and then juxtapose the amount of data this produces to the amount of random data we can so reconstruct.

2.5 Incompressibility[9]

In the present section, we want to warm up for the general proof by proving a weaker version of Theorem 2.18. We are interested in this weaker version because it admits a much more simple and concise proof than the one providing full accuracy. The simpler proof also nicely illustrates what is the main idea in the full proof of Theorem 2.18, namely an information theoretic argument balancing the number of random bits consumed by the algorithm with the number of bits necessary to write down its execution protocol.

The way this simple proof works makes it necessary for us to prescribe, in Algorithm 1, a rule according to which violated clauses are being selected for resampling. Let an arbitrary but fixed *lexicographical ordering* be imposed on the set of clauses. We modify the algorithm as given in Algorithm 2.

First note that despite its more complicated structure, Algorithm 2 is just one incarnation of Algorithm 1. The latter allows for *any* way of choosing a violated clause for resampling in each iteration, while the former prescribes such a rule by means of the recursion used. In particular, the simple argument we have sketched for why this algorithm always terminates, carries over. We now prove the following bound on the expected running time when calling Algorithm 2 on a ClSP formula which is even somewhat sparser than the local condi-

[9]The material in this section is the result of a process of continuous simplification of the proof presented in [Mos09], the refinement of which from [MT10] we will detail in the Section 2.6. After publication, it became apparent that an argument of this type had already been applied to a Lovász Local Lemma type problem in independent work by Schweitzer [Sch09], who presents an almost identical proof, merely missing the realization that this yields an efficient algorithm rather than just an alternative way of reproving known consequences of the Lovász Local Lemma. The term 'incompressibility' in this context is due to Schweitzer.

Significant parts of the present write-up have appeared as [MW12a].

Algorithm 2 `LocalSolverLexicographical`(F)

Require: A satisfiable ClSP formula F.
Ensure: Output is a satisfying assignment.

$\alpha \leftarrow$ a uniformly random assignment from $\mathcal{S}$
while $\exists C \in F : \alpha$ violates F **do**
  $C \leftarrow$ the lexicographically first clause violated by α
  $\alpha \leftarrow$ `LocallyCorrect`(C, α)
end while

function `LocallyCorrect`(C, α) **do**
  for $x \in \mathrm{vbl}(C)$ **do**
    $\alpha(x) \leftarrow$ new uniformly random value from L_x
  end for
  while $\exists D \in \Gamma_F^+(C) : \alpha$ violates F **do**
    $D \leftarrow$ the lexicographically first α-violated clause in $\Gamma_F^+(C)$
    $\alpha \leftarrow$ `LocallyCorrect`(D, α)
  end while
end function

tion requires.

Theorem 2.23. *If $|\Gamma_F(C)| \leq d^{k-3} - 1$ for all clauses C in a (d,k)-ClSP formula F, then Algorithm 2 terminates after at most $\mathcal{O}(|F| \log |F|)$ correction steps on expectation.*

We now set out for a proof of this statement.

Genuine Improvements. The recursive procedure `LocallyCorrect` as defined above tries to correct one violated clause C and then invokes itself recursively in order to make sure that also the neighborhood of C is satisfied. This facilitates the bookkeeping in the proof because whenever the procedure returns, we are sure to have made

genuine progress. We formalize this in the following way.

Lemma 2.24. *Let $\alpha : V_n \to \{1..d\}$ be any assignment and $C \in F$ a clause which is violated under α. Suppose we invoke* `LocallyCorrect`(C, α). *If this procedure ever returns, then the assignment α' it has constructed satisfies all clauses containing a variable whose value changed, i.e.*

$$\forall D \in F \ : \ (\exists x \in \mathrm{vbl}(D) : \alpha(x) \neq \alpha'(x)) \ \to \ \alpha' \text{ satisfies } D$$

and in particular, C is always among these clauses and thus satisfied.

Proof. The proof proceeds by reductio ad absurdum. Let

$$V' := \{x \in V \mid \alpha(x) \neq \alpha'(x)\}$$

and suppose that $D \in F$ is any clause that contains at least one variable from V' and is left unsatisfied after the process ends. Consider the variable $x \in \mathrm{vbl}(D)$ which is the last one to be resampled among all variables occurring in D. This resampling happens inside some recursive call of `LocallyCorrect` for some clause $E \in F$ with $x \in \mathrm{vbl}(E)$. When that recursive call returns, D, being a member of $\Gamma_F^+(E)$, must be satisfied. And ever after, none of the variables occurring in D change their value anymore which is a contradiction. □

From the lemma, we can conclude that if a call to `LocallyCorrect` ever returns, then the set of clauses violated under α' is a strict subset of the set of clauses violated under α, since the clause we initially called the procedure for was violated and is now satisfied, and no new violated clauses can appear. Therefore,

Corollary 2.25. *The outer loop in* `LocalSolverLexicographical` *cannot be repeated any more than $|F|$ times.*

The corollary can be strengthened insofar as the number of top-level invocations cannot be larger than in fact n/k because once one

such invocation returns, not only the clause it has been started for is "fixed", but all variables it contains are "fixed" in the sense that no clause containing any of these variables can ever after appear in a top-level invocation anymore. But a linear bound suffices in general.

It now remains to prove that the total number of recursive calls to `LocallyCorrect` cannot become too large. We do this by showing that if it did become too large, then there would be a way of representing the journal which is more concise than it should be information-theoretically possible.

A bit of information theory. The argument bases on one of the simplest principles in information theory, namely that uniformly random data is not effectively compressible. In order to formalize this intuition, we have to say what it means to effectively compress.

Suppose we fix two integers $t, t' \in \mathbb{N}$ such that $t' < t$. Now we consider a d-ary $(t, t')-$*compression function*

$$\mathfrak{C} : [d]^t \to ([d]^{t'} \cup \{\texttt{'no'}\})$$

that takes as input a string of length t and outputs either a string of shorter length t' or the failure message `'no'` (read: "can't compress this"). Its counterpart is a d-ary (t', t)-*decompression function*

$$\mathfrak{D} : [d]^{t'} \to [d]^t$$

which takes a compressed bit string of length t' and decodes the longer string of length t. What we require for $(\mathfrak{C}, \mathfrak{D})$ to be a *sound compression system* is that whenever $\mathfrak{C}(s) \neq$ 'no' for some $s \in [d]^t$, then $\mathfrak{D}(\mathfrak{C}(s)) = s$.

For practical cases of compression, we would of course not allow the mapping to 'refuse' compressing; instead we would allow for the compressed images of some statistically unlikely data to be larger than the data itself. And we would require many more things, e.g. that $\mathfrak{C}$

and $\mathfrak{D}$ be efficiently computable. Here, such practical aspects are irrelevant, we are only interested in the information theoretic principle.

The following lemma states the simple observation that there is no perfect compression.

Lemma 2.26. *Let $t, t' \in \mathbb{N}$ with $t' < t$ and let $\mathfrak{C}$ and $\mathfrak{D}$ be d-ary (t, t')-compression and (t', t)-decompression functions such that $(\mathfrak{C}, \mathfrak{D})$ is a sound compression system. Now let $S \in [d]^t$ be distributed uniformly at random among all d-ary strings of length t. Then*

$$Pr(\mathfrak{C}(S) = \text{'no'}) \geq 1 - d^{t'-t}.$$

Proof. The proof is by simple counting. There cannot be any two distinct strings $s, s' \in [d]^t$ for which $C(s) = C(s') \neq \texttt{'no'}$ because that would imply $\mathfrak{D}(\mathfrak{C}(s)) = \mathfrak{D}(\mathfrak{C}(s'))$ and thus $\mathfrak{D}(\mathfrak{C}(s)) \neq s$ or $\mathfrak{D}(\mathfrak{C}(s')) \neq s'$. But since there are d^t strings of length t and only $d^{t'}$ strings of length t', at most $d^{t'}$ of the input strings can be mapped to an output other than 'no'. Therefore, if S is chosen u.a.r. from $[d]^t$, the probability that $\mathfrak{C}(S) \neq \texttt{'no'}$ is at most $d^{t'-t}$, as claimed. □

In what follows, we will apply this lemma to bound the expected running time of our algorithm. The idea will be that we decorate the algorithm with a few extra statements that encode the journal of the algorithm in a compact fashion. We already know that from the journal, the random input bits can be largely reconstructed. Thus we prove that *either* the algorithm solves our formula quickly, *or else* it effectively compresses the random bits it has received. The latter can however not happen too often, leaving us with the former we are actually heading for.

Concisely Encoding the Journal. Suppose we want to record, as our algorithm runs, a compact log of which clauses are being corrected at what time. We could write down the index of each clause in the

journal. However, doing this in a trivial way wastes a certain amount of information: if there are $m = |F|$ clauses in total, then we need $\lceil \log_d(m) \rceil$ d-ary characters to represent a clause. But if a clause C is corrected first and then recursively a clause $D \in \Gamma_F^+(C)$, then identifying D should need substantially fewer characters since $\Gamma_F^+(C)$ contains only a small number of clauses.

The extended version of the algorithm, Algorithm 3, makes use of this fact to produce a log in the form of a d-ary string $s \in [d]^*$ documenting its actions. As for notation, if $F' \subseteq F$ is a set of clauses and $C \in F'$ some clause in that set, we denote by $\texttt{daryCode}(C, F')$ the d-ary encoding of the *index of C in F'*, that is the number of clauses in F' that occur before C in the lexicographic ordering, padded with '1'-characters[10] in such a way that the length of each code is $\lceil \log_d(|F'|) \rceil$. Moreover, we use the operator $\circ$ to denote the catenation of strings and we assume that the function `AppendToLog(..)` is some arbitrary means of output on a side-channel.

Whenever a recursive call is made for correction of a clause $D \in \Gamma_F^+(C)$ in the neighbourhood of a previously corrected clause C, we write the index of D in $\Gamma_F^+(C)$ to the log. We prepend this index by a single '2'-character indicating that a deeper recursion level is being created. Let us call this a *message of type I*. Whenever `LocallyCorrect` finds that all clauses in the currently inspected neighbourhood are satisfied and thus returns control to a higher recursion level, we append a single '1'-character to the log to represent this fact. We call this a *message of type II*.

On a global scale, whenever the outermost loop starts a new recursive correction process, we write the index of the clause C corrected on the topmost recursion level to the log. As there is no previous infor-

[10]Note that we are using the characters '1' and '2' as special characters at certain points and as normal digits in encodings of integers at others. Still, decoding is unambiguous.

mation available as to which clause this could be, we have to use a full encoding of C in F. We call this a *message of type III*. As the recursion depth before the top-level call can be reconstructed by counting previous messages of types I and II, no extra character is needed to indicate the type of message appended. Note that the top-level call, too, will automatically output a '1'-character in the end to indicate termination.

Algorithm 3 `LocalSolverLexicographicalAnnotated`(F)

Require: A satisfiable ClSP formula F.
Ensure: Output is a satisfying assignment and a log.

$\alpha \leftarrow$ a uniformly random assignment from $\mathcal{S}$
while $\exists C \in F : \alpha$ violates F **do**
  $C \leftarrow$ the lexicographically first clause violated by α
  `AppendToLog(daryCode(C,F));`
  $\alpha \leftarrow$ `LocallyCorrect`(C, α)
end while

function `LocallyCorrect`(C, α) **do**
  for $x \in \mathrm{vbl}(C)$ **do**
    $\alpha(x) \leftarrow$ new uniformly random value from L_x
  end for
  while $\exists D \in \Gamma_F^+(C) : \alpha$ violates F **do**
    $D \leftarrow$ the lexicographically first clause in $\Gamma_F^+(C)$ violated by α
    `AppendToLog('1'` $\circ$ `daryCode(D,` $\Gamma_F^+(C)$`))`
    $\alpha \leftarrow$ `LocallyCorrect`(D, α)
    `AppendToLog('0')`
  end while
end function

Let us precisely quantify the number of digits that are needed to store the log being output. A message of type I needs $\lceil \log_d(d^{k-3}) \rceil = k - 3$ characters for encoding the clause to be corrected next and one

additional bit being prepended. A message of type II needs exactly one bit. A message of type III needs $\lceil \log_d m \rceil$ digits. In order to facilitate the calculation, let us note that messages of type II always occur paired up with messages of type I or III, which are uniquely associated with a specific invocation of the local correction procedure. Therefore each top-level invocation incurs a total of $\lceil \log_d m \rceil + 1$ digits of output, each recursive invocation then produces an additional $(k-3)+2 = k-1$ characters of output to the log.

Suppose the algorithm makes a total of t invocations of the correction procedure, including the top-level calls (of which a maximum of m can be done according to Lemma 2.24). Then the size of the log we produce is smaller than $m(\lceil \log_d m \rceil + 1) + (k-1)t$. Note that we are very generous here and the top-level calls are being overcounted.

It is very easy to see that from the log output, it is possible to unambiguously reconstruct the entire history of clause corrections. In turn, from this history, as we know from Lemma 2.22, it is possible to reconstruct for each variable $x \in V$, *all* the values it has received during the procedure, *except* for its final value as our reconstruction always lags behind by one.

We now claim the following.

Lemma 2.27. *For any $t \in \mathbb{N}$, the probability that the algorithm does not terminate before making t recursive invocations of the local correction procedure is at most $d^{m(\lceil \log_d m \rceil + 1) - t}$.*

Given this bound, then using Lemma A.6 with $C := d^{m(\lceil \log_d m \rceil + 1)}$ and $\epsilon := 1 - 1/d$, we obtain that the average number of recursive invocations cannot be any larger than

$$\frac{1}{1-1/d} \cdot (\ln d \cdot m(\lceil \log_d m \rceil + 1) + 1) = \mathcal{O}(m \log_d m),$$

concluding the proof of the theorem.

Proof of Lemma 2.27. Fix some $t \in \mathbb{N}$. We incorporate our algorithm into a compression scheme consisting of functions

$$\mathfrak{C} : [d]^{n+tk} \to ([d]^{n+m(\lceil \log_d m \rceil + 1)+t(k-1)} \cup \{\texttt{'no'}\})$$

and

$$\mathfrak{D} : [d]^{n+m(\lceil \log_d m \rceil + 1)+t(k-1)} \to [d]^{n+tk}.$$

The compression $\mathfrak{C}$ of a string $s \in [d]^{n+tk}$ is defined as follows: run Algorithm 3, using s to replace the randomness, i.e. the first n characters of s are used as the initial assignment and the remaining digits are used, in chunks of k each, for replacing values within the local correction procedure. We let the algorithm go on for up to t calls to $\texttt{LocallyCorrect}(C)$. If it has not finished its job by then, we interrupt it and make $\mathfrak{C}$ output the log produced so far, which we know has size at most $m(\lceil \log_d m \rceil + 1) + t(k-1)$, followed by n digits representing the current value of α when the algorithm was interrupted. If necessary, we can pad the string with zeroes for it to have the required length. If, however, the algorithm is successful in finding a satisfying assignment before reaching t invocations of the correction procedure, then we let $\mathfrak{C}$ output $\texttt{'no'}$. This way, *either* the solver is successful *or* the compression is.

The decompression $\mathfrak{D}$ does the reverse. As we have described above, each of the t calls to $\texttt{LocallyCorrect}(C)$ we have recorded in our log allows for the reconstruction of k determined positions from s. As further described, the only characters we have not reconstructed this way are the final values of each variable in α. But since $C(s)$ has the n bits describing α appended to its end, we know these values too. Finally, we have reconstructed all $n + tk$ bits from s correctly and thus whenever $C(s) \neq \texttt{'no'}$, we have $D(C(s)) = s$.

Since this is a compression scheme satisfying exactly the conditions in Lemma 2.26, the probability that $C(s) = \texttt{'no'}$ if s is chosen uniformly at random has to be at least $1 - d^{m(\lceil \log_d m \rceil + 1)-t}$. But those are exactly

the cases when a satisfying assignment is output, concluding the proof of Lemma 2.27. □

More Concise Logging. Theorem 2.23 can be strengthened in various ways.

Firstly, note that we have used d-ary characters for representing the log output and in particular we have used d-ary characters for the recursion level opening and closing indicators. Those are in reality binary bits, so we could prescribe that at the corresponding positions, only a binary bit is placed and read. This allows for neighborhood sizes of $d^k/8 - 1$ instead of $d^{k-3} - 1$.

Secondly, one can strengthen the proof so as to show that there is only a linear number of recursive invocations on average. To see this, one has to modify the way top-level invocations are being journaled. Instead of posting a message of type III each time this happens, we can withhold all these messages from the log and instead prepend to the very beginning of the whole log a bitstring of length m in which every bit corresponds to a clause, arranged in the lexicographic ordering. If the bit is set, this means that for the corresponding clause, a top-level invocation is being made in this run of the algorithm. If the bit is cleared, it means that there is no such invocation for this clause. All information about the algorithm's execution is still reconstructible from this log because each time a top-level invocation completes, we can scan the prefix string until the next admissible clause has a set bit, then we continue decoding the recursive main body of the log. While this logging format has lost the beautiful chronological one-pass layout, it is even more compact than the standard one: only $\mathcal{O}(m)$ bits are needed for the top-level invocations and now the information-theoretic balance starts being lopsided after only linearly many recursive invocations.

We have proved the following stronger version.

Theorem 2.28. *If $|\Gamma_F(C)| \leq d^k/8 - 1$ for all clauses C in a (d,k)-ClSP formula F, then Algorithm 2 terminates after at most $\mathcal{O}(|F|)$ correction steps on expectation.*

Inhomogeneous Clause Sizes and Prefix-Free Addressing. Another strengthening allows this proof to cater for formulas where clauses have inhomogeneous sizes. What we then get is the following just slightly weaker constructive counterpart of Theorem 2.5.

Theorem 2.29. *If in a non-degenerate $(\geq 2, \leq d)$-ClSP F, for all clauses $C \in F$ we have*

$$\sum_{D \in \Gamma_F^+(C)} w(D) \leq \frac{1}{d^3},$$

then Algorithm 2 terminates on input F after at most $\mathcal{O}(|F|)$ correction steps on expectation.

One can strengthen the theorem even further to allow neighborhood weights of $1/8$ instead of $1/d^3$ removing the dependency on d. This is using similar tricks as outlined in the symmetric case above. One then obtains the following.

Theorem 2.30. *If in a non-degenerate ClSP F, for all clauses $C \in F$ we have*

$$\sum_{D \in \Gamma_F^+(C)} w(D) \leq \frac{1}{8},$$

then Algorithm 2 terminates on input F after at most $\mathcal{O}(|F|)$ correction steps on expectation.

We will however only detail a proof of Theorem 2.29. The stronger version requires too many unaesthetic details to justify us going into it here. Note that, of course, Theorem 2.30 follows from Theorem 2.5 via Corollary 2.20 anyways, so we will prove it along with the fully

general version in the next section so soon as we leave string incompressibility.

The generalization which leads to the proof of Theorem 2.29 is fairly straightforward. In this version, we are dealing with a situation where "heavy" clauses with few literals which have a high likelihood to be violated and therefore tend to appear in journals more often cannot be too numerous in any neighborhood, while "light" clauses which are long and thence occur only rarely can appear in larger numbers as the neighbors of a given clause. We need to find a way of representing the clauses in such neighborhoods in such a way that clauses occurring often have a short representation and those which occur rarely a longer one.

This can be done by means of variable-length prefix-free codes. The very well-known theorem of Leon G. Kraft provides necessary and sufficient conditions for a prefix-free code of a certain shape to exist. Recall that a code $\mathcal{Z} \subseteq \{1..d\}^*$ is called *prefix-free* if there are no two distinct codewords z and z' in $\mathcal{Z}$ such that z is a prefix of z'.

Theorem 2.31 ([Kra49])**.** *Let $d_1, d_2, \ldots, d_r \in \mathbb{N}$. A prefix-free code $\mathcal{Z} \subseteq [d]^*$ with r codewords $\mathcal{Z} = \{z_1, z_2, \ldots, z_r\}$ such that $|z_i| = d_i$ for $1 \leq i \leq r$ exists if and only if*

$$\sum_{i=1}^{r} d^{-d_i} \leq 1.$$

Moreover, such a code can be generated efficiently (in time linear in its size).

The algorithm and proof are both very simple. One can partition the numbers greedily into d parts in such a way that each part contains at most $1/d$-fraction of the weight. Build smaller codes for each part recursively, then prefix the words in the i^{th} category with character i, for all $1 \leq i \leq d$.

We can now use prefix-free codes for representing the clauses chosen for correction in recursive invocations of our procedure. For every

clause $C \in F$, fix an arbitrary prefix-free code $\mathcal{Z}_C$ with the property that if $\Gamma_F^+(C) = \{D_1, D_2, \ldots, D_r\}$, then the lengths of the words in $\mathcal{Z}_C$ are

$$d_i := |D_i| - 3 \quad \text{for } 1 \leq i \leq r.$$

We can easily check that such a code exists because the hypothesis entails that

$$\sum_{i=1}^{r} d^{-d_i} = \sum_{i=1}^{r} d^{-|D_i|+3} = d^3 \cdot \sum_{D \in \Gamma_F^+(C)} w(C) \leq 1,$$

hence existence follows from Theorem 2.31.

We now imagine repeating the proof of Theorem 2.23, but in the annotations, we replace `daryCode`(D, $\Gamma_F^+(C)$) by some encoding, let us call it `prefixFreeCode`(D, C), which outputs the codeword corresponding to D within the precomputed prefix-free code Z_C. This being the only change, the information theoretic balance works again as expected: each time a message of type I/II indicating recursion from C into D is posted to the log, this needs $|D| - 3$ characters for the codeword from Z_C, then a leading and a later trailing character to delimit the recursion level. This incurs a total of $|D| - 1$ characters posted to the log. But the knowledge that D had to be fixed allows for the reconstruction of $|D|$ values from the variables histories, outweighing the encoding by exactly one character. Since we save this one character per every recursive invocation, the information-theoretic balance goes lopsided after a linear number of invocations.

A slight technical problem will arise when fixing the number of invocations done before we surrender and when trying to invoke our information theoretic Lemma 2.26. If the number of invocations is fixed, the length of the log and the number of reconstructible characters are both random variables because these depend on the length of the clauses fixed in those invocations, a situation to which Lemma 2.26 is not tailored. However, this problem is easy to solve. Instead of prescribing a fixed number of invocations before we give up, prescribe a fixed

number of random characters to be used. Once this number is exhausted, give up. If we do it this way, the number of reconstructible characters is fixed (up to negligible divisibility issues). Also, the number of characters needed for the log can be bounded by a fixed number using that every clause cannot be larger than n literals and thus, every n characters at the latest, we information-theoretically save one character.

2.6 Witness Trees[11]

We now want to go on to prove Theorem 2.18 in full generality. To my current knowledge, this cannot be done using the method outlined in the previous section or a mild variant of it[12] – although the philosophy of the proof detailed hereafter is of course the same. The idea of representing bad executions of the algorithm by a journal in the form of a string of bits or other characters has its inherent limits. Instead, we will now go on to express the journal - or rather parts of the journal - by trees.

Note that now we are back to Algorithm 1 in its generic form. We do not specify a rule as to which violated clause to select for correction if there are multiple to choose from. Still, let us now fix an arbitrary rule so that our random experiment is well-defined. And then let N be the random variable denoting the number of steps Algorithm 1 car-

[11] All original material in this section is joint work with Gábor Tardos and appeared in [MT10]. The write-up as presented here has appeared in significant parts in [MW11].

[12] There has been an attempt at making the incompressibility proof as strong as possible by Messner and Thierauf [MT11]. However, their proof strategy uses witness trees as well, so we do not consider this a 'pure' encoding proof. We note that, of course, *any* proof based on counting can be converted into a proof based on encoding because the two concepts are combinatorically equivalent. If however a detour via witness trees is necessary anyways, then the beautiful simplicity of what we sketched in the previous section is to a large degree lost.

ries out until a satisfying assignment is found. We know that N is finite with probability one. We now claim that $\mathbb{E}[N]$ satisfies the bound claimed in Theorem 2.18.

Witness Tree Extraction. Indeed, the journal representation we previously chose allows for the reconstruction of all correction steps carried out and from this, in turn, the reconstruction of the variable histories. But in reality, even less information is sufficient to achieve this goal, and in order to get an intuition for this fact let us look at *some* information contained in $\mathcal{C}$ that is *not* necessary for our reconstruction of histories. Suppose there are two clauses A and B in F which do not have any variables in common, i.e. $\{A, B\}$ is a non-edge in G_F. Suppose further that our journal contains the consecutive entries $\ldots A, B \ldots$ somewhere. In the spirit of the proof of Lemma 2.22, both entries allow for the reconstruction of some values in the histories of the variables in $\mathrm{vbl}(A)$ and $\mathrm{vbl}(B)$. But the journal also contains the information that A was corrected *before* B. This does not matter. So we have to find objects that retain sequencing information for clauses that depend on each other and discard it for those that do not.

To that end, let $t \in \mathbb{N}$ and consider the t^{th} correction step of the algorithm. Why was the clause C_t violated immediately before the t^{th} step? Well, each of the variables that occur in C_t received their most recent values (which violate C_t) at some point in the procedure: some of them may still have their initial values, others may have been resampled a number of times. For each of the variables $x_i \in \mathrm{vbl}(C_t)$, we can point either to the initial assignment or to some time index $t' < t$ such that the sample generated at random for x_i at that time step t' evaluated exactly the way that makes C_t unhappy. Suppose x_i was resampled at time t' for the last time. We can ask again: why was it necessary to resample x_i in the t'^{th} step? Well, the reason was because immediately before step t', some other clause $C_{t'}$ was violated, so we had to fix it, and x_i occurs in $C_{t'}$. Again we can ask: why was

$C_{t'}$ violated back then? Where did the samples for variables in $C_{t'}$ originate from and in turn we can point either to the initial assignment or to some even earlier time step $t'' < t'$ when these samples were drawn. If we iterate this process of repeatedly asking for justification, we will generate a 'tree of reasons' which we will now formalize.

A *witness tree* T (for a fixed formula F) is a finite (!) rooted tree together with a mapping $[\cdot] : V(T) \to F$ that labels each node in T by some clause of our ClSP, satisfying the following properties:

(i) if for $u, v \in V(T)$, u is the parent node of v, then $[u]$ and $[v]$ share at least one variable, i.e. whenever $\{u, v\}$ is an edge in T, then $\{[u], [v]\}$ is an edge in G_F, and

(ii) if u and u' are both children of the same parent node v, then their labels are distinct, $[u] \neq [u']$.

We denote by $\mathcal{T}$ the set of all witness trees (for the fixed formula F) which satisfy properties (i) and (ii). And by $\mathcal{T}_A$ for $A \in F$, we denote the set of all $T \in \mathcal{T}$ of which the root node is labelled by A.

We now define particular witness trees associated with an execution of our algorithm. We generate one witness tree per correction step which the algorithm makes. That is, for any $t \in \mathbb{N}$ we define a witness tree T_t based on the first t entries $\mathcal{C}_t = \langle C_1, C_2, \ldots, C_t \rangle$ of the journal. Note that of course, since $\mathcal{C}_t$ is a random variable, T_t is a random variable, too, which induces some probability distribution on the set $\mathcal{T}$ of all possible witness trees. For $t > N$, T_t is undefined.

We build the witness tree T_t in the following backward inductive fashion.

(1) let $T_t^{(t)}$ be a single root vertex labelled C_t

(2) for $j = t-1, t-2, \ldots, 1$, repeat the following:

(2.1) check if there is a node $v \in V(T_t^{(j+1)})$ such that $[v]$ has some variables in common with C_j.

(2.2) if yes, choose among all such nodes a node v^* of largest depth (if there are multiple deepest ones, choose *any* one of them). Then create a new node v' as a child to v^* and label it $[v'] := C_j$. Let $T_t^{(j)}$ be the tree so obtained.

(2.2) if no, skip this step and let $T_t^{(j)} := T_t^{(j+1)}$

(3) let $T_t := T_t^{(1)}$

From the definition, it is immediate that T_t is a witness tree. Additionally, the witness trees $T_1, T_2, \ldots, T_N$ created by the above rules satisfy some extra properties that we want to investigate. For any T_i and any node $u \in V(T_i)$ let us say that *u represents* correction step $j \leq i$ if u was added when scanning C_j and thus labelled $[u] = C_j$, that is j is the largest number such that $u \in V(T_i^{(j)})$. It has to be distinctly understood that the information which correction step is represented by a node is *not* part of the witness tree, it can only be obtained if $\mathcal{C}_t$ is additionally known.

We henceforth call a witness tree T *proper*, if for any two nodes $u, v \in V(T)$ at the same depth, that is $d(u) = d(v)$, $[u]$ and $[v]$ do not share any variables, i.e. $\{[u], [v]\}$ is a non-edge in G_F.

Lemma 2.32. *For any run of the algorithm, the following holds.*

(i) For all $1 \leq i \leq N$, T_i is a proper witness tree.

(ii) Let $1 \leq i \leq N$ and $u \in V(T_i)$ be any node and $j \leq i$ the correction step that u represents. Let $x \in \mathrm{vbl}([u])$ be any variable resampled in that step. Then the number of times x was resampled before step j, i.e. the number of times x appears in the clauses $C_1, C_2, \ldots, C_{j-1}$, is the number of nodes $v \in V(T_i)$ with $d(v) > d(u)$ and $x \in \mathrm{vbl}([v])$.

(iii) For any $1 \leq i < j \leq N$, $T_i \neq T_j$.

Proof. For (i), suppose there were two nodes u and v at the same depth that did share a variable. Without loss of generality, u was added before v during the backward scan of the journal. But then, by the rule that every node is attached to the deepest node its label shares a variable with, v would have been attached as a child to u or to an even deeper node, which is a contradiction.

For (ii), since u represents correction step j, during the construction of T_i, u is born in the step for $T_i^{(j)}$. After that, the backward scan will look at $C_{j-1}, C_{j-2}, \ldots, C_1$ and each time a clause C_k is encountered that contains x, a node u' labelled C_k will be added to the tree. As this node will be attached to the deepest node of $T_i^{(k+1)}$ sharing any variables with C_k and since the node $u \in T_i^{(k+1)}$ *is* sharing variables with C_k, the new node will definitely have a larger depth than u, $d(u') > d(u)$. The number of occurrences of x in the prefix of length $j-1$ or the journal is therefore *at most* as large as the number of nodes at depths larger than u whose label contains x.

Conversely, if there were *any* additional node u'' whose label contained x and which occurred at depth larger than u in T_i, then that node *must* represent a correction step $k < j$. For suppose it represented a step $k > j$, then the node u'' already existed in $T_i^{(j+1)}$, and then the addition of u would have happened either as a child to u'' or as a child to an even deeper node, contradicting the fact that $d(u'') > d(u)$. So, such additional nodes cannot exist and therefore the number of occurrences of x in the prefix of length $j-1$ of the journal is also *at least* as large as the number of nodes at depths larger than u whose label contains x.

For (iii), suppose that $T_i = T_j$ for $i < j$, then their root is labelled $C_i = C_j$, implying that T_i contains one node labelled C_i per occurrence of $C_i = C_j$ in $\mathcal{C}_i$, while T_j contains one node labelled C_j per occurrence of $C_i = C_j$ in $\mathcal{C}_j$ and since $\mathcal{C}_j$ has at least one more occurrence of C_j, T_j must contain at least one more node labelled C_j than T_i has, a contradiction. So $T_i \neq T_j$ for all $1 \leq i < j \leq N$. □

Reconstruction from Trees. As indicated above, a witness tree gives a justification for the fact that the last correction step (at the root of the witness tree) had to be carried out. And it somehow comprises all the information necessary to understand what sequence of events lead to that last correction. It thereby only retains the amount of information that is really necessary: if the labels $[u]$ and $[v]$ of two nodes are connected by an edge in G_F, i.e. if they have some variables in common, then they are at different depths in the tree and we can tell which of the steps associated with them was carried out before the other. If, on the other hand, u and v are at the same depth in the tree, we cannot tell which was added first, but neither do we need to know, as they are independent from one another.

The following lemma formalizes this by saying that by just looking at some witness tree T_t obtained in this way, we can reconstruct large portions of the histories of all variables involved in the sequence of corrections represented by T_t. It is an immediate analogue and in certain ways a strengthening of Lemma 2.22.

Lemma 2.33. *There is a reconstruction map*

$$\mathcal{R}' : \mathcal{T} \to 2^{\{1..n\} \times \mathbb{N}_0 \times L}$$

such that for all fixed witness trees $T \in \mathcal{T}$, the following holds. Let $r_i(T)$ be the number of labels in T that contain variable x_i, for all $1 \leq i \leq n$. If $T = T_t$ for some $t \in \mathbb{N}$, then

$$\mathcal{R}'(T) = \{\ (i, j, X_i^{(j)}) \mid 1 \leq i \leq n,\ 0 \leq j < r_i(T)\ \}.$$

Proof. We will reuse Lemma 2.22. Let $\mathcal{J} \in F^*$ be the sequence of clauses (the 'artificial journal') that we obtain if we enumerate the labels of all vertices in T in a level-by-level fashion, starting at the deepest nodes, e.g. like in a reverse BFS. Now we define $\mathcal{R}'(T) := \mathcal{R}(\mathcal{J})$. The claim is that now, if $T = T_t$ for some $t \in \mathbb{N}$, then $\mathcal{R}'(T)$ is a true reconstruction set.

In order to see this, fix t such that $T = T_t$. We go back to the proof of Lemma 2.22 and recall how the reconstruction $\mathcal{R}$ works. It reads the journal $\mathcal{J}$ entry by entry and for the i^{th} entry

$$J_i = \{(x_{i_1} \neq v_1), (x_{i_2} \neq v_2), \ldots, (x_{i_k} \neq v_k)\},$$

it adds the triples

$$\{(i_q, r_{i_q}, v_q) \mid 1 \leq q \leq k\}$$

to the reconstruction set, where r_{i_q} here denotes the number of times the variable x_{i_q} has appeared in $\mathcal{J}$ before J_i. The node u_i from which the entry J_i was produced during the reverse BFS represents some correction step $C_j = J_i$, during which all the variables $x_{i_1}, \ldots, x_{i_k}$ were resampled, because they previously had the values $v_1, \ldots, v_k$, which violated C_j. And by virtue of Lemma 2.32, statement (ii), we know that the number of times any x_{i_q} had been resampled before the j^{th} step equals the number of nodes u' of depth larger than $d(u_i)$ such that $x_{i_q} \in [u']$ and thus also the number r_{i_q} of times x_{i_q} appears in a clause listed in $\mathcal{J}$ before J_i. Therefore,

$$X_{i_q}^{(r_{i_q})} = v_q, \quad 1 \leq q \leq k$$

and thus all the reconstructed triples are true. □

We summarize. Looking at any proper witness tree T_t that is constructed from the first t entries in the execution journal $\mathcal{C}_t$, we can reconstruct one value of an independent random sample for every occurrence of a literal in the labels of T_t.

A valid question which should arise at this point is whether with the concept of witness trees we have now found a 'most concise' representation of the information we are seeking to encode. As it will turn out, this level of conciseness is what is necessary in order to prove Theorem 2.4 in full generality. However, the representation is obviously *not* ultimatively concise. For example, in building a witness tree, we

sometimes had a choice of where to attach a new child if there were several nodes at the same depth sharing variables with the clause to be attached. This type of choice makes for inefficiencies in the representation and so one can ask what happens if they are being removed. And indeed, removing such inefficiencies leads to even strongers versions of the Local Lemma as for example Kolipaka and Szegedy have investigated [KS11]. We will very briefly look at such strengthenings in Section 2.10, while we now go on heading for Theorem 2.4.

Bounding the Running Time. We will now bound the expected time the algorithm takes, that is the number of correction steps done on average, using the witness trees we constructed before. We are interested in the probability that a fixed witness tree occurs at some point in the journal.

For notation, let us extend the notion of *weight* and *μ-weight* to witness trees. For any fixed tree $T \in \mathcal{T}$, we define

$$w(T) := \prod_{u \in V(T)} w([u])$$

and

$$w_\mu(T) := \prod_{v \in V(T)} \left(\mu([v]) \prod_{C \in \Gamma_F([v])} (1 - \mu(C)) \right).$$

We go on to derive that the probability of any fixed tree of occurring in the journal is no larger than its weight.

Lemma 2.34. *Let $T \in \mathcal{T}$ be any fixed witness tree. For the probability that T occurs at some place in the algorithm's journal, we have*

$$Pr\left[\exists t \in \mathbb{N} : T_t = T\right] \;\leq\; w(T).$$

Proof. As we have seen in Lemma 2.33, $\exists t : T_t = T$ implies that $\mathcal{R}'(T)$ is a true reconstruction set. So

$$\Pr\left[\exists t \in \mathbb{N} : T_t = T\right] \;\leq\; \Pr\left[\forall (i,j,v) \in \mathcal{R}'(T) \;:\; X_i^{(j)} = v\right].$$

As we noted in the very beginning, $X_i^{(j)} \in L_i$ is distributed uniformly, so the probability that it has exactly the prescribed value v is $1/|L_i|$. Moreover, all these samples are independent from one another (recall that a reconstruction set has at most one triple featuring the same i and j), thus

$$\Pr\left[\exists t \in \mathbb{N} : T_t = T\right] \;\leq \prod_{(i,j,v)\in\mathcal{R}'(T)} \frac{1}{|L_i|}.$$

Going back to the proof of Lemma 2.33, we see that $\mathcal{R}'(T)$ contains exactly one triple (i, j, v) for every literal '$x_i \neq v$' occurring *somewhere* at a node in T. This establishes the lemma. □

Now, we can also compute the expected running time of our algorithm. What we will find is that the following holds. Let $\mathcal{T}_A$ be the set of all witness trees whose root carries the label A.

Lemma 2.35. *For the expected number of times A appears in a journal of the algorithm, we have*

$$\mathbb{E}[|\{t \in \mathbb{N} : C_t = A\}|] \leq \sum_{T\in\mathcal{T}_A} w(T) \leq \frac{\mu(A)}{1-\mu(A)}.$$

This lemma then readily implies the Theorem 2.18. The first inequality is simple: we observe that the sequence $T_1, T_2, \ldots, T_N$ as constructed above features N witness trees which are pairwise distinct according to Lemma 2.32, and each time $C_t = A$ we also have $T_t \in \mathcal{T}_A$. We now proceed to proving the second inequality of Lemma 2.35.

Growing a Witness at Random. Let $\mathcal{T}_A$ be the set of all witness trees for the formula F whose root is labelled with a fixed clause $A \in F$. Our goal is to bound the sum of the weights of all trees in $\mathcal{T}_A$. We shall derive this bound by considering a probability distribution on $\mathcal{T}_A$.

Consider the following process, let us call it $P(A)$, that grows a random witness tree $T \in \mathcal{T}_A$ (or, potentially, an infinite tree) and therefore

induces a probability distribution on $\mathcal{T}_A$ (and the infinite trees, which, however, we will not be interested in).

(1) start with a single root labelled A

(2) while there exists a leaf node v not marked 'done' yet

  (2.1) for each $C \in \Gamma_F^+([v])$ independently, do

    (2.1.1) with probability $\mu(C)$, create a new child v' of v and label it $[v'] := C$, with probability $1 - \mu(C)$ do not do anything

  (2.2) mark vertex v as 'done'

(3) once all vertices are marked 'done', output T

Such a process is known as a *Galton-Watson branching process*. Francis Galton and Henry William Watson were mathematicians in Victorian Britain who analysed reproductive processes where one tracks generations of a population. In our case, the generations are the levels of the witness tree we are building and a snapshot of the population consists of the multiset of clauses at the labels at a given depth. Each clause then produces a series of offspring, its children in the tree, according to certain probabilities. The characterising trait of the process is that each clause of a given generation produces offspring independently of the other members of the population at that time. There is a vast body of literature on Galton-Watson processes, but for our purpose an elementary investigation will suffice.

We now compute the probability distribution this process induces, that is for every tree $T \in \mathcal{T}_A$, we determine the probability that $P(A)$ outputs T. We obtain the following bound.

Lemma 2.36. *For any fixed $A \in F$ and any fixed $T \in \mathcal{T}_A$, we have*

$$\Pr[P(A) \text{ outputs } T] \;=\; \frac{1-\mu(A)}{\mu(A)} \;\cdot\; w_\mu(T).$$

The proof is just a matter of sufficiently many times rearranging terms and can be found in Appendix A.9.

Since $P(A)$ induces a probability distribution on $\mathcal{T}_A$ (and certain infinite trees we are not interested in), and these probabilities cannot sum up to more than one, we obtain a bound on the sum of μ-weights of *all* witness trees with label A at the root.

Corollary 2.37. *We have*

$$\sum_{T \in \mathcal{T}_A} w_\mu(T) \leq \frac{\mu(A)}{1 - \mu(A)}.$$

We can therefore now finish the proof of Lemma 2.35 and thus of Theorem 2.18: since the local hypothesis (and this is the only place we use it) entails that each tree has a smaller weight than μ-weight, Corollary 2.37 readily implies Lemma 2.35. □

Beyond Clause Satisfaction. Our last task in this section is to generalize the argument from ClSPs to arbitrary CSPs, i.e. to establish Theorem 2.21. This is an easy thing however. Going through the proof, we notice that there is only one single point where the argument does not go through effortlessly, namely Lemma 2.33, where we demonstrated that if a witness tree is given which occurs in the journal, then by recovering an artificial partial journal, a certain number of random samples can be reconstructed. We have used this to bound the probability with which a witness tree can occur in the journal.

Now that we are talking about CSPs, reconstructing an artificial partial journal does not suffice in order to reconstruct variable histories. Depending on the type of a constraint C, there can many, not just one, combinations of values of the variables $\mathrm{vbl}(C)$ which lead to C being violated and thus to the possibility of picking C for resampling. From the mere knowledge that C was resampled at some point we can not yet infer all values of $\mathrm{vbl}(C)$ at that particular point in time.

There is an easy fix though. Knowing that C had to be resampled at some point *does* give us *some* information. We can now store all additional information necessary for the value reconstruction inside the witness tree. Indeed, let $u : F \to \mathbb{N}$ be a map which counts the number of value tuples which constraints in F forbid, that is for each $C \in F$, $u(C)$ is such that

$$w(C) \;=\; u(C) \cdot \prod_{x \in \mathbf{vbl}(C)} |L_x|^{-1}$$

holds. Knowing that C had to be resampled in the t-th step tells us that at that time, the assignment to the variables in $\mathrm{vbl}(C)$ was *one* out of those $u(C)$ tuples. If we decorate witness trees with numbers at the vertices telling us *which* of those tuples actually occurred, full reconstruction is again possible.

To be formal, let a *decorated witness tree* be a witness tree T together with a mapping $u_T : V(T) \to \mathbb{N}$ such that for all $v \in u_T$, $u_T(v) \in \{1..u([v])\}$ holds. Building a decorated witness tree T_t for the t-th correction step works as usual, with additionally recording for each constraint correction that is being attached as a new vertex v, the tuple of values which the variables in $[v]$ where holding before the step we are representing by v. The tuple of values is stored as its index according to some lexicographical ordering of all the $r([v])$ tuples that can make C_t unhappy. It is obvious that for such a decorated tree, the equivalent of Lemma 2.33 holds again.

The remainder of the argument works as we have done it for ClSPs. When summing the weights of all witness tress in $\mathcal{T}_A$, we can imagine that we are actually summing over all decorated trees instead: each $T \in \mathcal{T}_A$ represents all possible decorations of T. The number of decorated trees an undecorated tree stands for is given by the *product* of all the $u([v])$ for $v \in V(T)$. On the other hand, the probability that a fixed decorated tree occurs in the journal is the (usual) weight $w(T)$, *divided* by the product of all the $u([v])$ because if we condition on a constraint

C being violated by uniform random samples, then the probability that a specific violating tuple of values occurs is exactly $1/u([v])$. As these two effects cancel nicely, the rest of the calculation can stay as is, establishing Theorem 2.21.

One can apply the algorithm to even *more* general settings by further abstraction. Whenever it is possible to *somehow* store a current evaluation of variables, to efficiently find a violated constraint and finally to efficiently resample all values underlying a constraint, the outlined technique is most likely to apply.

2.7 Slacked Hypotheses

Hereafter, we will be concerned with parallelizing and derandomizing Algorithm 1. It will be possible to do both almost perfectly, with only mild additional assumptions. Those additional assumptions will come in the form of *slack* which will be necessary in the local condition. So instead of asking the clauses or constraints in the problem to meet the requirements of the local lemma *just*, we instead ask them to meet the requirement generously, with a certain amount of leeway.

We will consider two forms of slacked local condition. One where the slack is a multiplicative factor and one where it climbs into the exponent.

In the present section, we will not only define the two types of slack, but we are particularly interested in the question how strong the additional assumptions are in the case of clause satisfaction problems and in how far they change the symmetric and asymmetric simplified criteria we might want to use. Furthermore, we establish that asking for slack does not influence the possibility to bound μ-values from above.

Hypothesis with Multiplicative Slack. Let us say that a given ClSP F *meets the local hypothesis with multiplicative slack* ϵ if there exists some association $\mu : F \to (0,1)$ such that for all $C \in F$, we have

$$w(C) \leq (1-\epsilon) \cdot \mu(C) \cdot \prod_{D \in \Gamma_F(C)} (1-\mu(D)).$$

Under normal circumstances, this requirement is mild. Let us look at simplified versions analogous to Theorem 2.3 and Theorem 2.5 to see what it is that we need to require. It turns out that the symmetric version is more intricate because we have aimed for the very best possible[13] constant $1/e$ so that the calculation is already relatively tight.

Lemma 2.38. *If F is a (d,k)-ClSP, $d,k \geq 2$, and $\delta \in [0,1]$ is such that for all $C \in F$, we have*

$$|\Gamma_F(C)| \leq (1-\delta) \cdot \left(\frac{d^k}{e} - 1 \right),$$

then F satisfies the local hypothesis with multiplicative slack

$$\epsilon = \frac{1}{2} \cdot \left(2^\delta - 1 + \frac{e}{d^k} \right).$$

For our asymmetric interpretation on the other hand, there is some inherent slack which we can exploit. Recall that a *star* is a graph G in which there exists a center vertex v to which all other vertices are connected, while all vertices other than v form an independent set. Call a ClSP F a *star formula* if its dependency graph is a star. Star formulas which are non-degenerate are satisfiable since one can pick any variable from the center clause and assign it in any way which satisfies the center clause and then the remaining star falls apart into an independent set of non-empty clauses each of which can be separately satisfied.

[13] See discussion of tightness in Section 2.10.

Lemma 2.39. *There exists a universal constant $\epsilon > 0$ such that the following holds. If F is a non-degenerate ClSP such that for all $C \in F$, we have*

$$\sum_{D \in \Gamma_F(C)} w(D) \leq \frac{1}{4},$$

then F can be split into parts $F = F_0 \dot\cup F_1$ such that F_0 and F_1 are independent, F_0 consists of independent star formulas, and F_1 satisfies the local hypothesis with multiplicative slack ϵ.

The proofs are to be found in Appendix A.10 and Appendix A.11. For more general settings, let us also check whether multiplicative slack jeopardizes keeping μ-values bounded away from one. Indeed, if we can simply allow the multiplicative slack to drop by a little bit, then bounding the μ-values away from one is no problem.

Lemma 2.40. *There exists a universal constant Λ with the following property. If F is a non-degenerate ClSP satisfying the local condition with slack $\epsilon \in [0,1]$ and mapping μ, then there is another mapping μ' such that F satisfies the local condition with μ' and slack $\epsilon/2$ and such that $\mu'(C) \leq \Lambda$ for all $C \in F$.*

The proof is carried out in Appendix A.12. Note that the lemma carries over to more general CSPs and other LLL settings; what we have used is only the symmetry (i.e. undirectedness) of the dependency graph and the fact that if a ClSP is non-degenerate, no constraint has weight larger than $1/4$. To any CSP satisfying these properties, the lemma applies.

Hypothesis with Exponential Slack. We will say that for some number $\epsilon \geq 0$, a ClSP F *satisfies the local hypothesis with exponential slack* ϵ if there is a mapping $\mu : F \to (0,1)$ such that for all $C \in F$, we have

$$w(C) \leq \left(\mu(C) \cdot \prod_{D \in \Gamma_F(C)} (1 - \mu(D)) \right)^{1+\epsilon}.$$

This requirement is clearly stronger than the multiplicative one as the weight can in a non-degenerate ClSP not get larger than $1/4$ but can conversely become smaller and smaller in which case the additional ϵ-power on the right hand side tends to zero.

Similarly, we can once again investigate into simplified stronger symmetric variant of the slacked hypothesis.

Lemma 2.41. *If F is a (d,k)-ClSP such that for some $\delta \in [0,1]$, for all $C \in F$ we have*

$$|\Gamma_F(C)| \leq \left(\frac{d^k}{e}\right)^{1-\delta} - 1,$$

then F meets the local hypothesis with exponential slack $\delta/2$.

The proof is carried out in Appendix A.13. In the case of strong exponential slack, the asymmetric version needs a stronger hypothesis too.

Lemma 2.42. *If F is a non-degenerate ClSP and $\delta \in [0,1)$ a number such that for all $C \in F$, we have*

$$\sum_{D \in \Gamma_F(C)} w^{1-\delta}(D) \leq \frac{1}{6},$$

then F satisfies the local hypothesis with exponential slack δ.

The lemma is being proved in Appendix A.14. In analogy to the multiplicative case, the requirement for exponential slack has no effect on the fact that the μ-values can be bounded away from one, as long as the exponential slack is not enormous.

Lemma 2.43. *There exists a universal constant Λ with the following property. If F is a non-degenerate ClSP satisfying the local condition with exponential slack $\epsilon \in [0,1]$ with mapping μ, then there is another mapping μ' such that F satisfies the local condition with μ' and exponential slack ϵ as well and such that $\mu'(C) \leq \Lambda$ for all $C \in F$.*

The proof is in Appendix A.15. Note that we have required the slack to be $\epsilon \leq 1$ here. Exponential slack is also imaginable for larger ϵ in which case this has to be checked with caution. The bound might not be tight, in particular Lemma A.5 which is at the heart of this might be improvable. Under normal circumstances however, ϵ should be (very) close to zero.

Note again that the lemma carries over to more general CSPs and other LLL settings; what we have used is only the symmetry (i.e. undirectedness) of the dependency graph and the fact that if a ClSP is non-degenerate, then no constraint has weight larger than $1/4$. If some CSP satisfies these properties, the lemma applies.

2.8 Parallelization[14]

Our next goal is to show that Algorithm 1 can be parallelized and can be executed in polylogarithmic time on polynomially many processors.

Theorem 2.44. *There exists a randomized parallel algorithm for a CREW-PRAM, which, on input any ClSP F which satisfies the Local Lemma hypothesis with multiplicative slack $\epsilon > 0$, using a number of processors polynomial in size(F), outputs a satisfying assignment in an expected time which depends polylogarithmically on size(F) and linearly on ϵ^{-1}.*

For the definition of the model of computation constituted by the *Concurrent Read Exclusive Write - Parallel Random Access Machine*, abbreviated *CREW-PRAM* and its basic capabilities, the reader is referred to [KR90]. There, it is for example spelt out how basic arithmetic and boolean operations over n bits (like an n-wise Boolean disjunc-

[14]The material in this Section is joint work with Gábor Tardos and has appeared in [MT10].

tion needed for checking whether a clause is currently violated) can be executed in logarithmic time on such a machine.

Before we go on to prove the theorem by exhibiting and analyzing an approriate algorithm, let us check what this means for the running time of our standard applications.

In the symmetric case, where we have a (d,k)-ClSP with every clause having no more than d^k/e inclusive neighbors, we have seen that the hypothesis is satisfied with a multiplicative slack that is proportional to d^{-k}. This means that the algorithm's running time will be proportional to d^k. As long as d^k is polylogarithmic in the size of the formula, the running time will remain polylogarithmic in total. Note that by virtue of Theorem 2.1, d^k cannot grow enormously in any interesting cases; it has to stay well-below $\mathrm{size}(F)$. However, there is a range between polylogarithmic and linear where d^k increases too quickly and the running time increases quickly as well. In the latter case, we need to require that the neighborhoods be at most $(1-\delta)d^k/e$ for some $\delta > 0$ remaining bounded away from zero (or decreasing very slowly). Then we have a sufficient slack according to Lemma 2.38 and the running time will be bounded accordingly. We get the following corollary.

Corollary 2.45. *There exists a randomized parallel algorithm for a CREW-PRAM which on input any (d,k)-ClSP F where for all $C \in F$ we have*

$$|\Gamma_F(C)| \leq (1-\delta) \cdot \left(\frac{d^k}{e} - 1 \right)$$

for some $\delta \geq 0$, using a number of processors polynomial in $\mathrm{size}(F)$, outputs a satisfying assignment in expected time at most

$$\frac{1}{2^\delta - 1 + d^{-k}} \cdot \mathit{polylog}(\mathit{size}(F)).$$

In the case of the simplified asymmetric criterion which we considered in Lemma 2.39, ϵ is constant so for the slacked part F_1 of the

formula, the running time is polylogarithmic.

Corollary 2.46. *There exists a randomized parallel algorithm for a CREW-PRAM which on input any non-degenerate ClSP F such that for all $C \in F$, we have*

$$\sum_{D \in \Gamma_F(C)} w(D) \leq \frac{1}{4},$$

outputs a satisfying assignment in expected time polylogarithmic in size(F) using a number of processors polynomial in size(F).

For a formal proof, we also need to establish what happens on the star components. So we can do this only once we have described and analyzed the algorithm.

Algorithm and Analysis. Not surprisingly, the basic idea of such a parallel algorithm is that we proceed in phases and each phase performs not one correction step but a bunch of correction steps at once.

In order for the parallel steps not to interfere with one another, we require that they work on independent clauses. To make sure that we can work on an independent set of violated clauses in any given phase, we need to determine such a set first. In order for the number of phases to be bounded appropriately, we need that set to be maximal. Luby has described an algorithm in [Lub86] with the following properties. On input a graph G, it runs on a CREW-PRAM with polynomially (in the number of vertices and edges) many processors, and takes polylogarithmic time to negotiate, among all vertices, a maximal independent set in G. Let `LubyMaximalIndependentSet` denote this procedure. As we will run this algorithm on subgraphs $G'_F \subseteq G_F$, it is important to note that Luby's algorithm can easily cope with G'_F being given as G_F with some vertices and/or edges marked as 'deleted'. Indeed, this is part of the strategy the algorithm applies internally.

To start our parallel solver, we in parallel sample a random initial assignment for each variable.

Then, for $i \geq 1$, the i-th phase starts by a negotiation of which independent set of currently violated clauses to resample. To this end, we produce (in parallel or even implicitly) the subgraph $G'_F \subseteq G_F$ induced by the currently violated clauses and run Luby's algorithm on it. We end up with all clauses marked which belong to a maximal independent set M_i of violated clauses. All of this can be done in parallel in polylogarithmic time.

In the second part of the phase, we determine which variables to resample. This as well can be done in parallel in logarithmic time using polynomially many processors per variable. In the end, all variables to be resampled are. The procedure is summarized as Algorithm 4.

Algorithm 4 LocalSolverParallel(F)

Require: A satisfiable ClSP formula F.
Ensure: Output is a satisfying assignment.

parallel foreach $x \in \mathrm{vbl}(F)$ **do**
  $\alpha(x) \leftarrow$ new uniformly random value from L_x
end parallel foreach
while F is not satisfied by α (parallel Boolean evaluation) **do**
  parallel foreach $v \in V(G_F)$ **do**
    mark v if it is violated by α (parallel Boolean evaluation)
  end parallel foreach
  $M \leftarrow$ `LubyMaximalIndependentSet`(marked part of G_F)
  parallel foreach $x \in \mathrm{vbl}(C)$ with $C \in M$ **do**
    $\alpha(x) \leftarrow$ new uniformly random value from L_x
  end parallel foreach
end while

The total running time of one phase is clearly polylogarithmic. We now claim that the expected number of phases necessary is logarithmic on expectation. And thus the overall running time is polylogarith-

mic as claimed. The following lemma readily implies the theorem (via Lemma 2.40).

Lemma 2.47. *Suppose that F satisfies the local condition with mapping μ and slack ϵ. Then, on expectation, no more than*

$$\mathcal{O}\left(\frac{1}{\epsilon} \cdot \log\left(\sum_{C \in F} \frac{\mu(C)}{1-\mu(C)}\right)\right)$$

phases are necessary until the algorithm terminates because all clauses are satisfied.

Proof. Just as in the case of the sequential algorithm, we associate a journal with an execution of Algorithm 4. Let the *journal* be the sequence $\mathcal{M} = \langle M_1, M_2, \ldots \rangle$ of independent sets $M_i \subseteq F$ of clauses resampled in the i-th step. We can serialize it by ordering the constraints within one phase according to the lexicographical ordering. Let $\mathcal{C}$ be the *sequential journal* so obtained. With every clause correction $\langle M_i, C \rangle$ with $C \in M_i$, we can associate a witness tree $T_{i,C}$ defined as the tree T_t we obtain from our construction in the sequential case, where t is the index where the correction of C within phase i occurs in the sequential journal.

We note that $T_{i,C}$ has at least i nodes, which can be seen by induction. For $i = 1$, all trees $T_{1,C}$ with $C \in M_1$ have exactly one node. Now suppose all trees $T_{i-1,\cdot}$ have at least $i-1$ nodes. Now we consider a tree $T_{i,C}$. At the beginning of the i-th phase, constraint C was violated as only then $C \in M_i$ is possible. Furthermore, there exists some $D \in M_{i-1}$ sharing at least one variable with C, as otherwise the variables of C would not have been touched in the $(i-1)$-th phase, implying that C was already violated at the beginning of that phase and could have been added to M_{i-1}, contradicting the maximality of M_{i-1}. By the way $T_{i,C}$ is constructed, it must contain a node to represent each correction step that is represented in $T_{i-1,D}$ and at least

one more node as the root. Therefore $T_{i,C}$ has more nodes than $T_{i-1,D}$ which has $i-1$ nodes.

Furthermore, it is easy to check that the equivalent of Lemma 2.34 holds here. That is, just as in the sequential case, the probability that a fixed witness tree T *occurs* in the journal, i.e. that there exists i and C such that $T = T_{i,C}$ is bounded by its weight $w(T)$. This weight is always at most the μ-weight $w_\mu(T)$, and now, since the local condition is met even with slack ϵ, we have that in fact $w(T) \leq (1-\epsilon)^{|V(T)|} w_\mu(T)$. Therefore, the probability that a fixed witness tree T occurs in the journal is bounded as

$$\Pr\left[\exists i, C : T = T_{i,C}\right] \leq (1-\epsilon)^{|V(T)|} \cdot w_\mu(T).$$

We now bound the probability that the algorithm conducts at least i phases. If it does, then there exists $C \in M_i$ and thus a tree $T_{i,C}$ having at least i nodes which occurs in the journal. So the probability that at least i phases are conducted is bounded by the probability that any witness tree having at least i nodes occurs in the journal. Let $\mathcal{T}_i$ be the set of all witness trees having at least i nodes. Then we obtain

$$\Pr\left[\text{at least } i \text{ phases}\right] \leq \sum_{T \in \mathcal{T}_i} (1-\epsilon)^i \cdot w_\mu(T) \leq$$

$$\leq (1-\epsilon)^i \sum_{T \in \mathcal{T}} w_\mu(T) \leq (1-\epsilon)^i \cdot \sum_{C \in F} \frac{\mu(C)}{1-\mu(C)},$$

where the last inequality is by Corollary 2.37. Given this bound, the claim now follows from Lemma A.6. □

There is the open task left of proving Corollary 2.46. For this we can use the following lemma.

Lemma 2.48. *If the input is a formula F is a union of disjoint stars, then Algorithm 4 runs in expected time polylogarithmic in size(F).*

The proof is done in Appendix A.16.

Corollary 2.46 now readily follows from Theorem 2.44, Lemma 2.48 and Lemma 2.42. Note that we use here that if a formula is input which consists of several disconnected components, then Algorithm 4 runs in an expected time which is at most the sum of the expected times of all components (which is a very crude overestimate but sufficient for our purposes here). So since both the star part (as a whole) and the remainder can be solved in polylogarithmic time, both parts together can, too.

2.9 Derandomization[15]

We have illustrated that philosophically, the power of Algorithm 1 stems from the fact that compared to their likelihood, there are only very few choices of random bits that do not quickly lead to the algorithm's termination. In Section 2.5, this fact was phrased in terms of comparing the information-theoretic effort of encoding such bad executions and juxtaposing the number of random bits they allow to reconstruct. In Section 2.6, we have redone the same thing summing up the probabilities of all possible witnesses of bad executions.

When going for the derandomization of this approach, the former perspective appears to be less suitable than the latter one. If we wanted to exploit the fact that most random strings are incompressible and incompressible strings lead to quick termination of the algorithm, what we would be looking for was a deterministically constructible hitting set for the class of incompressible strings (strings with high

[15]The first part of the section is joint work with Gábor Tardos and has appeard in [MT10]. In the second part, we survey the follow-up result by Chandrasekaran, Goyal and Haeupler in [CGH09].

so-called *Kolmogorov complexity*[16]). Such a hitting set cannot exist by definition, as it would imply the compressibility of the string we are looking for.

On the other hand we are more lucky with the witness tree viewpoint. Since witness trees have a well-defined structure, there might be a chance of deterministically generating sequences of bits replacing the randomness in the algorithm in such a way that large witness trees consistent with the sequence, which are bound to occur in long executions of the algorithm, do a priori not exist. This guarantees that the algorithm terminates quickly when using such a sequence to supplant randomness.

To get formal, suppose again our formula F is over variables $V = \{x_1, x_2, \ldots, x_n\}$. Let a *table* $\mathfrak{T}$ for F be a map $\mathfrak{T} : [n] \times \mathbb{N}_0 \to L$ such that for all $x_i \in V$ and all $j \in \mathbb{N}$, we have $\mathfrak{T}(i, j) \in L_{x_i}$. A table can be used to supplant the supply of random samples $X_i^{(j)}$, that is for each variable $x_i \in V$, we use $\mathfrak{T}(x_i, 0)$ as its initial value and then the j-th time a value for x_i is being resampled we assign value $\mathfrak{T}(i, j)$ to it. We call this a *run of* Algorithm 1 *using table* $\mathfrak{T}$.

A usual randomized run of Algorithm 1 corresponds to a run using a table chosen *uniformly at random*, by which we in this context mean choosing each of the infinitely many entries $\mathfrak{T}(i, j)$ uniformly at random and independently of all other entries. We recall our analysis of such a run using witness trees as done in Section 2.6. Let us recall in particular the proof of Lemma 2.33 where we have defined a reconstruction map $\mathcal{R}'$ to recover variable histories. Let us now say that a given witness tree $T \in \mathcal{T}$ is *consistent* with a given table $\mathfrak{T}$, if $\mathcal{R}'(T)$ is a true reconstruction set given that we use $\mathfrak{T}$ for the variable histories, or, phrased differently, if for all $(i, j, v) \in R'(T)$ we have $\mathfrak{T}(i, j) = v$.

[16] See, for example, Gasarch and Haeupler [GH11] for a review of this topic in the LLL setting.

The randomized analysis has demonstrated that if $\mathfrak{T}$ is chosen uniformly at random, then the expected number of distinct witness trees consistent with $\mathfrak{T}$ can be bounded and this in turn yields a bound on the expected running time. In order to derandomize Algorithm 1, we will assume a slacked hypothesis and use it to prove a stronger bound, namely that the number of *large* witnesses consistent with a random table is smaller than *one* and hence with constant probability, *no* large witness is consistent with such a table. As from this, the existence of many tables $\mathfrak{T}$ admitting no large consistent witnesses follows, we can then use the standard method of conditional expectations due to Erdős and Selfridge [ES73] in order to deterministically construct one. This technique is not new in the context. It has already been applied by Beck in [Bec91] for the very same purpose, although the notion of witness trees was considerably different in his approach (see Section 2.3). Running Algorithm 1 using such a $\mathfrak{T}$ is then deterministic.

Note that if there are no large consistent witnesses, then the running time has to be short: if any clause $A \in F$ is resampled s times, then there exists at least one consistent witness featuring at least s clauses, namely T_t where t is the index of the last time A is being resampled. So if there exists no witness having at least s vertices which is consistent with $\mathfrak{T}$, then running the algorithm using $\mathfrak{T}$ results in each clause being resampled no more than s times.

The notion of what constitutes a "large" witness will depend on what kind of assumptions we make.

Constant Neighborhood Bounds. Our goal is first to prove the following simple statement.

Theorem 2.49. *There exists a deterministic algorithm which on input some CISP formula F satisfying the local condition with multiplicative slack ϵ outputs a satisfying assignment to F in time at most*

$$(\mathcal{O}(|F|))^{\frac{1}{\epsilon}\cdot(1+\ln\Delta)\cdot k} \cdot poly(size(F)),$$

where

$$\Delta := \max_{v \in V(G_F)} deg_{G_F}(v) + 1$$

is the maximum neighborhood size and k is the size of the largest clause in F.

The algorithm is thus polynomial if there is a constant bound on the neighborhood degree Δ and the clause size and if ϵ is not too small. For a simple corollary, we get the following.

Corollary 2.50. *If d, k are constants, then there is a deterministic procedure solving any (d,k)-ClSP F in which each clause shares variables with at most $d^k/e - 1$ other clauses in time polynomial in size(F).*

Proof. Lemma 2.38 yields that if d and k are constant, then under this neighborhood bound ϵ can also be chosen constant. The claim readily follows. □

The asymmetric version from Lemma 2.39 can of course also be used but the dependence on Δ will remain.

After proving the theorem, we will later in the section go on to explain how Chandrasekaran, Goyal and Haeupler [CGH09] managed to relax these preconditions.

For the proof, we proceed as sketched and demonstrate that with constant probability, no large witness tree is consistent with a uniformly random table $\mathfrak{T}$. The first step is to assume that $\mu(C) \leq \Lambda$ for all $C \in F$ with the universal constant Λ which Lemma 2.40 promises to exist. If this bound did not hold, then we would have to apply Lemma 2.40 first to establish it. Note that in this case, ϵ will have to decrease, but only by a constant.

For shortness of notation, define the parameter

$$s := \frac{1}{\epsilon} \ln \left(2|F| \cdot \frac{\Lambda}{1-\Lambda} \right).$$

Lemma 2.51. *If $\mathfrak{T}$ is a table chosen uniformly at random, then the expected number of witness trees which are consistent with $\mathfrak{T}$ and have at least s vertices is at most* $1/2$.

Proof. We start from Corollary 2.37 saying that for every $A \in F$,

$$\sum_{T \in \mathcal{T}_A} w_\mu(T) \leq \frac{\mu(A)}{1-\mu(A)}.$$

Let now $\mathcal{T}'_A \subseteq \mathcal{T}_A$ be the subset of those witness trees with the root label A which have at least s vertices. Using the slacked hypothesis, we obtain that

$$\sum_{T \in \mathcal{T}_A} w(T) \leq (1-\epsilon)^s \cdot \frac{\mu(A)}{1-\mu(A)}.$$

Plugging in the value of s and using that $\ln(1-\epsilon) \leq -\epsilon$ (which is Lemma A.1), we obtain

$$\sum_{T \in \mathcal{T}_A} w(T) \leq \frac{1}{2|F|}.$$

Since the left hand side equals the expected number of large trees consistent with $\mathfrak{T}$, the claim follows by summing over all $A \in F$. □

At least half the tables are hence such that no large tree is consistent with them. We would like to construct such a table deterministically. In order to apply the method of conditional expectations however, we need to be able to compute the expected number of trees consistent with a table of which some entries are fixed and others are kept random. The problem with computing such expected numbers is that $\mathcal{T}' := \cup_{A \in F} \mathcal{T}'_A$ contains infinitely many trees and thus we have to sum infinitely many terms. For remedy, we will define a small list $\mathcal{F} \subseteq \mathcal{T}$ of *forbidden witness trees* with the property that if any $T \in \mathcal{T}'_A$ is consistent with a table $\mathfrak{T}$, then there exists also a tree $\tilde{T} \in \mathcal{F}$ which

is also consistent with $\mathfrak{T}$. Lemma 2.51 implies that the expected number of trees from $\mathcal{F}$ which are consistent with $\mathfrak{T}$ is smaller than $1/2$. Using the method of conditional expectations, we can fix values of $\mathfrak{T}$ one by one until the number of witness trees from $\mathcal{F}$ consistent with $\mathfrak{T}$ has dropped to zero. Then there is no witness from $\mathcal{F}$ that can appear in the journal and by construction of $\mathcal{F}$therefore also no witness from $\mathcal{T}'_A$.

We define $\mathcal{F}$ via the size of the witness trees. Let $\mathcal{F} \subseteq \mathcal{T}'$ be the set of all witness trees T of which the number of nodes lies in the range $[s, ks]$. We claim that this definition serves the purpose.

Lemma 2.52. *Let $\mathfrak{T}$ be any table. If there is a tree $T \in \mathcal{T}'$ consistent with $\mathfrak{T}$, then there is also a tree $\tilde{T} \in \mathcal{F}$ also consistent with the table.*

Proof. Let $T \in \mathcal{T}'$ be a witness tree consistent with the table which has at least s nodes and is among all such trees a tree with the smallest number of nodes. For the sake of contradiction, suppose T has more than ks nodes.

Now consider the artificial journal $\mathcal{C}_T$ (cf. Lemma 2.33) which is recoverable by a bottom-up BFS of T and the entries in this journal which are represented by the children of the root of T. Since the label at the root has at most k literals, there are at most k such children. Let $i_1, i_2, \ldots, i_t$ with $t \leq k$ be the indices of the corresponding entries in the artificial journal $\mathcal{C}_T$. Now build witness trees $T_{i_1}, T_{i_2}, \ldots, T_{i_k}$ for these entries on grounds of $\mathcal{C}_T$ the same way we built witness trees for the usual journal. All these trees are consistent with the table by construction. And since all nodes in T except for the root must be represented in at least one of them, without loss of generality. T_{i_1}, has size at least $|V(T)|/k$. But if $|V(T)| > ks$, then $|V(T_{i_1})| > s$, which is a contradiction as we assumed T to be the smallest such example. □

Using the definition of $\mathcal{F}$, it is easy to bound its size. For any $C \in F$, consider the infinite ordered rooted tree I of which the root is labelled

with C and for every node with label $D \in F$ in the tree, its children are labelled with $\Gamma_F^+(D)$ in lexicographic order. Every witness tree for F is clearly a subtree of I, and according to a well-known bound (see, e.g., [Knu73]), the number of rooted subtrees of size at most ks of an infinite rooted tree where each node has at most Δ children is at most

$$(e\Delta)^{ks} = \mathcal{O}(|F|)^{\frac{1}{\epsilon} \cdot k \cdot (1+\ln \Delta)}.$$

Therefore there cannot be any more than this number of distinct witness trees in $\mathcal{F}$.

To summarize, the determinstic strategy is now the following. We enumerate explicitly all witness trees in $\mathcal{F}$. Then we build a table $\mathfrak{T}$ with which no tree from $\mathcal{F}$ is consitent using the method of conditional expectations: we calculate the expected number of trees from $\mathcal{F}$ which are consistent if the table is chosen randomly. Then we fix $\mathfrak{T}$ entry by entry, each time choosing that value from the admissible domain which minimizes this expected number under the assumption that the remaining entries remain uniformly at random. When filling the table top down, at the latest after having fixed ks values in the history of each variable, this expected value must have dropped to zero. Then we can run Algorithm 1 using the table $\mathfrak{T}$ so constructed for the randomness and we know that it terminates well-before ns correction steps.

This concludes the proof of Theorem 2.49. □

Derandomization under Exponential Slack. This approach was further strengthened in follow-up work by Chandrasekaran, Goyal and Haeupler in [CGH09] so as to remove certain weaknesses of Theorem 2.49. Foremost, we want to be able to treat problems where the clause weight decreases when the problem size increases.

On the cost side, we must transition from the multiplicative slack we have used so far to the stronger exponential-type slack.

Applied to our setting of ClSP problems, Chandrasekaran, Goyal and Haeupler now prove the following. Note that we are stating a largely simplified, convenient but weaker result here, for the full details the reader is referred to [CGH09].

Theorem 2.53 ([CGH09])**.** *There exists a deterministic algorithm which, given a non-degenerate ClSP F satisfying the local condition with exponential slack $\epsilon \in (0,1]$, returns a satisfying assignment in a time which can be bounded by a polynomial in size(F) and $(\min_{C\in F} w(C))^{-1}$ of which the degree is $\mathcal{O}(1+1/\epsilon)$.*

Although there remains the dependency on $(\min_{C\in F} w(C))^{-1}$ and thus applications with sporadic large clauses are still not tractable, this version is considerably more flexible than our Theorem 2.49. As an example, Corollary 2.50 on symmetric (d,k)-ClSPs can now be obtained without assuming constant d and k if there is a slight slack in the hypothesis.

Corollary 2.54. *There is a deterministic procedure solving any (d,k)-ClSP F such that there is $\delta \in (0,1)$ such that each clause shares variables with at most*

$$\left(d^k/e\right)^{1-\delta} - 1$$

other clauses, in a time which is bounded by a polynomial in size(F) and of which the degree depends only on δ.

Proof. Lemma 2.41 yields the required magnitude of ϵ. For applying the theorem, we only need to bound $(\min_{C\in F} w(C))^{-1} = d^k$ by a polynomial. Obviously, this quantity depends exponentially on k. But according to Theorem 2.1, we may assume that $|F| > d^k$, as otherwise we can solve the formula trivially using the method of conditional expectations. Hence d^k is a quantity also polynomial in size(F) and we are done. □

We now go on to explain which are the modifications which were used by Chandrasekaran, Goyal and Haeupler in order to obtain this strengthening. The key is to make witness trees *partial* and *weighted*.

Before we start, assume that the $\mu(C)$ for $C \in F$ are bounded by the universal constant Λ as usual. If not, we can use Lemma 2.43 to achieve this without changing the slack.

Over the fixed set V of variables in our clause satisfaction problem, let a *splitting rule* $\varphi : 2^V \to 2^V \times 2^V$ be a function such that for all $U, U_0, U_1 \subseteq V$, if $\varphi(U) = (U_0, U_1)$, then $U = U_0 \dot\cup U_1$ and if $|U| \geq 2$ then $U_0 \neq \emptyset$ and $U_1 \neq \emptyset$. Fix, globally and arbitrarily, a *canonical splitting rule* φ. That is just to say that whenever some set $U \subseteq V$ of two or more variables is given, then there is a prescribed way of partitioning those variables into two groups. How to do this is arbitrary but globally fixed. We also extend φ to sets of literals in the natural way: a set of literals is being split under φ exactly the way the underlying variables would be split.

Let the *splitting hull* $H(C)$ of a clause C (not necessarily from F) be all clauses that can be produced by repeated applications of the splitting rule, i.e. $H(C)$ is the smallest set which contains C and is closed under φ. Analogously, let

$$H(F) := \bigcup_{C \in F} H(C)$$

be the smallest set containing all of F and closed under φ. For the size of splitting hulls, we note that $|H(C)| \leq 3|C| - 1$ which can easily be seen by induction because $|H(C)| \leq |H(C_1)| + |H(C_2)| + 1$ for $\varphi(C) = (C_1, C_2)$ and the induction base is trivial for unit clauses.

A *partial witness tree* is defined the same way as the witness tree we have used in Section 2.6, with the exception that the root itself is not anymore labelled by a clauses from the formula, but by a member of its splitting hull, i.e. the labelling is now of the type $[\cdot] : V(T) \to H(F)$.

The notions of being *proper* and of being *consistent* with a given table remain the same. For the notions of *weight* and *μ-weight*, let them be defined as for usual witness trees, with the exception that we disinclude the root node from the value, i.e. we define

$$w(T) := \prod_{u \in V(T) \setminus \{\mathrm{root}(T)\}} w([u])$$

and

$$w_\mu(T) := \prod_{v \in V(T) \setminus \{\mathrm{root}(T)\}} \left(\mu([v]) \prod_{C \in \Gamma_F([v])} (1 - \mu(C)) \right).$$

In our weaker proof as discussed above, we have established the statement of Lemma 2.52 to limit the number of witness trees we have to take into account. We have classified witness trees by their size in terms of the number of vertices. Chandrasekaran, Goyal and Haeupler replace this by a similar statement using μ-weights as a notion of size for partial witness trees.

For what follows, let M be the parameter

$$M := 6 \cdot |F| \cdot |V| \cdot \max_{C \in F} \left(\frac{\mu(C)}{1 - \mu(C)} \cdot \frac{1}{w_\mu(C)} \right)$$

and note that the order of magnitude is

$$M = \Theta\left(\left[\min_{C \in F} w(C) \right]^{-1} \right).$$

We note that M is larger than $(\min_{C \in F} w(C))^{-1}$. Next we set the dependent parameter

$$\gamma := M^{-1/\epsilon}$$

which is always smaller than $1/M$, whence it follows that $\gamma \leq w_\mu(C)$ for all $C \in F$.

Recall that in the classical analysis, we defined for any $t \in \mathbb{N}$ the witness tree T_t of which the root is labelled C_t and the related events are being attached during a backward scan of the log. For the present refined analysis, let additionally $A \in H(C_t)$ be any subclause from the splitting hull of C_t. The tree $T_{t,A}$ is build according to the same rules as T_t with the only difference that now the root is labelled with A instead of C_t. The backward scan through the journal works as usual such that only constaints overlapping A can become children of the root. The labels of all non-root vertices are full clauses from the formula as usual. We say that some partial witness tree T *occurs* in the journal if there exists t and $A \in H(C_t)$ such that $T = T_{t,A}$.

We now prove the following analogon of Lemma 2.52.

Lemma 2.55 ([CGH09])**.** *If a proper partial witness tree T of μ-weight at most γ occurs in the journal, then there is also a partial witness tree of a μ-weight in the range $[\gamma^2, \gamma]$ that occurs in the journal as well.*

Proof. Let T be a witness tree, which

- occurs in the journal, i.e. there are t, A such that $T = T_{t,A}$,
- has μ-weight at most γ,
- among all trees satisfying (i) and (ii) has largest μ-weight,
- among all trees satisfying (i), (ii) and (iii) has a root label with the smallest number of literals.

We claim that in this case, T has μ-weight at least γ^2. For suppose the contrary. Then we distinguish two cases. Either the root of T has a single child. Then consider the subtree T' rooted at this child. This subtree occurs in the journal, has a μ-weight larger than that of T, but also μ-weight at most $w_\mu(T)/\gamma$ by definition of γ. But this contradicts the choice of T which was selected to be a tree with largest μ-weight with such properties.

In the other case, the root of T has more than one child. In that case, A must have more than one literal as otherwise it would not be possible for the two children of the root to be disjoint and T is proper by assumption. Then consider the subclauses $(A_0, A_1) = \varphi(A)$ arising from splitting A canonically. And consider $T_0 = T_{t,A_0}$ and $T_1 = T_{t,A_1}$. Since

$$w_\mu(T_0) \cdot w_\mu(T_1) \leq w_\mu(T) < \gamma^2,$$

one of the trees T_0 or T_1 must have μ-weight at most γ. The μ-weight has to be at least as large as the one of T though and the root label has fewer literals than the one in T. This contradicts the choice of T. □

Then, analogously to what we did in our weaker version, they go on to prove that the number of such witnesses is limited so well as their occurrence frequency in the journal. Central to these arguments is the following estimate which is the analogon of Lemma 2.51.

Lemma 2.56 ([CGH09])**.** *Let $\mathcal{F}$ be the set of all proper partial witnesses for F with a μ-weight in the range $[\gamma^2, \gamma]$. Then*

(i) the sum of the weights of all of $\mathcal{F}$ is at most $1/2$.

(ii) the sum of μ-weights of all of $\mathcal{F}$ is at most M.

The proof starts from the bound on the sum of weights of full witnesses from Lemma 2.36 and then it merely needs to translate this bound to partial witnesses. The definition of M is chosen such that the bound becomes constant and smaller than one. For completeness, we include the details in Appendix A.17. Using this bound, we can use basically the same algorithm we used in our weaker variant, but with the updated definition of $\mathcal{F}$.

First we note that just like in the randomized case, a partial witness can only occur in the journal if it is consistent with the random table used. Therefore the probability that a fixed partial witness $T \in \mathcal{F}$

occurs in the journal cannot be any larger than the weight of the tree. Note that we are just ignoring the root node which makes the bound only larger. Using the local hypothesis, the weight can a fortiori not be larger than the μ-weight of the tree. The expected number of trees from $\mathcal{F}$ occurring in the journal is therefore via Lemma 2.56 at most one half and the method of conditional expectations will yield a table with which none of these trees is consistent.

For this method to be efficient, the only thing left to check, just as in the weak variant, is that $\mathcal{F}$ cannot become too large. As a consequence of Lemma 2.56.(ii), we obtain that the μ-weights sum up to no more than M and since each tree in $\mathcal{F}$ has a μ-weight of at least γ^2 by definition, this yields that there can be no more than

$$\frac{M}{\gamma^2} = M^{1+\frac{2}{\epsilon}}$$

trees in this set. This has the required order of magnitude.

The proof of the theorem is thus complete.

Chandrasekaran, Goyal and Haeupler prove considerably more general statements and also combine the parallelization and derandomization to present an approach that can run on a deterministic CREW-PRAM. We omit the details and refer the reader to [CGH09].

2.10 Beyond

In this section, we survey – without proofs – a few results which are important in the context and which extend what we have investigated so far.

Shearer in [She85] has first investigated into questions concerning the tightness of the various formulations of the Local Lemma.

Tightness questions are best formulated as follows. Let G be a simple graph with associated weights $w : V(G) \to [0,1]$. Let us say that $\langle G, w\rangle$ is a *Local Lemma graph* if it holds that for any probability space Ω and any finite set $\mathcal{A}$ of events in that space, if there exists a bijection $\varphi : \mathcal{V}(G) \to A$ such that $\varphi(G)$ is a dependency graph for $\mathcal{A}$ and for every $v \in V(G)$ we have $\Pr[\varphi(v)] \leq w(v)$, then the events do not cover the probability space, i.e. $\Pr\left[\bigcap_{A\in\mathcal{A}} \overline{A}\right] > 0$. Let us write $\mathcal{L}$ for the set of all weighted graphs which are Local Lemma graphs.

For notation, let us say that a ClSP or CSP F has $\langle G, w\rangle$ as a dependency graph if the vertices of G can be mapped one-to-one onto the constraints of F in such a way that G is a dependency graph for F and the constraint's weights are no larger than the corresponding weights w.

All standard formulations of the LLL can be nicely expressed in this terminology. Theorem 2.7 reads that if there exists an association $\mu : V(G) \to (0,1)$ such that for all $v \in V(G)$,

$$w(v) \leq \mu(v) \cdot \prod_{u\in\Gamma_G(v)} (1-\mu(u)),$$

then $G \in \mathcal{L}$. A general-events version of Theorem 2.3 yields that if all weights in G are equal to some value p and G has maximum degree $1/(ep) - 1$, then $G \in \mathcal{L}$, and so forth. However, each of these criteria is only sufficient, not necessary.

Shearer has found an equivalent characterization. He associates

with any weighted graph $\langle G, w\rangle$ the *independent-set polynomials* which are defined for any $S \subseteq V(G)$ as

$$p_G(S) := (-1)^{|S|} \sum_{\substack{I \in T(G) \\ I \supseteq S}} \left((-1)^{|I|} \prod_{A \in I} w(v) \right),$$

where $T(G)$ denotes the set of all independent sets of G and then proved the following.

Theorem 2.57 ([She85]). *$G \in \mathcal{L}$ if and only if $\forall S \subseteq V(G) : p_G(S) > 0$.*

And using this theorem, Shearer could demonstrate that the constant of $1/e$ in the symmetric version of the Local Lemma is tight. Note however that our incarnation of it, Theorem 2.3, is restricted to clause satisfaction problems and the examples of tightness of Shearer use a setup which is far from even being a constraint satisfaction problem.

The question whether the present form of Theorem 2.3 is also tight has been answered by Gebauer, Szabó and Tardos [GST11] in the affirmative. They show that there exist unsatisfiable k-SAT formulas where every clause has at most $(1/e + o(1)) \cdot 2^k$ many neighbors. Their result seamlessly translates to (d, k)-ClSP where d is even and the odd case should be possible to settle by adapting their proofs slighty.

Coversely, Kolipaka and Szegedy [KS11] provided a counterexample which demonstrates that Shearer's condition is not tight for the CSP case, i.e. there do exist weighted graphs $\langle G, w\rangle \notin \mathcal{L}$ which fail Shearer's criterion and still all CSPs having $\langle G, w\rangle$ as a dependency graph are satisfiable.

Bissacot, Fernández, Procacci and Scoppola [BFPS11] have proved a strengthened version of the Local Lemma which was very much inspired by Shearer's independent set polynomial. They showed the following.

Theorem 2.58 ([BFPS11]). *Let $\langle G, w\rangle$ be a weighted graph. If there exists an association $\nu : \mathcal{V}(G) \to (0, \infty)$ of numbers with the vertices such that for*

all $v \in V(G)$, we have

$$w(v) \;\leq\; \nu(A) \;\cdot\; \left(\sum_{\substack{I \in T(G) \\ I \subseteq \Gamma_G(v)}} \prod_{u \in I} \nu(u) \right)^{-1},$$

then $\langle G, w \rangle \in \mathcal{L}$.

By substituting

$$\nu(A) := \frac{\mu(A)}{1 - \mu(A)},$$

we easily find that this version of the local hypothesis is strictly more powerful than the traditional one in Theorem 2.4. Pegden [Peg11] was able to prove that also our algorithmic approach translates seemlessly to this more general version.

Theorem 2.59 ([Peg11])**.** *Suppose that F is a ClSP/CSP having $\langle G, w \rangle$ as a dependency graph and that $\langle G, w \rangle$ satisfies the condition of Theorem 2.58 with mapping ν. Then Algorithm 1 on input F conducts $\sum_{C \in F} \nu(C)$ correction steps in expectation and then returns a satisfying assignment.*

The proof is strikingly simple and relies on the fact that for our proof in Section 2.6, we summed over the set $\mathcal{T}_A$ of witness trees having a fixed clause A at the root label, where a witness tree was defined with the property that any two sibling labels must be *distinct*. We know however that those witness trees which can occur in the journal are what we called proper, which means they have the stronger property that the sibling labels must even be *disjoint* (see Lemma 2.32). One could thus sum only over the smaller set of proper witness trees when estimating the running time of the algorithm. And the Galton-Watson process we used for summing over $\mathcal{T}_A$ can easily be adapted to sum only over proper trees by simply repeating it until the resulting random tree is proper. Doing the straightforward calculations, the theorem surfaces.

Using an even stronger method, Kolipaka and Szegedy were able to demonstrate in [KS11] that if a ClSP/CSP passes Shearer's test, then the expected number of rasamplings which Algorithm 1 conducts is

$$\sum_{C \in F} \frac{p_{G_F}(C)}{p_{G_F}(\emptyset)}.$$

If there is a certain slack of the sort that the violation probabilities of constraints in the CSP are by factor $1+\epsilon$ lower than the probabilities in the Local Lemma graph associated with the problem, then this number drops well-below m/ϵ, and if furthermore $(1+\delta)^2 = 1+\epsilon$, then the parallel version terminates in time

$$\frac{1}{\delta} \log(\frac{m}{\delta}) \cdot \text{polylog}(\text{size}(F)),$$

yielding more powerful versions in cases where the criteria are satisfied more narrowly.

Finally, let us note that the Local Lemma and Shearer's independent-set polynomial have caught the attention of statistical physicists studying the so-called lattice gas, a particular model of particle interactions [SS05, SS06, FP07, KS11]. Their work and ideas have inspired several of the aforementionted improvements to the Local Lemma and exhibit applications beyond combinatorics.

Also, quantum versions of the LLL and our algorithms have recently been developped [AKS10, CS11, Yin11].

3

Schöning's Algorithm

In the previous chapter, we have considered an algorithm that efficiently finds a satisfying assignment to ClSPs meeting the hypothesis of the Lovász Local Lemma. For general formulas without such properties, already *deciding* whether they admit a satisfying assignment is considered to be computationally hard.

We now want to investigate into an algorithm capable of solving *all* formulas, albeit of course coming at the cost of an exponential running time. Analyzed first by Uwe Schöning in [Sch99], it starts at a random assignment and conducts a random local search in its proximity. It bears a striking resemblance to the algorithm we have considered in the previous chapter, though there is an intriguing difference as well.

3.1 Introduction

Exponential time algorithms for solving SAT, ClSP or CSP have a long history. A brute force approach to a general ClSP consists of enumerating the entire solution space. Indeed, if there is no restriction on constraint or alphabet sizes, *no* algorithm known today does substantially better than that. To make any real progress, we at least require some bounds on the input. If F is a (d,k)-ClSP, the brute force approach takes $\mathcal{O}(d^{n+o(n)})$ time.

Branching Algorithms. The first ones to improve on this trivial ansatz were Monien and Speckenmeyer in 1985 [MS85]. In their study of the k-SAT problem, they observed that a simple recursive algorithm can do faster by picking a k-clause $C \in F$ arbitrarily, then branching on the $2^k - 1$ assignments to $\mathrm{vbl}(C)$ which are not forbidden by C. This saves a fraction of one out of 2^k assignments from being explored and since this can be repeated on each recursive level, we get an exponential running time of $\mathcal{O}(c^{n+o(n)})$ for some $c < 2$. The same argument carries over to any $d > 2$.

They went on to observe further potential for improvements, for example that for $C = \{u_1, u_2, \ldots, u_k\}$, branching on the k partial assignments

$$\beta_i = \{u_1 \mapsto 0, u_2 \mapsto 0, \ldots, u_{i-1} \mapsto 0, u_i \mapsto 1\}$$

for $1 \leq i \leq k$ is again exponentially more efficient than always assigning all variables from C. Adding additional tricks and improvements, they achieved a running time of $\mathcal{O}(\Phi^{n+o(n)})$ for 3-SAT and similar nontrivial numbers for larger k, where $\Phi \approx 1.618$ is the *golden ration conjugate*, i.e. the larger of the two real solutions to the equation $\Phi^2 - \Phi = 1$.

Having caught the attention of the community, the simple idea by Monien and Speckenmeyer was to become the starting point of a race

for the best branching rules for k-SAT. To the author's knowledge, the culmination of this type of approach was reached with the 70+-page paper of Kullmann [Kul99] and another surprisingly little-known paper of Rodošek [Rod96], containing 3-SAT algorithms using branching rules of an unprecendented level of sophistication. Rodošek achieved a running time of $\mathcal{O}(1.476^n)$.

Novel Randomized Approaches. As is often the case with this kind of research field, chains of improvements of quickly growing complexity are abruptly rendered obsolete by a novel simple and clean idea. In the case of k-SAT, there were two such milestones arriving in parallel at the end of the last century, inspired also by a new trend to apply randomness in algorithms: the one-pass satisfiability decoding algorithms named *PPZ* and *PPSZ* after its inventors Paturi, Pudlák, Saks and Zane from [PPZ99, PPSZ05] and the random local search algorithm by Uwe Schöning from [Sch99], which was a generalization of the 2-SAT algorithm provided by Papadimitriou in [Pap91].

The rather involved analysis of the PPSZ variant given in [PPSZ05] demonstrated that PPSZ was the fastest known (randomized) algorithm for k-SAT when $k \geq 5$, while Schöning's algorithm with its much simpler analysis seemed to best PPSZ for $k = 3$ and $k = 4$. For 3-SAT, PPSZ was assumed to run in time $\mathcal{O}(1.364^n)$ and Schöning in time $\left(\frac{4}{3}\right)^{n+o(n)}$, or $\mathcal{O}(1.334^n)$ for easy comparison. Improvements to Schöning's algorithm, one by Hofmeister, Schöning, Schuler and Watanabe [HSSW02] to $\mathcal{O}(1.331^n)$ and another by Rolf [Rol03] to $\mathcal{O}(1.328^n)$ followed.

Iwama and Tamaki [IT04] took a closer look and realized that in the cases of $k = 3$ and $k = 4$, Schöning's algorithm was only faster if the search space was rather densely packed with solutions, while in the case of few satisfying assignments, PPSZ had a higher success probability. They combined the two algorithms to interpolate between

the two cases, reaching an algorithm of running time $\mathcal{O}(1.324^n)$. Rolf improved this variant to $\mathcal{O}(1.323^n)$. Iwama, Seto, Takai and Tamaki advanced to $\mathcal{O}(1.322^n)$. With Hertli and Scheder [HMS11], we added a preprocessing step to achieve $\mathcal{O}(1.321^n)$.

Only last year, Hertli [Her11] was able to demonstrate that the analysis which Paturi, Pudlák, Saks and Zane presented in their original work [PPSZ05] involved certain coarse estimates and that when looked at the right way, the PPSZ algorithm had a running time of $\mathcal{O}(1.308^n)$ in the first place. For 4-SAT as well, Hertli established that PPSZ alone was most efficient. On the date of this writeup, therefore, PPSZ is the most efficient algorithm known for k-SAT for arbitrary $k \geq 3$. Moreover, a generalization to non-Boolean (d,k)-ClSP is presently under investigation [Sze11, Mil12].

Deterministic Variants. So much for *randomized* algorithms. If one insists on *deterministic* algorithms not using randomness, the situation looks slightly different. While there is hope that with Hertli's new and simpler analysis, PPSZ will eventually turn out to be derandomizable at little or no loss in efficiency as well, to date, it is not clear whether this works.

The currently best deterministic approaches are derandomizations of Schöning's algorithm, the main reason probably being that its analysis is much simpler and more straightforward and lends itself to an in-depth inspection more easily than the elaborate arguments which are being employed in [PPSZ05] and [Her11]. The first deterministic alternative of Schöning's iterative procedure was given in [DGH$^+$02] which was not lossless and reached a running time of $\mathcal{O}(1.481^n)$ in the case of 3-SAT. This was followed up on by Brueggemann and Kern [BK04], who improved the running time to $\mathcal{O}(1.473^n)$. Later, Scheder [Sch08] and then Kutzkov and Scheder [KS10] improved it further to $\mathcal{O}(1.465^n)$ and $\mathcal{O}(1.439^n)$.

As the main contribution of the present chapter, we demonstrate in

Section 3.7 that the local search approach can be derandomized without any substantial loss in efficiency for all cases of (d,k)-ClSP with $d \geq 2, k \geq 3$. In particular, for 3-SAT, we match the randomized running time of $\left(\frac{4}{3}\right)^{n+o(n)}$, or $\mathcal{O}(1.334^n)$. This is joint work with Dominik Scheder and has appeared in [MS11].

Following up on our work, Makino, Tamaki and Yamamoto in [MTY11] have demonstrated that for the special case of 3-SAT, the improvement to Schöning's algorithm devised by Hofmeister, Schöning, Schuler and Watanabe [HSSW02] can be derandomized as well and then combined with our deterministic version. The resulting combined algorithm is the fastest currently known deterministic procedure for solving 3-SAT, running in time $\mathcal{O}(1.331^n)$.

Outline of the Chapter. In Section 3.2, we present Schöning's randomized ClSP algorithm and we detail the simple and elegant analysis he provided in order to estimate its success probability or running time. We then go on to put the algorithm under closer inspection. Firstly, in Section 3.3, we notice its strong similarity with Algorithm 1 which we have used for formulas satisfying the local condition in the previous chapter and we answer in the negative the question whether the two algorithms are interchangeable. We as well ask questions of tightness of the analysis presented by Schöning, in particular in Section 3.4, we demonstrate that there exist formulas on which this analysis is always tight in a very strong sense. In Section 3.5 on *typical executions*, we take a closer look at the algorithm's analysis and find that the vast majority of paths leading to a successful run have very specific characteristic traits. The findings we make there will provide us with a recipe as to how the algorithm is to be made deterministic. A vital ingredient to this recipe are the so-called *covering codes* which we will define and inspect in Section 3.6. Finally, in Section 3.7, we will present the missing parts of a full and lossless derandomization of the algorithm.

3.2 Algorithm and Analysis

Here is how Uwe Schöning defined his local search algorithm.

Algorithm 5 Schöning(F)

Require: A satisfiable ClSP formula F.
Ensure: Output is a satisfying assignment.

1: $\alpha \leftarrow$ a uniformly random assignment from[1] $\mathcal{S}$
2: **while** $\exists C \in F : \alpha$ violates C **do**
3: $\quad C \leftarrow$ any clause violated by α
4: $\quad x \leftarrow$ a uniformly random variable from vbl(C)
5: $\quad \alpha(x) \leftarrow$ a uniformly random value from $L_x \setminus \{\alpha(x)\}$
6: **end while**

You will immediately notice the strong similarities to Algorithm 1 from the chapter on the Lovász Local Lemma. Indeed, the only difference in *this* iterative procedure is that in every correction step, *one* variable is picked uniformly at random and flipped to a uniformly *different* value instead of resampling *all* variables to new uniform values.

Correctness and termination of this algorithm both follow along the same lines as the arguments in Section 2.3: if F is satisfiable, then every correction step opens an opportunity for getting one step closer to a fixed satisfying assignment and so termination occurs in finite time with probability one. Moreover the loop stopping criterion ensures that the output is always a satisfying assignment.

In how far the two flavors of the same algorithm really differ is not obvious. Unfortunately, it remains a nagging open question whether the variant here terminates in polynomial time on formulas satisfying the local condition. The proof techniques we used in the previ-

[1] Recall that we use the notation $\mathcal{S} := \{\alpha : V_n \to L \mid \alpha(x) \in L_x\}$ for convenience.

ous chapter simply do not carry over well when applying this subtle change. As deciding on a single variable from a clause for flipping needs only $\log k$ bits, for an information theoretic argument to go through one would have to restrict neighborhood sizes to something linear in k. Not only does such a small bound render the algorithm useless because of competition by simple algorithmic versions of Corollary 2.15, but even with the small bound it is not totally clear how to carry out the proof and it would mean a considerable amount of work. But I conjecture that a different suitable proof technique would yield the desired result.

Conjecture 3.1. *If a ClSP satisfies the local condition, then Algorithm 5 terminates in time polynomial in size(F).*

On the other hand, analyzing the performance of Algorithm 1 on general formulas not necessarily satisfying the hypotheses of the local lemma can be done quite simply, adapting the proofs we are going to explain in *this* chapter just slightly. We will do so in Section 3.3.

The two algorithms can most easily be brought under a common parameterized umbrella. Let some number $p \in (0, 1]$ govern the probability which each variable in the clause to be repaired is to be resampled with.

Correctness and termination with probability one are proved the usual way and are independent of the parameter p. It is trivial to see that the algorithm on input $p = 1$ behaves exactly like the Local Lemma Solver.

While there is no p such that the algorithm behaves exactly like Schöning's variant, we can easily see that for $p \to 0$, the algorithm's behavior 'converges' to the one of Schöning's in all essential respects. Having p approach zero, only every (roughly) k/p-th correction step

[2]Recall that we use the notation $\mathcal{S} := \{\alpha : V_n \to L \mid \alpha(x) \in L_x\}$ for convenience.

Algorithm 6 GeneralizedRandomCorrect(F, p)

Require: A satisfiable ClSP formula F, a parameter $p \in (0, 1]$
Ensure: Output is a satisfying assignment.

$\alpha \leftarrow$ a uniformly random assignment from $^2\mathcal{S}$
while $\exists C \in F : \alpha$ violates C **do**
  $C \leftarrow$ any constraint violated by α
  for $x \in \mathrm{vbl}(C)$ **do**
    $v \leftarrow$ a uniformly random value from L_x
    with probability p **do**
      $\alpha(x) \leftarrow v$
    end with
  end for
end while

does at all have an effect, which stretches execution time by a factor of $1/p$. Then *if* a correction step has an effect, there is a negligible probability smaller than kp that anything else but a single flip occurs in the step and as will become apparent shortly, Schöning's analysis is very robust with respect to such very rare effects and thus the algorithm's effectiveness is not affected[3] if this happens. For $p \to 0$, the running time of the algorithm thus 'converges'[4] to the running time of Schöning's variant.

In between, we have a whole continuum of algorithms, but as we demonstrate in Section 3.3, the worst case efficiency deteriorates

[3] At least the worst-case bounds we can prove are not being affected. This does, naturally, not rule out the possibility that *any* subtle changes to the algorithm can have a deterimental effect upon it for particular problem instances for which it normally runs much faster than the worst-case analysis predicts.

[4] The term 'converge' should here be understood philosophically rather than mathematically; we spare ourselves the effort of making this claim precise as it is just here for illustrative purposes.

monotonically with increasing p. The above conjecture 'generalizes'[5] to the parameterized variant.

Conjecture 3.2. *Algorithm 6 terminates in time polynomial in size(F) and $1/p$ if the formula satisfies the local hypothesis.*

But it is only for $p = 1$ that we have a proof for this, and for smaller values the only thing we know is that one can artificially lower neighborhood bounds to accommodate the parameter, a thing which makes little sense to do.

Schöning's Analysis. We now explain Schöning's original, very simple and elegant analysis for bounding the running time of Algorithm 5 when it is being run on a (d,k)-ClSP formula F. The claim is as follows.

Theorem 3.3 ([Sch99])**.** *If F is a satisfiable (d,k)-ClSP formula F on n variables, then the probability that Algorithm 5 on input F terminates well-before carrying out $C \cdot n$ correction steps is at least*

$$\left(\frac{1}{d} \cdot \frac{k}{k-1}\right)^{n+o(n)},$$

where $C = C_{d,k}$ is a constant depending only on d and k but not on n.

This easily turns into an algorithm of expected running time at most

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

by restarting every $C_{d,k} \cdot n$ steps.

In fact, Schöning [Sch99] provided more accuracy by proving that one can choose $C_{d,k} = 3$, thus getting rid of the dependency on d and k,

[5]The term 'generalizes' should again be understood philosophically rather than mathematically; formally speaking, this is not strictly a generalization.

and that moreover the $o(n)$ error term in the exponent can be replaced by a polynomial factor. A contribution by Welzl [Wel12][6] furthered this yet another time so as to determine the actual polynomial which sits in the probability. For the number of steps to be carried out before restart, it will turn out in Section 3.5 that the constant can be bounded further to show that the number of steps necessary actually decreases with increasing values of d and k. For now, we do not insist on the full accuracy as in the case of an exponential time algorithm, both a sublinear term in the exponent and a constant in the number of steps are arguably negligible and the proofs get considerably easier if one accepts them.

For the proof, as we assume that F is satisfiable (note that the unsatisfiable case leads to the algorithm running indefinitely), let us fix a satisfying assignment α^*. We keep track of the progress the algorithm is making by observing the distance $\mathrm{dist}(\alpha, \alpha^*)$ of α^* from the assignment α the algorithm is internally evolving.

Let us call the succession of assignments the algorithm is considering $\alpha_0, \alpha_1, \alpha_2, \ldots$, i.e. α_0 is the uniformly random assignment chosen in the beginning and α_i the assignment after i correction steps for all i. If the algorithm terminates, let all further α_i be the satisfying assignment output so that we always define an infinite series. Note again that very similar to the case of the analysis of the Local Solver, the points in the probability space where the algorithm does *not* terminate form a nullset due to the simple argument supporting termination with probability one, but formally we have to account for the possibility of an infinite run.

Let furthermore

$$D_i := \mathrm{dist}(\alpha_i, \alpha^*)$$

[6]We here refer to the most recent write-up of this contribution in a lecture notes chapter. The result itself is older.

for all i, let

$$N := \min\{\, i \in \mathbb{N}_0 \mid \alpha_i \text{ satisfies } F \,\}$$

be the stopping time (where we consider the minimum to be infinite if it does not exist) and call

$$\mathcal{C} = \langle C_1, C_2, C_3, \ldots \rangle$$

the succession of clauses selected for correction in each step, with some special symbol $C_i = \circ$ for $i > N$. This yields a *journal* of very much the same type as in the analysis of the Local Solver.

Note that $D_i = 0$ implies $N \leq i$ but $N \leq i$ does not imply $D_i = 0$. The reason is that we fixed *some* satisfying assignment α^* of which the formula might have multiple and we cannot prevent one of the others from being found on the way while the D_i are tracking the distance to α^* exclusively.

We now study the evolution of the random process $\{D_i\}_{i \in \mathbb{N}_0}$ by what is well-known as a *coupling argument*. Couplings are used to regularize a random process such that it becomes easier to analyze in a manner which allows for the conclusions of interest to be drawn.

In the present case, since our interest lies in bounding the stopping time N, what we will do is define a regularized process $\{E_i \in \mathbb{Z}\}_{i \in \mathbb{N}_0}$ exhibiting a smoother behavior and satisfying the condition $\forall i \leq N : D_i \leq E_i$. We then bound the time N_E it takes for the E-process to hit zero for the first time. Since $E_i = 0$ together with $N \geq i$ implies $D_i = 0$ and thence $N = i$, this bound will also apply to N. The idea of coupling is that both the D-process and the E-process are being steered by the same supply of randomness and are thus 'coupled'.

Let us define such a common source of randomness. Let $\mathcal{V} := \langle V_1, V_2, \ldots, V_n \rangle$ be an n-tuple of independent uniform random variables taking values from the set $\{1..d\}$ each. Moreover, let us denote by $\mathcal{W} = \langle W_1, W_2, \ldots \rangle$ an infinite supply of independent uniform random variables taking values from the set $\{1..d-1\}$ each and

$\mathcal{X} = \langle X_1, X_2, \ldots \rangle$ an infinite supply of independent uniform random variables taking values from $\{1..k\}$ each.

We now define both the D- and the E-process as deterministic functions of $(\mathcal{V}, \mathcal{W}, \mathcal{X})$.

The D-process arises with the above definitions from running Algorithm 5, where we now use $(\mathcal{V}, \mathcal{W}, \mathcal{X})$ for the randomness in the following way. Let some arbitrary but globally fixed ordering (henceforth called the *lexicographic* ordering) be imposed on both the variables $V = \{x_1, x_2, \ldots, x_n\}$ as well as on the formula F.

Build the starting assignment α_0 by assigning, for all $1 \leq i \leq n$, a value to x_i determined as follows. If $V_i = 1$, assign x_i the value $\alpha^*(x_i)$. Else, map each of the remaining $d-1$ values possible for x_i to the remaining values possible for V_i lexicographically.

Next, for the iterative process, fix any arbitrary rule as to how Algorithm 5 picks violated clauses for fixing, for instance have it pick the lexicographically first violated clause to be C_i in the i-th iteration. Now use $\mathcal{X}$ in order to decide which literal to flip in the following way. Number the variables in $\mathrm{vbl}(C_i)$ in such a way that the lexicographically first variable $x \in \mathrm{vbl}(C_i)$ for which $\alpha(x) \neq \alpha^*(x)$ disagree receives number one. Then number the remaining variables lexicographically. Now select the X_i^{th} variable according to this numbering , let us call it x for now, for flipping.

Then, use $\mathcal{W}$ to decide what value to flip x to as follows. In case $\alpha_{i-1}(x) \neq \alpha^*(x)$, flip $\alpha_i(x) := \alpha^*(x)$ if $W_i = 1$ and map all of the remaining $d-2$ values of W_i to the remaining $d-2$ values possible for flipping in the lexicographic way. If $\alpha_i(x) = \alpha^*(x)$, flip to any of the possible $d-1$ other values according to W_i using the lexicographic mapping.

A 'good case' is now in each iteration if $W_i = 1$ and $X_i = 1$, since this will mean that we flip the first variable in C_i which currently has

'a wrong value' to 'the correct value', at least with respect to α^*. The renumbering however does not alter the behavior of the algorithm at all. Since all random variables used were defined uniformly and independently and since all mappings are one-to-one onto, it requires no further argument to see that Algorithm 5 run this way behaves exactly the way it is normally supposed to behave when this complicated route of preselecting all randomness is not taken. The random variables D_i and N are defined as above, depending on the outcome of this run of Algorithm 5.

For comparison, we now define the regularized E-process as follows. Let E_0 be the number of indices $j \in \{1..n\}$ such that $V_j \neq 1$. Then for all $i > 0$, let

$$E_i = \begin{cases} E_{i-1} - 1 & \text{if } W_i = 1 \wedge X_i = 1 \\ E_{i-1} & \text{if } W_i = 1 \wedge X_i \neq 1 \\ E_{i-1} + 1 & \text{if } W_i \neq 1 \end{cases}.$$

We claim that with these definitions,

$$D_i \leq E_i \quad \text{for all } i \leq N. \tag{3.1}$$

And this is not difficult to see. Note first that $D_0 = E_0$ because for every variable $x_i \in V$, $\alpha_i(x) = \alpha^*(x)$ if and only if $V_i = 1$ and E_0 is defined such that it counts the number of non-ones in $\mathcal{V}$. And then in every evolution step, if the E-process decreases in the i-th step, then $W_i = 1$ and $X_i = 1$, implying that the aforementioned 'good case' occurs and thus the D-process decreases in this step as well. If the E-process rests, then this happens because $W_i = 1$ and $X_i \neq 1$ which means that in the algorithm, a variable which previously disagreed with α^* is selected and flipped to a value which still disagrees with α^* and thus the D-process rests as well. In all other cases, the E-process increases. While the D-process might rest or even decrease in such steps, our invariant is being preserved in all cases.

In particular, Equation 3.1 implies that

$$N_E := \min\{i \in \mathbb{N}_0 : E_i = 0\} \geq N$$

(where again we consider the minimum to be infinite if it does not exist) and so for any constant c,

$$\Pr[N \leq cn] \geq \Pr[N_E \leq cn].$$

So we can now analyze the probability that the simple regularized process hits zero within cn steps and conclude that Algorithm 5 has at least this probability of discovering a satisfying assignment within cn iterations.

And for bounding $\Pr[N_E \leq cn]$, we can invoke Lemma A.7. The Markov chain E is characterized by transition probabilities

$$\tau(-1) = \frac{1}{k} \cdot \frac{1}{d-1}$$

$$\tau(0) = \frac{1}{k} \cdot \frac{d-2}{d-1}$$

$$\tau(1) = \frac{k-1}{k}$$

and therefore according to Lemma A.7, the probability of hitting zero within Cn steps (where C is now the constant provided by the Lemma) if starting from a fixed state j is $\lambda^{j+o(n)}$ where λ is the unique root in the range $(0,1)$ of the polynomial

$$p(x) = \frac{1}{k} \cdot \frac{1}{d-1} + \left(\frac{1}{k} \cdot \frac{d-2}{d-1} - 1\right) \cdot x + \frac{k-1}{k} \cdot x^2,$$

which can easily be found to be

$$\lambda = \frac{1}{(d-1)(k-1)}.$$

This is conditioned on $E_0 = j$. To remove the conditional clause, we use that

$$E_0 \sim \text{Bin}\left(n, 1 - \frac{1}{d}\right)$$

and obtain that $\Pr[N_E \leq Cn]$ is bounded from below by

$$\sum_{0 \leq j \leq n} \binom{n}{j} \left(1 - \frac{1}{d}\right)^j \left(\frac{1}{d}\right)^{n-j} \left(\frac{1}{(d-1)(k-1)}\right)^{j+o(n)}.$$

Using the binomial theorem,

$$\Pr[N_E \leq Cn] \geq \left(\frac{d-1}{d} \cdot \frac{1}{(k-1)(d-1)} + \frac{1}{d}\right)^{n+o(n)}.$$

Or simplified,

$$\Pr[N_E \leq Cn] \geq \left(\frac{1}{d} \cdot \frac{k}{k-1}\right)^{n+o(n)},$$

as claimed, concluding the proof of Theorem 3.3. □

Restricting to Boolean Satisfiability. In the remainder of this chapter, we will go on to investigate interesting aspects of Schöning's iterative procedure and, most prominently, to provide a deterministic variant. Before we set out to such investigations, we want to make an observation which will greatly simplify arguments and calculations throughout the chapter. The observation is that considering the Boolean case $d = 2$ can in the case of Schöning be done without loss of generality.

The reason for this lies in the algorithm's success probability. As the success probability we were able to prove, and it will turn out in Section 3.4 that this calculation was tight in the strongest possible sense, is proportional to d^{-n}, the method for reducing a (d,k)-ClSP to a series of k-SAT problems known as *random downsampling* can be applied losslessly in the present case.

Random downsampling is the following simple idea. If we have a k-SAT solver at our disposition and we would like to solve a (d,k)-ClSP formula F, we can iterate through all variables x occurring in F and for each variable uniformly at random select two out of the d possible values for x. We delete all other $d-2$ values from L_x. The result is a k-SAT formula F' which we try to solve. If a satisfying assignment is not forthcoming, we repeat.

It is very easy to see that the probability that one attempt at random downsampling preserves a fixed satisfying assignment α^* and will thus be a success is exactly $(2/d)^n$. Multiplying this with the success probability for Schöning's algorithm when plugging in $d = 2$ yields a total success probability of

$$\left(\frac{2}{d}\right)^n \cdot \left(\frac{1}{2} \cdot \frac{k}{k-1}\right)^{n+o(n)} = \left(\frac{1}{d} \cdot \frac{k}{k-1}\right)^{n+o(n)}$$

and thus we can recover Theorem 3.3 losslessly without the need to analyze Algorithm 5 for any case other than $d = 2$.

Furthermore, as we will learn in Section 3.6, there is also a near-lossless derandomization of this technique, in the following sense.

Lemma 3.4. *For each n, there exists a set*

$$\Gamma_n \subseteq \left\{\gamma : [n] \to \binom{\{1..d\}}{2}\right\}, \quad |\Gamma_n| = \left(\frac{d}{2}\right)^{n+o(n)}$$

such that for every assignment $\alpha \in \mathcal{S}$, there exists $\gamma \in \Gamma_n$ with the property that α is preserved when restricting L_{x_i} to $\gamma(i)$ for all $x_i \in V$. Moreover, this set is efficiently constructible in the sense that there exists a universal algorithm which on input n and j will output the j-th element of Γ_n in time polynomial in n.

We note that the lemma is trivial in case d is even: simply pair up the elements of every variable's alphabet in an aribtrary way and use the cross product of all the sets of pairs as Γ_n. For odd d, the situation

is slightly more involved and the general proof will be delivered in Section 3.6.

We conclude that both for the randomized and for the deterministic variant of Schöning's algorithm, it is fully sufficient to study the Boolean case. We will therefore assume $d = 2$ henceforward. Concerning the notation of variable values in the following sections, please recall that it is customary to consider the alphabet $\{0, 1\}$ rather than $\{1, 2\}$ in the binary case.

3.3 The Local Solver Contrasted[7]

Similarities between Algorithm 5 and Algorithm 1 are striking and we have already explained how the two algorithms can be regarded as the same parameterized procedure. While we do not know to date whether Algorithm 5 will perform as well as Algorithm 1 on formulas satisfying the condition of the Lovász Local Lemma (see Conjectures 3.1 and 3.2), it is not hard to adapt the machinery from the previous section to draw a conclusion about the performance of Algorithm 1 on general formulas not passing the LLL criteria. With the considerations about random downsampling in mind, we simplify our investigation by restricting to the Boolean case. What we will find is the following.

Theorem 3.5. *When running Algorithm 1 on any satisfiable k-CNF formula on n variables, then the probability that a satisfying assignment is output within $\Theta(n)$ resampling steps is at least*

$$\left(\frac{1+\lambda_k}{2}\right)^{n+o(n)},$$

[7] This is joint work with Andrei Giurgiu. Parts of this material have been published in his Master's thesis [Giu09].

where λ_k is the unique root in the range $(0,1)$ of the polynomial

$$p_k(x) = -x + \left(\frac{x+1}{2}\right)^k.$$

In the case of $k = 3$, the probability is $\phi^{n+o(n)}$, where $\phi \approx 0.681$ is the golden ratio.

Proof. Analogous to the previous section, we define a common source of randomness. Again, we use uniform random variables V_1 through V_n which take values from $\{0,1\}$ to determine the initial assignment, where to every variable x_i we assign $\alpha^*(x_i)$ iff $V_i = 1$. In contrast to the previous section, for the resampling steps we need random variables W_i^j for $1 \leq i \leq k$ and $j \in \mathbb{N}$, where W_i^j takes a uniformly random value from $\{0,1\}$ and is assigned to the i-th variable in the clause C_j during the j-th resampling step. For determining which variable is the i-th, we use the same rule we did in the case of Algorithm 5, i.e. we take the lexicographically first variable where α and α^* disagree to be variable number one, and index the remaining $k-1$ ones lexicographically. And then we assign to each variable x the value $\alpha^*(x)$ if the corresponding sample is one and $1 - \alpha^*(x)$ if the corresponding sample is zero.

As usual, the D-process measures the distances of α from α^* if Algorithm 5 is executed using the prescribed randomness. We juxtaposed a regularized E-process which now looks as follows.

We define E_0 to be the number of indices j such that $V_j \neq 1$. Therefore $E_0 = D_0$. For the evolution steps $i \geq 1$, let E_i be defined as

$$E_i = E_{i+1} - 1 + \sum_{j=0}^{k} W_j^i.$$

We claim that in this case, $D_i \leq E_i$ for all $i \leq N$ as required. To see this, note that in the clause C_i, there is at least one variable x where $\alpha_i(x) \neq \alpha^*(x)$ which we numbered 'variable one' before. The other

$k-1$ variable can be agreeing or disagreeing. The D-process decreases by the number of variables which previously disagreed and are resampled to agree and increases by the number of variables which previously agreed and are now resampled to disagree. The E-processes decreases by one if and only if the designated disagreeing variable is flipped and increases by the total number of other variables resampled to disagree. From this comparison, it is obvious that $D_i \leq E_i$ is preserved during such a step.

Now consider the distribution of the steps in the E-process. Since the variables are flipped independently, the difference between consecutive steps is distributed binomially. More precisely, we find that in the terminology of Lemma A.7, E is a Markov chain with transition probabilities

$$\tau(i) = \binom{k}{i+1} \cdot \left(\frac{1}{2}\right)^k$$

for all $i \in \{-1, 0, \ldots, k-1\}$. The lemma now yields that the probability that when starting from a fixed state j of reaching zero within $\Theta(n)$ steps is determined by $\lambda_k^{j+o(n)}$, where λ_k is the unique root in $(0,1)$ of the polynomial

$$p_k(x) = -x + \left(\frac{1}{2}\right)^k \cdot \sum_{i=0}^{k} \binom{k}{i} x^i = -x + \left(\frac{1+x}{2}\right)^k.$$

Since E_0 is distributed binomially with parameters n and $1/2$, we obtain for the total success probability within cn steps

$$\left(\frac{1}{2}\right)^n \sum_{i=0}^{n} \binom{n}{i} \lambda_k^{i+o(n)} = \left(\frac{\lambda_k + 1}{2}\right)^{n+o(n)}.$$

The number in 3-SAT follows by calculating the root. □

These success probabilities can easily been checked to be exponentially below the performance of Algorithm 5. And if it appears that the estimates and the coupling used in this proof might simply be too

lossy for the variant under consideration, the arguments from the following section demonstrate that Theorem 3.5 is actually tight, at least if a worst-case choice rule (a 'devil', see next section) is scheduling clauses for correction.

3.4 Angels and Devils[8]

Our next mission is to determine whether Schöning's analysis of Algorithm 5 is tight. If we scan Section 3.2 for inaccuracies, we find that the only place we have been rather generous was the coupling argument: firstly, we have considered only one single satisfying assignment α^* and secondly, if the algorithm selected a clause where the current assignment and α^* disagreed in more than one variable, we have declared only one of these variables to offer a productive flip and treated the other ones as if they already agreed. In contrast, the calculations on the success probability in Lemma A.7 can easily be seen to be tight, there are no estimates involved there.

It is easy to produce samples of k-SAT formulas where the first assumption, the uniqueness of the satisfying assignment α^*, is tight. For example, consider the maximal satisfiable formula F_n^+ on n variables consisting of all clauses containing at least one positive literal. F_n^+ has the unique satisfying assignment α^* which sends every variable to one. If this formula is input to Algorithm 5, the only way the algorithm can terminate is when $D_i = 0$. In this case $D_i = 0$ and $N \leq i$ are equivalent conditions.

The second assumption, that in the selected clause there is exactly one variable where the current assignment disagrees with α^* and thus whose flipping will be advantageous, moves the focus to an aspect of

[8]Part of this section is joint work with Andrei Giurgiu and Stefan Schneider. Parts of this material have been published in their Master's theses [Giu09, Sch10].

the algorithm so far largely ignored: the *choice rule*.

The description of Algorithm 5 admits in each iteration 'any' way of selecting a clause C_i among all currently violated clauses for fixing. For the analysis of the success probability, we have fixed an arbitrary rule for doing this selection so that our random variables would be well-defined, but the analysis went through for *any* rule.

The choice rule can make a considerable difference. If F_n^+ is input to the algorithm, then selecting an appropriate choice rule can make the algorithm terminate in polynomial time: as long as the current assignment differs from the satisfying assignment in at least k variables and by the maximality of F_n^+, we can have it select a clause such that all k variables in that clause differ. No matter what literal is chosen for flipping, the distance to α^* is bound to decrease by one with probability one. This can be done until α and α^* differ by at most $k-1$ variables and from that point onwards, the success probability is constantly large according to the standard analysis. In case of failure of such an attempt, a 'good' clause can be selected again and we get another chance. The total number of iterations it takes on average is linear in n.

On the other hand, another choice rule can select, as long as α^* and α do not differ in more than $n-k$ variables which is a rare event, a clause in each iteration in which all but one variable already agree with α^* and there is just one the flipping of which would take us closer to the solution. This situation is in one-to-one correspondence with the definition of an evolution step of the regularized E-process, resulting in Schöning's analysis to be tight in this case.

Choice rules of the latter type is what we will call a *devil* because they aim at hindering the algorithm at terminating successfully for as long as it is in any way possible. Analogously, we call a choice rule of the former type which can make the algorithm terminate in polynomial time an *angel*.

Types of Angels. An important thing to note is that in the examples of angels and devils for F_n^+ which we have just presented, computational issues have not played any role. This is deliberate: choice rules in the way we want to discuss them are unbounded in computational resources (implying in particular that they know, e.g., a satisfying assignment). We are distinctly *not* talking about heuristics for choosing clauses for fixing where the question of how efficient they are in making their choice is central. Rather, as the main statement in this section, we will later go on to demonstrate that there are formulas for which the choice rule has no significant influence on the time it takes Algorithm 5 to reach the satisfying assignment, *not even* if computationally unbounded rules are considered.

Formally therefore, a *choice rule* is nothing but a *mapping* which selects, depending on the current state of the algorithm, a clause to be fixed. We can give choice rules various degrees of insight into an algorithm's run. The weakest version would tell the choice rule which clauses are currently violated only. A slightly stronger one could give the complete current assignment. Even stronger ones could take into account other information about the algorithm's execution, like how many correction steps have been carried out so far. The strongest type of rule will be the one we apply here: a rule which can take into account all information about the complete history of the algorithm's execution. As such, a *choice rule* is defined to be any mapping from the input formula F and the algorithm's execution journal composed of $\langle \alpha_0, C_1, \alpha_1, C_2, \ldots, \alpha_i \rangle$ to some clause $C_i \in F$ violated by α_i to be selected in the subsequent correction step.

Let us define an *angel* for a family $\{F_n\}_{n\in\mathbb{N}}$ of k-SAT formulas where F_n is over n variables as a choice rule with the property that the success probability of Algorithm 5 when using this choice rule and running the algorithm for a number $q(n)$ of steps where q is an appropriate polynomial, is lower bounded by the inverse of some polynomial in n.

Conversely, a *devil* for a family $\{F_n\}_{n\in\mathbb{N}}$ of k-SAT formulas where F_n is over n variables is a choice rule such that for any polynomial $q(n)$, the probability which Algorithm 5 has to succeed, even if we allow it to make $q(n)$ correction steps before we abort, is upper bounded by

$$\left(\frac{k}{2(k-1)}\right)^{n+o(n)}.$$

The maximal satisfiable formulas F_n^+ we have discussed in the introductory example admit both angels and devils as selection rules. For some formulas, angels exist but finding them is highly non-trivial. The Master's thesis of Andrei Giurgiu [Giu09] features a collection of such formulas and the corresponding angels.

Also, one can discuss the various possible definitions of angels and compare their strength and universality properties. In [Giu09], many different definitions are being compared and constrasted.

Perhaps the most interesting distinction is the one between *angels in probability* and *angels in expectation*. Instead of letting Algorithm 5 run for $q(n)$ many steps for some polynomial q and asking the success probability within that time frame to be polynomial (which is what we call an *angel in probability*), we could instead consider letting Algorithm 5 run for an arbitrary number of steps and ask that the expected number of steps it takes to arrive at a satisfying assignment be bounded by some polynomial $q(n)$ (which we call an *angel in expectation*).

While it is clear by Markov's inequality that an angel in expectation is also an angel in probability for suitably chosen polynomials, the converse is not true. Consider the family of formulas $\{F_n^+\}_{n\geq k}$ we defined previously. Recall that for any assignment α with

$$k-1 \leq \operatorname{dist}(\alpha,\alpha^*) \leq n-k+1$$

there always exists at least one violated clause where all variables differ between α and α^* and one violated clause where only exactly one

variable does. Let now R be a choice rule which, if α is the current assignment, selects a clause of the former type whenever

$$k - 1 \leq \mathrm{dist}(\alpha) \leq \frac{n}{2} - \sqrt{n}$$

and a clause of the latter type in all other cases. The probability that the starting assignment will satisfy

$$\mathrm{dist}(\alpha_0, \alpha^*) < \frac{n}{2} - \sqrt{n}$$

is inversely polynomial since the distance is binomially distributed. Given that this event occurs, since R will then select clauses where all variables differ until we reach a state with

$$\mathrm{dist}(\alpha_0, \alpha^*) = k - 1,$$

the probability that we reach the satisfying assignment in n steps is constant as in the introductory example. Therefore, R is an angel in probability for $\{F_n^+\}_n$. On the other hand, the probability that the starting assignment will satisfy

$$\mathrm{dist}(\alpha_0) > \frac{n}{2} + \sqrt{n}$$

is also bounded from below by the inverse of a polynomial. But if we start there, then the choice rule will select clauses with just one productively flippable variable at least as long as we have not reached a state with

$$\mathrm{dist}(\alpha, \alpha^*) \leq n/2 - \sqrt{n}.$$

But the expected time it takes to overcome these $2\sqrt{n}$ states with corrections steps successful only with probability $1/k$ is – as we know from the analysis of Markov chains in Lemma A.7 – clearly exponential in $\sqrt{n}$ and hence superpolynomial in n. Therefore, R is not an angel in expectation for $\{F_n^+\}_n$.

Still, the choice rule R is a somewhat artificial rule constructed with the goal in mind of exhibiting exactly this hybrid behavior. As we already know, the given family of formulas admits both a pure devil and

a pure angel. Unfortunately, we have so far not been able to establish the following conjecture.

Conjecture 3.6. *If a family $\{F_n\}_{n\in\mathbb{N}}$ of formulas admits an angel in probability then it also admits an angel in expectation.*

In this section, we are however not so much considered with different possible definitions of angels but rather with the tightness of Schöning's analysis. We will now establish that there exist formulas which do not only not admit angels but rather which *always* take the same expected effort, no matter what choice rule is being used.

Worst Case Formulas. For any k, we construct a family of k-SAT formulas $\{F_n^{\sim k}\}_{n\in\mathbb{N}}$ with the property that the choice rule cannot make any substantial difference. We call these formulas *chain formulas* based on their shape: each $F_n^{\sim k}$ over the variables $\{x_1, x_2, \ldots, x_n\}$ consists of all the $2^k - 1$ clauses over $\{x_1, x_2, \ldots, x_k\}$ which contain at least one positive literal and then additionally for $i \geq k+1$ one more clause $\{\overline{x_{i-k+1}}, \ldots, \overline{x_{i-1}}, x_i\}$ each. The *core* of the first $2^k - 1$ clauses enforces that the only satisfying assignment must set the first k variables to one. The *chain* of remaining clauses attached to it then enforces that all other variables be set to one as well. There is hence only one unique satisfying assignment α^*. We now prove that the choice rule has little influence on the success probability of Algorithm 5.

Theorem 3.7. *Let $q(n)$ be any polynomial. The probability with which Algorithm 5, on input $F_n^{\sim k}$, outputs a satisfying assignment after at most $q(n)$ correction steps, equals*

$$\left(\frac{k}{2(k-1)}\right)^{n+o(n)}.$$

Or stated differently, all choice rules are devils on the family $\{F_n^{\sim k}\}_{n\in\mathbb{N}}$.

Proof. The idea is that we consider each variable x_i for $i > k$ as a separate random process with the goal of setting it to 'true'. If x_i is set to 'true' in the initial assignment, this process is immediately terminated. Otherwise, there is only one way of fixing the variable, namely the selection of clause $D_i := \{\overline{x_{i-k+1}}, \ldots, \overline{x_{i-1}}, x_i\}$ by the choice rule. Once this happens, there is a one in k chance that x_i will be fixed and the process terminates. In all other cases, another variable previously set to the correct value 'true' will be reset to 'false'. We consider this newly appearing 'mistake' as being the 'fault' of x_i. By virtue of the shape of the formula, the newly created mistake will *have* to be fixed well-before another attempt at flipping x_i can be made, because D_i is satisfied as long as the other variable stays set to 'false'. The analysis then counts the number of 'faults' currently to be 'blamed' on x_i and analyses the evolution of this number as a separate random process.

Let us formalize. We define the *blame assignments* $\beta_i : V \to V \cup \{\circ\}$ for all $0 \leq i \leq N$ as follows. Initially, for all $x_j \in V$

$$\beta_0(x_j) = \begin{cases} x_j & \text{if } \alpha_0(x_j) = 0 \\ \circ & \text{otherwise} \end{cases}.$$

In words, initially each variable set to zero is *blamed* for its own currently wrong value. And each variable set to one is free of blame. In the subsequent evolution steps of the algorithm, we propagate the blame of any variable set to zero to variables newly set to zero during an attempt at fixing the variable. That is we set, for all $i \geq 1$,

$$\beta_i(x_j) = \begin{cases} \circ & \text{if } \alpha_i(x_j) = 1 \\ \beta_{i-1}(x_r) & \text{if } \alpha_i(x_j) = 0 \text{ and } C_i = D_r \text{ and } x_j \in \mathrm{vbl}(D_r) \\ \beta_{i-1}(x_j) & \text{otherwise} \end{cases}.$$

So in each step, if a variable is flipped from zero to one, the corresponding variable gets mapped to $\circ$. If the choice rule selects a clause from the chain and if a variable is flipped from one to zero, then this

variable inherits the blame value from the variable x_r which supplies the only positive literal in the clause D_r being selected. If a core clause is selected for flipping, blame is not propagated.

Taking a closer look at the blame assignments, we see that in the vectors $\langle \beta_i(x_1), \beta_i(x_2), \ldots, \beta_i(x_n) \rangle$, non-$\circ$-values occur sorted, i.e. no variable x_j can occur to the left of a variable $x_{j'}$ with $j' < j$. This is because a new non-$\circ$-value x_j can in β_i only appear in one particular case, namely if D_r is selected for correction where x_r is the leftmost variable having $\beta_{i-1}(x_r) = x_j$, and the $k-1$ variable to the left of x_r must currently be assigned $\circ$ as otherwise D_r could not be selected. For the same reason, between any two occurrences of the same variable, there are at most $k-1$ other positions which are all assigned the value $\circ$. Conversely, a new $\circ$ can only appear when it replaces the leftmost occurrence of some variable.

We will now consider for each x_j with $j > k$ a random process $\{M_i^{(j)}\}_{i \in \mathbb{N}_0}$ which we define as

$$M_i^{(j)} = |\{x \in V \mid \beta_i(x) = x_j\}|.$$

These processes are resting in most of the cases. As argued above, the value $M_i^{(j)}$ can change only in two cases: either D_r is selected for correction where x_r is the leftmost variable blamed on x_j. In this case, with probability $1/k$, the process decreases, with the remaining probability it increases by one. The other case occurs if the leftmost variable blamed on x_j is one of the variables x_1 through x_k. In this case the process can only decrease.

The influences of the separate variables are now disentangled. For the algorithm to arrive at the satisfying assignment, all processes must decrease to zero. And the processes evolve *independently* of one another: the only type of influence there is between two processes is that if one process increases so much that the leftmost variable it includes and the rightmost variable of the next non-zero process to its left occur

together in one clause, then that process is *blocked* from further evolution until the process to its left decreases to zero and terminates. Only then can the right process continue to evolve.

Consequently, the influence of the *choice rule* is reduced to a minimum: it can only choose which of the parallel processes to evolve at what time, subject to blocking constraints. Apart from that, the processes run independently and their transition probabilities are predetermined and cannot be influenced by the rule.

Therefore, we can upper bound the success probability under *any arbitrary* choice rule by bounding the probability that all the processes hit zero when run in parallel for the given $q(n)$ steps. For each process, its probability of hitting zero gets only larger if we assume that all $q(n)$ steps evolve this process when in reality the choice rule may pause it (may even *have to* pause it) during lots of these steps. Looked at this way, each process corresponds to a Markov chain which in each step decreases with probability $1/k$ and increases with the remaining probability. Or, if the value of the process is so large that its leftmost position occurs in the core clauses, then it always decreases with probability one.

In other words, for every variable x_i for $i > k$, we have a Markov chain with transition probabilities $\tau(-1) = 1/k$ and $\tau(1) = (k-1)/k$ which has a reflection point at $i/(k-1)$, where we are generous because the reflection can as well occur only much later, depending on the number of $\circ$-positions between consecutive occurrences of x_i, which will influence matters only in favor of the proof goal. With probability $1/2$, this chain is activated with starting state one, which is the initial number of blames on x_i. With probability $1/2$, it is not activated at all. According to Lemma A.12 where such chains are being considered, the probability that it hits zero when starting at state one is at most

$$\frac{1}{k-1} + q(n) \cdot \left(\frac{1}{k-1}\right)^{\frac{i}{k-1}}.$$

The algorithm is successful only if all the variables x_i for $i > k$ are either one already in the beginning or are being fixed. Since we have demonstrated that the fixing processes evolve independently, we obtain that the total success probability cannot be any larger than

$$\prod_{i=k+1}^{n} \left(\frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{k-1} + q(n) \cdot \left(\frac{1}{k-1} \right)^{\frac{i}{k-1}} \right) \right).$$

This can be rewritten as

$$\underbrace{\prod_{i=k+1}^{n} \left(\frac{k}{2(k-1)} \right)}_{\left(\frac{k}{2(k-1)}\right)^{n+o(n)}} \cdot \underbrace{\prod_{i=k+1}^{n} \left(1 + q(n) \cdot \frac{2(k-1)}{k} \cdot \left(\frac{1}{k-1} \right)^{\frac{i}{k-1}} \right)}_{:=\varphi_n}.$$

The left product yields the success probability we are aiming for. The right product is the correction factor representing the additional success probability caused by the possibility of reflections at the core clauses. It remains to show that φ_n is subexponential in n.

To see this, let $I(n)$ be the smallest integer i such that the bracketed expression is below two. Note that since the second summand in the bracket is exponential in i, $I(n)$ is polylogarithmic in n. If we split the product at $i < I(n)$ versus $i \geq I(n)$, we obtain

$$\varphi_n \leq (1 + 2q(n))^{I(n)-k} \cdot \prod_{i=I(n)}^{n} \left(1 + q(n) \cdot \frac{2(k-1)}{k} \cdot \left(\frac{1}{k-1} \right)^{\frac{i}{k-1}} \right),$$

where we have used that our factors are upper bounded by $(1+2q(n))$ even for small i. We note that the first part of this product is subexponential because $I(n)$ is polylogarithmic and all the factors there are polynomial in n. For the second part of the product, we note that since $i \geq I(n)$, the second summand of each factor is a constant below one times something which decreases exponentially with i. We can thus use the variable transformation $i' = i - I(n)$ and apply Lemma A.1 so

as to obtain

$$\varphi_n \leq 2^{o(n)} \cdot \exp\left(\sum_{i'=0}^{n} \left(\frac{1}{k-1}\right)^{\frac{i'}{k-1}}\right) = 2^{o(n)},$$

since the geometric series is clearly convergent. □

This demonstrates that there exist formulas for which Schöning's original analysis is tight, no matter what rule is used for the scheduling of violated clauses for correction.

3.5 Typical Executions[9]

As outlined, the main contribution of this chapter will be a deterministic variant of Algorithm 5. There are many standard techniques around for derandomizing algorithms but very few of them seem to be applicable to the setting of exponential time search problems. At any rate, the most basic advice to give to anybody trying to derandomize an algorithm is to study the randomized variant more closely and figure out whether there are obvious reasons why the number of random samples used is superfluously abundant. This is what we will do here and once we understand this question in the case of Schöning's algorithm, the derandomized variant will in fact be immediate.

In Schöning's algorithm, random decisions are taken to generate an initial assignment α_0 to start from and then in every step to select a variable $x \in \mathrm{vbl}(C_i)$ for flipping. The choice of initial assignment thereby consumes n random bits once, the choice of $x \in \mathrm{vbl}(C_i)$ another $\log k$ bits per iteration. We have already figured out that cn of

[9] This is preparatory work for the derandomization in Section 3.7 which is joint work with Dominik Scheder and has appeared in [MS11]. A similar write-up has appeared in [MW12b].

these steps suffice to achieve an optimal success probability, where c is the constant we proved to exist, and so if we interrupt the algorithm after this number of steps, it has consumed a total of

$$q := n + \log k \cdot cn$$

random bits at most. Out of all the 2^q many possible random bitstrings, according to Theorem 3.3, at least a fraction of

$$\left(\frac{1}{2} \cdot \frac{k}{k-1}\right)^{n+o(n)}$$

must lead to successful termination outputting a satisfying assignment. However, without knowledge of the distribution of these 'good' bitstrings within the set of possible bitstrings, a straightforward brute force approach at derandomization must iterate through all of the latter, resulting in a running time very much worse than the randomized one. The next step is therefore to identify traits of 'typical' successful executions of the algorithm such that if we look at all random bitstrings of a certain shape, we are sure to hit one of the good ones.

To become formal, we recall the definition of the E-process from Section 3.2 which we coupled to the much less controllable behavior of the algorithm. Let us denote by

$$S := \{N_E \leq cn\}$$

the event that the process hits state zero before doing any more than cn evolution steps. We have demonstrated that *if* S occurs, *then* Algorithm 5 succeeds to output a satisfying assignment within the prescribed number of steps as well via the coupling argument. We have further demonstrated that

$$\Pr[S] \geq \left(\frac{1}{2} \cdot \frac{k}{k-1}\right)^{n+o(n)},$$

from which, as we recall, Theorem 3.3 followed.

Now let $\mathcal{E}$ be any arbitrary event determined by an evaluation of the E-process. We call the event $\mathcal{E}$ *typical for Schöning* if it has a high coincidence with successful executions, namely if

$$\Pr[\mathcal{E} \cap S] \geq \left(\frac{1}{2} \cdot \frac{k}{k-1}\right)^{n+o(n)}.$$

Note that for such an asymptotic statement to be well-formulated, $\mathcal{E} = \mathcal{E}(n)$ must in fact represent a whole sequence of events, one for every number n as we also define one E-process for every n. In the sequel, this is implicitly understood and we omit all indices for conciseness.

Intuitively, if in addition to success S we additionally ask $\mathcal{E}$ to occur, this lowers the probability only by a subexponential and in the context therefore negligible amount. For our purposes, this will mean that if we identify some $\mathcal{E}$ which is typical, it will be okay to analyze the performance of the algorithm considering only $S \cap \mathcal{E}$ to be 'a success'. As such a restriction does not impair the performance of the algorithm, it is harmless to impose, and as on the other hand finding a suitable $\mathcal{E}$ can narrow down the ampleness of possible paths leading to success to a more streamlined class, it may – and will – help to identify an optimal or near-optimal hitting set for the purpose of derandomization.

Observe that finding typical events is not difficult. Whenever we consider all paths leading to success S, any partition of these into subexponentially many classes contains at least one part which is typical. For instance, consider, for fixed $0 \leq j \leq n$, the event $\{E_0 = j\}$ that the starting state is at distance j from zero. Since the union of these events over all j cover the whole probability space and since there are only n many in total, one of them at least must intersect at least a $1/n$-fraction of the measure of S and thus be typical.

Theorem 3.8. *Let $f = o(n), f = \omega(1)$ be some positive, slowly growing integral function and define the parameters*

$$z := \lfloor n/f(n) \rfloor,$$

$$j := \left\lceil \frac{1}{k} \cdot f(n) \right\rceil,$$

$$r := \left\lceil \frac{1}{k(k-2)} \cdot f(n) \right\rceil,$$

$$l := r + j \approx \frac{k-1}{k(k-2)} \cdot f(n),$$

$$t := r + l \approx \frac{1}{k-2} \cdot f(n).$$

The intersection of the following events is typical for Schöning.

(i) $E_0 = jz$
(ii) *for all integers* $0 \leq b \leq z$,

$$|\{bt + 1 \leq i \leq (b+1)t \mid E_i = E_{i-1} - 1\}| = l,$$

$$|\{bt + 1 \leq i \leq (b+1)t \mid E_i = E_{i-1} + 1\}| = r.$$

The proof is a technical calculation and is carried out in detail in Appendix A.18.

What it tells us intuitively is that a large fraction of successful executions of the E-process, so large that we are okay with ignoring the rest, starts exactly at state n/k, then progresses towards zero in phases of t steps each in such a way that in each phase, there are l steps where the E-process decreases and r steps where it increases, to finally be at zero after exactly zt steps (not implying that it could not have reached zero already slightly earlier).

This will provide us with an almost straightforward recipe for derandomizing Algorithm 5. The tent poles are: first find an assignment α_0 to start from which has distance at most n/k from α^*, then for a suitable, slowly growing function $f(n)$, execute z phases of t flips, making sure that in each phase we make at least l good flips and at most r bad flips. This is exactly what we will do in Section 3.7. There will be some special cases which we have to care about, and for actually *finding* α_0

and suitable flipping patterns, we need another key ingredient, the *covering codes*.

3.6 Covering Codes[10]

In this section, we investigate different types of codes which we will be using in our deterministic variant of Schöning's algorithm.

Let Σ be a finite set which we call the *alphabet*. As usual, we use the notation Σ^n to denote all strings of length n over Σ. A *code over* Σ *of length* n is a set $\mathcal{C}_n \subseteq \Sigma^n$. The *size* of a code $\mathcal{C}_n$ is $|\mathcal{C}_n|$.

A *code sequence* is a sequence $\mathfrak{C} = \langle \mathcal{C}_1, \mathcal{C}_2, \ldots \rangle$ of codes where $\mathcal{C}_n$ is a code over Σ of length n. For a code sequence, we say that $\mathfrak{C}$ has *size* $\lambda \in [0,1]$ if $\mathcal{C}_n$ has size $\lambda^{n+o(n)}$ for all n. A code sequence $\mathfrak{C}$ is *efficiently constructible* if there exists an algorithm which on input two integers n and j, outputs the j-th codeword of $\mathcal{C}_n$ in time polynomial in n.

A *code property* $\mathcal{P}$ is a sequence $\mathcal{P} = \langle \mathfrak{P}_1, \mathfrak{P}_2, \ldots \rangle$, where for each n, $\mathfrak{P}_n$ is a set of codes over Σ of length n. A code $\mathcal{C}$ of length n is said to *satisfy* property $\mathcal{P}$ if $\mathcal{C} \in \mathfrak{P}_n$. We say that property $\mathcal{P}$ is *multiplicative* if it is true that whenever $\mathcal{C}_1 \in \mathfrak{P}_n$, $\mathcal{C}_2 \in \mathfrak{P}_m$ are two codes of length n and m satisfying $\mathcal{P}$, then their cartesian product $\mathcal{C}_1 \times \mathcal{C}_2 \in \mathfrak{P}_{n+m}$ is satisfying $\mathcal{P}$ too. A code sequence $\mathfrak{C}$ *satisfies* a code property $\mathcal{P}$ if $\mathcal{C}_n \in P_n$ holds for all n. We say that $\mathcal{P}$ is *efficiently checkable* if there exists an algorithm to determine whether a code $\mathcal{C}$ given as input satisfies $\mathcal{P}$ which runs in time polynomial in $n \cdot |\mathcal{C}|$.

With these notions in mind, we find a generic formulation for a family of codes which are efficiently constructible.

[10] None of the results presented here are new. Some of the material appears in [MS77] and it has been applied already in the first derandomization of Schöning due to Dantsin et al. [DGH+02]. Much of it is sufficiently elementary to consider it folklore.

Lemma 3.9. *If $\mathcal{P}$ is a multiplicative and efficiently checkable property and there exists a code sequence $\mathfrak{C}$ of size λ satisfying $\mathcal{P}$, then there also exists a code sequence $\mathfrak{C}'$ of size λ satisfying $\mathcal{P}$ which is moreover efficiently constructible.*

Proof. Since $\mathcal{P}$ is a multiplicative property, a block construction can be used to produce a corresponding code. We provide an algorithm which on input (n, j) outputs the j-th codeword of $\mathcal{C}'_n$, the desired code, in time polynomial in n.

Let $t = \lceil \log\log\log n \rceil$, $r = \lfloor n/t \rfloor$ and $r' = n - rt < t$.

We first produce codes $\tilde{\mathcal{C}}_t$ and $\tilde{\mathcal{C}}_{r'}$ which both have property $\mathcal{P}$ and size at most $\lambda^{t+o(t)}$ and $\lambda^{r'+o(t)}$ in time polynomial in n. This is possible because $\mathfrak{C}$ is proof that there exist codes of these dimensions satisfying $\mathcal{P}$ and because $r' < t \approx \log\log\log n$, we can try all 2^{2^t} possible codes of length t until we find suitable ones in time polynomial in n.

Now we define $\mathcal{C}'_n := \left(\tilde{\mathcal{C}}_t\right)^r \times \tilde{\mathcal{C}}_{r'}$ as the cartesian product of r factors C_t and once $C_{r'}$. This produces a code of exactly length n and of size $\lambda^{rt+r'+ro(t)} = \lambda^{n+o(n)}$ as desired and by $\mathcal{P}$ being multiplicative, $\mathcal{C}'_n$ has property $\mathcal{P}$.

Without explicitly building the code, we now can output the j-th word by simply concatenating the corresponding words from the smaller codes. □

Hereafter, we consider two particular multiplicative code properties of which we will make use in our derandomization.

Covering Codes. Consider the alphabet $\Sigma = \{1..d\}$. Let, for any arbitrary $t \in [0,1]$, $\mathcal{K}(t)$ denote the property of d-ary codes which is satisfied for any code $\mathcal{C} \subseteq \Sigma^n$ for which any word $w \in \Sigma^n$ has a

codeword $w' \in \mathcal{C}$ at distance at most tn, in signs

$$\forall w \in \Sigma^n : \exists w' \in \mathcal{C} : \mathrm{dist}(w, w') \leq tn.$$

Let us call a code satisfying this property is called a *covering code* of *covering radius* t. The property is clearly multiplicative and so using Lemma 3.9, once we know the required size for such a code, there is also a constructive version of it. For the size, we establish the following.

Lemma 3.10. *For any $t \in [0, 1/2)$ and any alphabet Σ of size d, there exists a code sequence of size*

$$\lambda := d \cdot (1-t) \cdot \left(\frac{t}{(d-1)(1-t)} \right)^t$$

satisfying $\mathcal{K}(t)$.

Proof. We prove existence of $\mathcal{C}_n$ separately for every n. The existence of the sequence then follows. Now for fixed n, we produce $\mathcal{C}_n$ by picking s codewords uniformly at random with replacement. For every fixed $w \in \Sigma^n$, the probability that there exists no $w' \in \mathcal{C}$ having

$$\mathrm{dist}(w, w') \leq tn$$

is at most

$$\left(1 - \frac{\mathrm{vol}_d(n, tn)}{d^n}\right)^s \leq \exp\left(-s \cdot \frac{\mathrm{vol}_d(n, tn)}{d^n}\right),$$

where we have used Lemma A.1 and where we define

$$\mathrm{vol}_d(n, r) := |\{ w' \in \Sigma^n \mid \mathrm{dist}(w, w') \leq r \}|$$

which does not depend on $w \in \Sigma^n$. By a union bound, the probability that there is *any* $w \in \Sigma^n$ for which there exists no such w' is at most

$$d^n \cdot \exp\left(-s \cdot \frac{\mathrm{vol}_d(n, tn)}{d^n}\right).$$

This probability drops below $\frac{1}{2}$ if we set

$$s := \left\lceil 2 \cdot n \cdot \ln d \cdot \frac{d^n}{\mathrm{vol}_d(n, tn)} \right\rceil .$$

With the probability below $\frac{1}{2}$, there is a positive probability that all words are appropriately covered and thus the required code exists. Finally, we note that by Lemma A.13,

$$\mathrm{vol}_d(n, \lceil tn \rceil) = \sum_{i=0}^{\lceil tn \rceil} \binom{n}{i} (d-1)^i = \left[\frac{1}{1-t} \left(\frac{(d-1)(1-t)}{t} \right)^t \right]^{n+o(n)} .$$

□

We note that this code is optimal in the sense that for any smaller value of λ, a code satisfying $\mathcal{K}(t)$ of size λ does not exist. This follows along the same lines as the lemma: each codeword has $\mathrm{vol}_d(n, r)$ words at distance at most r and thus any code of a smaller size must leave some words for which the closest codeword lies farther away.

For later use, let us denote by $\texttt{CoveringCode}(d, t, n, j)$ an algorithm which produces a d-ary covering code of length n satisfying $\mathcal{K}(t)$ by outputting on demand the j-th codeword.

Next, we investigate code sequences which we need in order to establish Lemma 3.4 of which we still owe the proof.

Box Coverings. Consider the alphabet

$$\Sigma = \binom{\{1..d\}}{2} .$$

Let $\mathcal{L}$ denote the property of codes over Σ which is satisfied for any code $\mathcal{C} \subseteq \Sigma^n$ such that for every word $w \in \{1..d\}^n$, there exists a codeword $w' \in \mathcal{C}$ which covers that word, i.e.

$$\forall w \in \{1..d\}^n : \exists w' \in \mathcal{C} : \forall i \in \{1..n\} : w_i \in w'_i.$$

We call a code satisfying this property a *box covering* of the d-ary space. The property is clearly multiplicative and so using Lemma 3.9, once we know the required size for such a code, there is also a constructive version of it. For the size, we establish the following.

Lemma 3.11. *There exists a code sequence of size $d/2$ satisfying $\mathcal{L}$.*

Proof. For every n, we provide a randomized construction for the code $\mathcal{C}_n$. From this, existence of the sequence follows. To build $\mathcal{C}_n$, we pick s codewords over Σ uniformly at random with replacement. For any fixed $w \in \{1..d\}^n$, the probability that a random word of length n over Σ covers w is $(2/d)^n$. The probability that w remains uncovered after picking s words is therefore

$$\left(1 - \left(\frac{2}{d}\right)^n\right)^s \leq \exp\left(-s \cdot \left(\frac{2}{d}\right)^n\right)$$

and the expected total number of uncovered words is

$$d^n \cdot \exp\left(-s \cdot \left(\frac{2}{d}\right)^n\right).$$

This drops below one half if we set

$$s := \left\lceil 2 \cdot n \cdot \ln d \cdot \left(\frac{d}{2}\right)^n \right\rceil,$$

so that a code of the claimed size exists. □

Lemma 3.4 now follows as a simple corollary from Lemma 3.11 and Lemma 3.9.

3.7 Deterministic Local Search[11]

We now go on to prove the following.

[11] This is joint work with Dominik Scheder and has appeared in [MS11].

Theorem 3.12. *There exists a deterministic algorithm which on input a satisfiable k-CNF formula F, outputs a satisfying assignment in time*

$$\left(2 \cdot \frac{k-1}{k}\right)^{n+o(n)}.$$

This matches the performance of randomized Schöning from Theorem 3.3. Recall that via Lemma 3.4, this running time translates to a lossless derandomization for the (d,k)-ClSP case as well.

Corollary 3.13. *There exists a deterministic algorithm which on input a satisfiable (d,k)-ClSP formula F, outputs a satisfying assignment in time*

$$\left(d \cdot \frac{k-1}{k}\right)^{n+o(n)}.$$

Our derandomization builds closely on previous work. We will now first explain the key idea of how Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan, and Schöning [DGH$^+$02] provided the first deterministic variant of Algorithm 5.

The Approach by Dantsin et al. In Section 3.5, we have identified 'typical behavior' of the randomized variant when run on a k-CNF formula. One of the discoveries we made was that there is a radius at around n/k such that if we consider only those executions a 'success' which start at an initial assignment α_0 at distance exactly n/k from the fixed satisfying assignment α^* and dismiss all others, the overall success probability of the algorithm decreases only by a negligible, subexponential factor. Hence it suffices to consider such executions.

The intuition that the same thing should be done in a deterministic version led to the algorithm due to Dantsin et al. [DGH$^+$02] (see Algorithm 7). It builds a covering code of radius $1/k$ and tries using each of its codewords as initial assignment. From Lemma 3.10 in conjunction

Algorithm 7 `SchoeningDeterministic`(F)

Require: A satisfiable k-CNF formula F.
Ensure: Output is a satisfying assignment.

```
1: j ← 1
2: while codeword α ← CoveringCode(2, 1/k, n, j) exists do
3:    β ← SearchBall(F, α, n/k)
4:    if β satisfies F then
5:       return β
6:    end if
7:    j ← j + 1
8: end while
```

with Lemma 3.9, we know that an efficiently constructible such code having at most

$$\left[2 \cdot \left(1 - \frac{1}{k}\right) \cdot \left(\frac{\frac{1}{k}}{(1-\frac{1}{k})}\right)^{\frac{1}{k}}\right]^{n+o(n)} \tag{3.2}$$

codewords exists. When starting a search from each of them, we are sure that in *one* of the attempts, the satisfying assignment α^* is at distance at most n/k from the α_0 so chosen. This approach is *optimal* in the sense that the number of steps (number of codewords) we need matches the inverse of the probability which the randomized algorithm has to fetch a starting assignment at distance n/k, up to negligible, subexponential factors (see discussion after Lemma 3.10.

What we now additionally need is a suitable deterministic procedure `SearchBall` which can, starting from such an assignment, search the ball of radius n/k around α_0 for a satisfying assignment. Apart from a brute force enumeration ignoring the formula F, the second most simplistic way of doing this is summarized in Algorithm 8: unless the center of the ball we are searching is itself a satisfying assign-

ment, we search in the neighborhood, thereby ignoring places which are clearly unsatisfying; in each step we pick a clause $C \in F$ which is currently violated so that we know that one of its k variables must receive a different value in α^* and we try flipping each of them, one after the other. In at least one of these attempts, we are sure to be one step closer to α^* and thus we recursively invoke the procedure to search the ball with a decreased radius centered at the modified assignment.

One detail of `SearchBall` is interesting to note. For the recursive invocation, not only does it change the value of the flipped literal u_i within assignment α' to be used as the new center of the ball to search, but also it *assigns*, permanently, the new value to u_i by substitution into the formula F, producing a simplified formula F'. With the simple search strategy applied here, this is legitimate and *could* save some time: once we decide that flipping the literal u_i is the right choice, then within this branch of the execution there is no reason for us to ever flip it back. Removing the variable from the formula makes the formula smaller and prevents us from selecting the same variable for flipping on deeper recursion levels.

What is the running time of `SearchBall`? Clearly, if we call it for a ball of radius r, then it invokes itself recursively for radius $r-1$ at most k times as the largest clauses in the formula contain no more than k literals. On each recursive level, computations done are polynomial. Therefore, the total running time amounts to $k^{r+o(n)}$. In contrast to the choice of initial assignment, *this* is *not optimal* because, if we do the calculation (see Section 3.5), then the 'ball searching part' of the randomized algorithm has a success probability of $(k-1)^{-r-o(n)}$. In order to derandomize it losslessly, the deterministic procedure would need to exhibit the same performance.

In total, there are as many top-level invocations for $r = n/k$ as there are words in our covering code used for the initial assignments (see expression (3.2)). Multiplying this by the running time $k^{r+o(n)}$ per invocation, we obtain the following.

Theorem 3.14. *Algorithm 7 finds a satisfying assignment to any satisfiable k-CNF formula on n variables in deterministic time*

$$\left[2\left(\frac{k-1}{k}\right)^{\frac{k-1}{k}}\right]^{n+o(n)}.$$

In comparison to the randomized version, there is a loss which manifests as an additional exponent smaller than one in the factor representing the algorithm's advantage over a brute force approach. For 3-SAT, this means a running time of $\mathcal{O}(1.527^n)$.

Dantsin et al. [DGH+02] have improved on this time by a variety of means. First and foremost, a simple observation is that as long as we stick to a ball searching subroutine which is slower than optimal, n/k is not the right radius to choose for the covering code. The number n/k was the result of the analysis of random executions and it is the point where the success probability peaks *given that* the iterative local search is optimal[12]. By optimizing the covering code radius, they decreased the running time to $\mathcal{O}(1.5^n)$.

As a next step, Dantsin et al. observed that there are improvements possible over the procedure `SearchBall`. For a vague idea, recall the more old-fashioned splitting and branching rule type of algorithms at which we had a short look in Section 3.1. Some ideas of this sort carry over to the splitting and branching done within the search ball procedure. Following up Dantsin et al., the step-wise improvements by Brueggemann and Kern [BK04], Scheder [Sch08] and then Kutzkov and Scheder [KS10] all based on more and more sophisticated branching rules within the `SearchBall` routine.

However sophisticated these advancements, the principle of splitting and branching has its inherent limits. Arguably, the problem with

[12] 'Optimal' here always means 'matching the performance of the randomized variant' as analysed during the discussion of typical executions.

it is the fact that on each flip, variables are assigned values *for good* rather than in a way which can be reversed if necessary. And as we have seen in our investigation of typical executions in Section 3.5, allowing for wrong flips and (re-)correcting them later is *exactly* what gives Schöning's algorithm its full power.

We will now devise a ball searching strategy which resembles the randomized variant much more closely and which we will prove to run in optimal[13] time $(k-1)^{r+o(n)}$. Note that we will be done once we manage to do this, because multiplying this running time by the number of codewords used as initial assignments given in (3.2) then readily yields the theorem.

Few Large Clauses. On the way there, the first observation is that in some special cases, the simple procedure `SearchBall` already exhibits this performance. Suppose for example that the $(\leq k)$-CNF formula F handed over and the current assignment α are such that *all* clauses violated by α have $k-1$ or less literals. As the procedure selects a smallest violated clause and branches on each of its literals, this incurs at most $k-1$ recursive invocations. Furthermore, since, and *here* this becomes vital, `SearchBall` does not only modify the current assignment α but *substitutes* values for the variables permanently, the only *newly* violated clauses which can appear as a result of such a substitution will *again* feature at most $k-1$ literals. So if the formula has *only* violated clauses with at most $k-1$ literals, then this property will be forever preserved along the recursive calls. We have thus demonstrated the following.

Lemma 3.15. *If F is an $(\leq k)$-CNF formula and α is an assignment such that all clauses in F violated by α have at most $k-1$ literals, then the procedure `SearchBall` (F, α, r) runs in time $(k-1)^{r+o(n)}$.*

[13] Again, optimal means that it matches the performance of its randomized counterpart. We do of course not claim there is no better procedure.

If an $(\leq k)$-CNF formula F has *some* violated k-clauses but not too 'many' in a sense to be made more precise, then one can make sure that they are being treated and made to disappear. To that end, the procedure `SearchBallOrLargeIS` takes F, α and r like `SearchBall`, plus an additional parameter $t \in \mathbb{N}$. If the ball of radius r around α contains a satisfying assignment, it outputs one of two things: *either* a satisfying assignment, *or* an independent set of at least t many k-clauses in F which are all violated by α, so as to 'justify' why a standard ball search will not be efficient on this input.

Let us determine the running time of this procedure.

Lemma 3.16. *If `SearchBallOrLargeIS` outputs an independent set of at least t clauses, it does so in polynomial time. If it outputs a satisfying assignment or 'no', then this takes up to $2^{kt} \cdot (k-1)^{r+o(n)}$ time.*

Proof. The first part is clear as a maximal independent set can be generated greedily and if one of the required size is found, then it is immediately returned.

Only otherwise does the algorithm iterate over all assignments over the variables occurring in the maximal independent set M exhaustively and call standard `SearchBall` each time. These invocations are on pairs F and α such that each clause in F violated by α contains at most $k-1$ literals, as everything else would be a contradiction to the maximality of M. Therefore, according to Lemma 3.15, each invocation of the latter takes $(k-1)^{r+o(n)}$ time at most. And the number of times we try is exactly 2^{kt} as we are iterating over all assignments to the variables occurring in M of which there are kt many. □

The algorithm `SearchBallOrLargeIS` conducts its actual search exclusively in those cases where we know the running time will be optimal. Otherwise, we obtain a large independent set of currently violated k-clauses much like a 'justification' why a standard ball search

will not be efficient on this input. And in fact, if we see a large independent set of currently violated k-clauses, then we can instead do something smarter much more closely inspired by the randomized model.

Reversible Flips. We flip back a few pages to have another look at Theorem 3.8, which we wanted to use as a guide to derandomizing Schöning. It tells us that typical executions leading to success start at an initial assignment at distance n/k and then proceed very regularly towards the target assignment, where 'regularly' means that for any slowly growing function[14] $t(n)$, among every $t(n)$ iterations, roughly a $1/k$-fraction results in 'bad' flips leading us away from α^* while the remaining flips are 'good' flips which compensate for the bad flips and take us towards the solution.

We properly put the first half in place: the deterministic search for a starting assignment at distance n/k is settled. It is now time to take care of the other part. For some slowly growing function $f(n)$, we want to make 'phases' of $f(n)$ flips each. And in each phase, we want to be sure to be making at most $1/k$ mistakes. It seems we can *no longer* allow values to be substituted for variables for good. Instead, we have to proceed as the randomized variant, changing, possibly repeatedly, assignments in α. But how to conduct the flips deterministically?

The first key idea is to use *independent sets of clauses*. If we want to proceed in phases of $f(n)$ flips each and we want to make these flips all *at once* instead of one after the other, then we should select for the purpose an independent set of $f(n)$ clauses which are all currently violated. In such an independent set, we can decide on one of the literals in each clause to be flipped independently of what we are doing in the

[14] Note that we use $t = t(n)$ as in Theorem 3.8 for the number of evolution steps per phase for consistency. As $f(n)$ was allowed to be any slowly growing function for the theorem, so is $t(n)$.

other clauses. This is not normally possible: if the clauses are selected one after the other and arbitrarily, then the question which clause will be fixed next may very well depend on which literal we flip in the previous one. We do not have to be concerned about *finding* a set of $f(n)$ clauses all of which are currently violated, as we already have the procedure `SearchBallOrLargeIS` which will either provide us with one or conduct the search optimally.

Now suppose that we receive an independent set $M \subseteq F$ of t clauses of size k all of which are currently violated by α. Let us call them

$$M = \{C_1, C_2, \ldots, C_{f(n)}\}$$

and call the literals they contain

$$C_i = \{u_{i1}, u_{i2}, \ldots, u_{ik}\},$$

for all i. We then select the first $f(n)$ clauses, C_1 through $C_{f(n)}$, for simultaneous fixing in this phase. Note that since α^* satisfies all of these clauses, it satisfies at least one of the literals in each clause. Denote by ζ_i for all $1 \leq i \leq f(n)$ the index of the first literal in C_i satisfied by α^*. Note that we have done the very same thing for the analysis of the randomized version: if we now pick values $X_1, \ldots, X_{f(n)} \in \{1 \ldots k\}$ uniformly at random, we have a probability of $1/k$ in each clause of flipping the designated ζ_i-th literal which will take us one step closer to α^*. And as our analysis of typical executions has shown, success probability peaks in cases where this happens in all but $1/k$-fraction of the clauses. To do the same thing deterministically, we should now find values $x_1, \ldots, x_{f(n)} \in \{1..k\}$ such that for at most $1/k$-fraction of the indices i, $x_i \neq \zeta_i$ disagree. As we can see, this is nothing else but yet again a covering code. This time not a code to cover the solution space with assignments, but a code to cover the space of possible such 'flipping patterns'. Using Lemma 3.10, there exists a code of k-ary words and covering radius $1/k$, i.e. with the property that for each possible string $\zeta_1\zeta_2 \ldots \zeta_{f(n)}$ which we have to be prepared for, there

exists a word $w_1 w_2 \dots w_{f(n)}$ such that all but a $1/k$-fraction of the digits agree with it and which has at most

$$\left[k \cdot \left(1 - \frac{1}{k}\right) \cdot \left(\frac{\frac{1}{k}}{(k-1)(1-\frac{1}{k})}\right)^{\frac{1}{k}}\right]^{f(n)+o(f(n))} = \left[(k-1)^{\frac{k-2}{k}}\right]^{f(n)+o(f(n))}.$$

codewords. For one of these codewords, we are sure that it has only $1/k$-fraction of 'mistakes' and will therefore take us at least

$$\frac{k-2}{k} \cdot f(n)$$

steps closer to α^*. Note that the performance is optimal: overcoming any distance r takes an effort of $(k-1)^{r+o(n)}$ branches we have to explore. The procedure is summarized as `SearchBallFast` in Algorithm 10.

Lemma 3.17. *`SearchBallFast` runs in time* $(k-1)^{r+o(n)}$.

Proof. According to Lemma 3.16, either the invocation of our subprocedure `SearchBallOrLargeIS` may take polynomial time and yield an independent set of t clauses violated by α, or it may take $2^{kt}(k-1)^{r+o(n)}$ time and return either a satisfying assignment or 'no'. In the latter case, the satisfying assignment or the failure are immediately returned and thus the total running time is $(k-1)^{r+o(n)}$ if we take into account that for $t = \log\log n$, the factor 2^{kt} can be subsumed within the negligible error term.

In the former case, the procedure issues a number of recursive invocations which equals the size of the k-ary flipping pattern covering code which we know is at most

$$\left[(k-1)^{\frac{k-2}{k}}\right]^{t+o(t)}$$

each of which is left with searching a ball of radius

$$r - \frac{k-2}{k} \cdot t.$$

We can assume that the smaller ball is being searched in time

$$(k-1)^{r-\frac{k-2}{k}\cdot t+o(n)}$$

as an induction hypothesis. Multiplying this time with the bound on the number of invocations thus yields

$$\left[(k-1)^{\frac{k-2}{k}}\right]^{t+o(t)} (k-1)^{r-\frac{k-2}{k}\cdot t+o(n)} = (k-1)^{r+o(n)},$$

as claimed. □

Plugging this faster variant into Algorithm 7 finally proves Theorem 3.12.

It is noteworthy that our deterministic algorithm does not only match the randomized original in terms of time-performance but also in terms of space consumption: both versions use only polynomial space. While trivial in the randomized case, in the deterministic case this follows from the fact that our covering codes are efficiently constructible (see Section 3.6), i.e. that they can be enumerated without being stored explicitly. The recursive local search itself clearly runs in polynomial space, storing only one call stack at a time.

Algorithm 8 `SearchBall`(F, α, r)

Require: A satisfiable $(\leq k)$-CNF formula F, some assignment α and a radius $r \in \{0..n\}$.

Ensure: If there exists a satisfying assignment α' at distance $\mathrm{dist}(\alpha', \alpha) \leq r$, then the output is *one such* satisfying assignment; otherwise the output is arbitrary.

1: **if** α satisfies F **then**
2: **return** α
3: **else**
4: **if** $r = 0$ **then**
5: **return** 'no'
6: **else**
7: $\{u_1, u_2, \ldots, u_{k'}\} = C \leftarrow$ a smallest clause in F violated by α
8: **for** $i = 1, 2, \ldots, k'$ **do**
9: $\alpha' \leftarrow \alpha[u_i \mapsto 1]$
10: $F' \leftarrow F^{[u_i \mapsto 1]}$
11: rv $\leftarrow$ `SearchBall`$(F', \alpha', r-1)$
12: **if** rv $\neq$ 'no' **then**
13: **return** rv
14: **end if**
15: **end for**
16: **return** 'no'
17: **end if**
18: **end if**

Algorithm 9 `SearchBallOrLargeIS`(F, α, r, t)

Require: A satisfiable $(\leq k)$-CNF formula F, some assignment α, a radius $r \in \{0..n\}$ and a number $t \in \mathbb{N}$.

Ensure: If there exists a satisfying assignment α' at distance $\mathrm{dist}(\alpha', \alpha) \leq r$, then output is *either* an independent set of clauses violated by α of size at least t or *some* satisfying assignment; otherwise output is arbitrary.

1: $M \leftarrow$ a maximal independent set of k-clauses violated by α
2: **if** $|M| \geq t$ **then**
3: **return** first t clauses of M
4: **else**
5: **for** all assignments $\beta : \cup_{C \in M} \mathrm{vbl}(C) \to \{0,1\}$ **do**
6: $\alpha \leftarrow$ `SearchBall`$(F^{[\beta]}, r, \alpha[\beta])$
7: **if** α satisfies F **then**
8: **return** α
9: **end if**
10: **end for**
11: **return** 'no'
12: **end if**

Algorithm 10 `SearchBallFast`(F, α, r)

Require: A satisfiable $(\leq k)$-CNF formula F, some assignment α and a radius $r \in \{0..n\}$.

Ensure: If there exists a satisfying assignment α' at distance $\text{dist}(\alpha', \alpha) \leq r$, then the output is *some* satisfying assignment; otherwise the output is arbitrary.

$t \leftarrow \lceil \log \log n \rceil$
$(M, \alpha) \leftarrow$ `SearchBallOrLargeIS` (F, α, r, t)
if α satisfies F **then**
  return α
else
  if M is independent set of t violated clauses $C_1, C_2, \ldots, C_t$ **then**
    $j \leftarrow 1$
    while codeword $x \leftarrow$ `CoveringCode`$(k, \frac{1}{k}, t, j)$ **do**
      $\beta \leftarrow \alpha$ with the x_i^{th} variable of C_i flipped for all $1 \leq i \leq t$
      $\gamma \leftarrow$ `SearchBallFast`$\left(F, \alpha, r - \frac{k-2}{k} \cdot t\right)$
      if γ satisfies F **then**
        return γ
      end if
      $j \leftarrow j + 1$
    end while
  end if
  return 'no'
end if

A

Auxiliary Statements and Deferred Proofs

The present chapter contains auxiliary statements of a mostly analytic nature and all proofs we have deferred from the main part because they are merely technical rather than enlightening.

A.1 Estimates involving the exponential series

Lemma A.1. *For all $x \in \mathbb{R}$, $1 + x \leq e^x$. Moreover, for $x \geq 0$, we have $1 + x + \frac{1}{2}x^2 \leq e^x$.*

Proof. The weaker statement is very well-known and does not need a proof. For the more precise statement in the case $x \geq 0$, considering the definition of the right hand side series,

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

the claim follows from the fact that all terms are positive. □

Lemma A.2. *Let $\beta : (1, \infty) \to (0, 1)$ be defined as*

$$\beta(\alpha) := 1 + \frac{1}{\alpha} \cdot W(-\alpha \cdot e^{-\alpha}),$$

where W is the Lambert-W function (see, e.g., [CGH+96] for a discussion of this function), i.e. $W(z)$ is the unique solution $t \in \mathbb{R}$ to the equation $z = te^t$. Let $\alpha > 1$. Then for all $x \in [0, \beta(\alpha)]$, $1 - x \geq e^{-\alpha x}$.

Proof. $e^{-\alpha x}$ is clearly a convex function, hence it can intersect the linear function $1 - x$ at most twice. One intersection point is at $x = 0$ and there is one more intersection point at $x = \beta(\alpha)$, as we can easily check by plugging in the definition of β and rearranging terms. The inequality thus holds for all points between the two intersections. □

Lemma A.3. *For all $x > 1$, we have*

$$\left(1 - \frac{1}{x}\right)^{x-1} \geq 1/e.$$

Moreover, for sufficiently large x,

$$\left(1 - \frac{1}{x}\right)^{x-1} \geq \frac{1}{e} + \frac{1}{2e} \cdot \frac{1}{x}.$$

Proof. To prove the simpler first statement, we invoke Lemma A.1 to obtain

$$\left(1 + \frac{1}{x-1}\right) \leq e^{\frac{1}{x-1}}.$$

Taking the $(1-x)$-th power on both sides and simplifying the sum yields the claim.

For the more difficult second claim, we introduce the variable substitution $\tau := 1/x$ and define the function $f : [0,1) \to \mathbb{R}$ as

$$f(\tau) := \begin{cases} (1-\tau)^{\frac{1}{\tau}-1} & \text{if } \tau > 0 \\ 1/e & \text{otherwise.} \end{cases}$$

It suffices to prove that for $\tau \to 0$, we have

$$f(\tau) = \frac{1}{e} + \frac{1}{2e} \cdot \tau + \frac{7}{24e} \cdot \tau^2 + \mathcal{O}(\tau^3), \tag{A.1}$$

because this leads to the conclusion that for sufficiently small τ, the cubic error term is dominated by the quadratic term whose coefficient is positive. In order to obtain the Taylor series in (A.1), we require that

(i) $f(\tau)$ is continuous and three times continuously differentiable,
(ii) $\lim_{\tau\to 0} f(\tau) = 1/e$,
(iii) $\lim_{\tau\to 0} f'(\tau) = 1/(2e)$,
(iv) $\lim_{\tau\to 0} f''(\tau) = 7/(12e)$ and
(v) $\lim_{\tau\to 0} f'''(\tau)$ exists.

Once we have established these properties, (A.1) follows by developing the Taylor series (see, e.g., [For06], page 233).

Property (i) follows from the other properties and the definition of f. Note that we are only interested in right-sided derivatives at $\tau = 0$ and if the limits in (ii), (iii) and (iv) exist, continuous differentiability cannot be jeopardized by the fact that we get rid of a removable singularity. It remains to prove (ii) through (v).

For (ii), we note that

$$f(\tau) = (1-\tau)^{\frac{1}{\tau}} \cdot \frac{1}{1-\tau},$$

where the first factor is well-known to converge to $1/e$ and the second one converges to one.

To see (iii), we differentiate f to get, for all $\tau \in (0,1)$,

$$f'(\tau) = f(\tau) \cdot \underbrace{\left(\frac{-(1-\tau)\ln(1-\tau) - \tau + \tau^2}{\tau^2 - \tau^3} \right)}_{=:\varphi(\tau)}.$$

The limit of $\varphi(\tau)$ can be seen to be $1/2$ by applying the rule of de l'Hôpital twice (see, e.g., [For06], page 162). Since the limit of $f(\tau)$ is $1/e$ by (ii), the claim follows.

To see (iv), we differentiate again to get, for all $\tau \in (0,1)$,

$$f''(\tau) = f'(\tau)\varphi(\tau) + f(\tau)\varphi'(\tau).$$

We already have the limits of f, f' and φ. What we are still missing is φ'. We obtain that

$$\varphi'(\tau) = \frac{\tau^2 - 2\tau + (2\tau - 2)\ln(1-\tau)}{\tau^4 - \tau^3}.$$

Again applying de l'Hôpital, we find that $\varphi'(\tau)$ converges to $1/3$. Plugging this in above, we find that f'' converges to $1/(4e) + 1/(3e)$, as claimed.

It remains to do another step to get (v). We obtain that

$$f'''(\tau) = f''(\tau)\varphi(\tau) + 2f'(\tau)\varphi'(\tau) + f(\tau)\varphi''(\tau),$$

where we know all limits except for $\varphi''(\tau)$. Differentiate

$$\varphi''(\tau) = \frac{12\tau\ln(1-\tau) - 6\ln(1-\tau)\tau^2 + 9\tau^2 - 2\tau^3 - 6\ln(1-\tau) - 6\tau}{\tau^4(\tau-1)^2}$$

De l'Hôpital yields that φ'' converges to $1/2$ which finally implies the last claim. □

A.2 Cumulation of multiplicative slacks

Lemma A.4. *Let $\epsilon_0, \epsilon_1 \in [0,1]$. Then*

$$1 + \epsilon_0 + \epsilon_1 \leq (1+\epsilon_0)(1+\epsilon_1) \leq 1 + 2\epsilon_0 + \epsilon_1.$$

Moreover, if $\epsilon_0 \leq \epsilon_1/2$, then

$$(1-\epsilon_0)(1+\epsilon_1) \geq 1.$$

Proof. We have

$$(1+\epsilon_0)(1+\epsilon_1) = 1 + \epsilon_0 + \epsilon_1 + \epsilon_0\epsilon_1$$

and the first claim follows from $\epsilon_1 \leq 1$. Moreover, we have

$$(1-\epsilon_0)(1+\epsilon_1) \geq (1-\frac{\epsilon_1}{2})(1+\epsilon_1) = 1 + \frac{\epsilon_1}{2} \cdot (1-\epsilon_1) \geq 1,$$

as claimed. □

A.3 Bounded away from one

Lemma A.5. *There exists a universal constant Λ, e.g. $\Lambda = 0.999$ will do, such that the following holds. Let G be any simple loopless graph on the vertices $\{1..m\}$. Consider the mapping*

$$\Xi \ : \ (0,1)^m \to (0,1)^m \ : \ x \mapsto \left\{ x_j \prod_{i \in \Gamma_G(j)} (1-x_i) \right\}_{j \in \{1..m\}}$$

where $\Gamma_G(j)$ are the neighbors of vertex j in G. For any vector $x \in (0,1)^m$, there exists another vector $x' \in (0,1)^m$, with the property that for each $1 \leq i \leq m$, we have $\Xi(x')_i \geq \min\{\frac{1}{2}, \Xi(x)_i\}$ and such that all components of x' are at most Λ.

Proof. We proceed by exhibiting a chain of vectors

$$x =: x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(r)} =: x' \in (0,1)^m,$$

all satisfying the conclusion and such that the set of very large numbers, formally

$$Z_i := \{j \in \{1..m\} \mid x_j^{(i)} \geq \Lambda\},$$

shrinks by at least one element, where we let $\Lambda := 0.999$.

To get from $x^{(i)}$ to $x^{(i+1)}$, we select an entry $z_i \in Z_i$ to work on.

Suppose z_i is an isolated vertex in G, then set

$$x_j^{(i+1)} = \begin{cases} x_j^{(i)} & \text{if } j \neq z_i \\ \frac{1}{2} & \text{if } j = z_i. \end{cases}$$

In this case it is easy to see that $x^{(i+1)}$ satisfies the hypothesis: component number j does not influence any other value than the j-th, and there we will have $\Xi(x^{(i+1)}) = x_j^{(i+1)} = \frac{1}{2}$, as required.

Suppose z_i has a non-empty set $\Gamma_G(z_i)$ of neighbors. Then let us decrease the value for z_i and for all its neighbors simultaneously and assign

$$x_j^{(i+1)} = \begin{cases} \frac{9}{10} & \text{if } j = z_i \\ \frac{1}{100} x_j^{(i)} & \text{if } j \in \Gamma_G(z_i) \\ x_j^{(i)} & \text{otherwise.} \end{cases}$$

Clearly, this yields at least $Z_{i+1} \subseteq Z_i \setminus \{z_i\}$ such that we make the desired progress. Now we must establish that $x^{(i+1)}$ still satisfies the induction hypothesis. First note that the values which can have changed are the ones at the components $R_i := \{z_i\} \cup \Gamma_G(z_i)$ and all input values have only decreased in comparison to $x^{(i)}$. Therefore, the output values under Ξ at components outside of R_i have only increased which is harmless. We only have to care about the components in R_i which are

influenced by both increasing and decreasing factors. We prove that the induction hypothesis is still satisfied for any fixed $j \in R_i$.

Consider the case $j \in \Gamma_G(z_i)$ first. In this case we have $z_i \in \Gamma_G(j)$ by symmetry such that

$$\Xi(x^{(i+1)})_j = x_j^{(i+1)} \cdot \prod_{k \in \Gamma_G(j)} \left(1 - x_k^{(i+1)}\right) =$$

$$= x_j^{(i+1)} \cdot \left(1 - x_{z_i}^{(i+1)}\right) \cdot \prod_{k \in \Gamma_G(j) \setminus \{z_i\}} \left(1 - x_k^{(i+1)}\right) \geq$$

$$\geq \frac{x_j^{(i)}}{100} \cdot \underbrace{\left(1 - \frac{9}{10}\right)}_{=0.1 \geq 100\left(1 - x_{z_i}^{(i)}\right)} \cdot \prod_{k \in \Gamma_G(j) \setminus \{z_i\}} \left(1 - x_j^{(i)}\right) \geq \Xi(x_j^{(i)}).$$

Now consider the case $j = z_i$. By construction, we have

$$\forall k \in \Gamma_G(j) : x_k^{(i+1)} < \frac{1}{100}.$$

Therefore we can apply Lemma A.2 with $\alpha = 2$ to find that

$$\rho' := \prod_{k \in \Gamma_G(z_i)} (1 - x_k^{(i+1)}) \geq \underbrace{\exp\left(-2 \sum_{k \in \Gamma_G(z_i)} x_k^{(i+1)}\right)}_{\exp\left(-\frac{2}{100} \sum_{k \in \Gamma_G(z_i)} x_k^{(i)}\right)}.$$

For comparison, we have, using Lemma A.1,

$$\rho := \prod_{k \in \Gamma_G(z_i)} (1 - x_k^{(i)}) \leq \exp\left(-\sum_{k \in \Gamma_G(z_i)} x_k^{(i)}\right),$$

and hence $\rho' \geq \rho^{2/100}$. Now either $\rho \geq 5/9$ in which case $\rho' > 5/9$ and thus

$$\Xi(x^{(i+1)})_{z_i} = x_{z_i}^{(i+1)} \cdot \rho' \geq \frac{9}{10} \cdot \frac{5}{9} = \frac{1}{2},$$

proving the claim. Or $\rho < 5/9$ in which case $\rho^{2/100} \geq \frac{10}{9}\rho$ and thus

$$\Xi(x^{(i+1)})_{z_i} = x_{z_i}^{(i+1)} \cdot \rho' \geq \frac{9}{10} \cdot \rho^{2/100} \geq \frac{9}{10} \cdot \frac{10}{9} \cdot \rho = \rho > \Xi(x^{(i)})_{z_i},$$

establishing the claim. □

A.4 Exponential Tail Estimate

Lemma A.6. *Let $\epsilon \in (0,1)$ and $C \in \mathbb{R}$ be constants and T an integral non-negative random variable such that for all $t \in \mathbb{N}$,*

$$Pr[T \geq t] \leq C \cdot (1-\epsilon)^t,$$

then

$$\mathbb{E}[T] \leq \frac{1}{\epsilon}(\ln C + 2).$$

Proof. Let t_0 be the smallest positive integer such that

$$C \cdot (1-\epsilon)^{t_0} < 1.$$

Using – from Lemma A.1 – that

$$C \cdot (1-\epsilon)^{t_0} < Ce^{-\epsilon t_0},$$

we have that

$$t_0 \leq \frac{1}{\epsilon} \ln C + 1.$$

Together with the hypothesis, this entails

$$\begin{aligned} \mathbb{E}[T] &= \sum_{t \geq 0} \Pr[T \geq t] \leq \sum_{0 \leq t < t_0} \Pr[T \geq t] + \sum_{t_0 \leq t} C(1-\epsilon)^t \\ &\leq t_0 + C(1-\epsilon)^{t_0} \sum_{0 \leq t} (1-\epsilon)^t < t_0 + \frac{1}{\epsilon} < \frac{1}{\epsilon}(\ln C + 2), \end{aligned}$$

as claimed. □

A.5 Certain Homogeneous Markov Chains[1]

A *Markov Chain on the integers* is a random process $\{E_i \in \mathbb{Z}\}_{i\in\mathbb{N}_0}$ with the property (*Markov property*) that E_{i+1} depends only on E_i and is independent of the history of the process. In the sequel, we analyze a very particular kind of such chain where $E_{i+1} - E_i$ is independent of i and of E_i and is given as a fixed vector of transition probabilities. This kind of chain is used in [Sch99] for the analysis of Schöning's algorithm. We generalize it somewhat here.

Lemma A.7. *Let $k \geq 2$ be some integer and*

$$\tau : \{-1, 0, 1, 2, \ldots, k-1\} \to [0, 1]$$

be a $(k+1)$-tuple of probabilities which sum up to one, where $\tau(-1) > 0$ and which have the bias

$$b_\tau := \frac{\tau(-1)}{1 - \tau(0)} < \frac{1}{2}.$$

There exists a constant C depending only on τ such that the following holds. Let $\{E_i \in \mathbb{Z}\}_{i\in\mathbb{N}_0}$ be a Markov Chain such that $E_0 = j$ and then

$$\forall i \geq 1 : \forall t \in \{-1, 0, 1, 2, \ldots, k-1\} : \; Pr\left[E_i = E_{i-1} + t\right] \; = \; \tau(t).$$

For such a chain,

$$Pr\left[\exists i \leq Cj : E_i = 0\right] = \lambda^{j+o(j)}$$

holds, where λ is the unique root in $(0, 1)$ of the polynomial

$$p(x) := -x + [\tau(-1) + \tau(0)x + \tau(1)x^2 + \ldots + \tau(k-1)x^k].$$

[1] The calculation presented here is joint work with Andrei Giurgiu and has appeared in his Master's thesis [Giu09]. Most probably however, it is also common knowledge among experts on Markov chains and random processes.

Let, for all $j \in \mathbb{N}_0$,

$$p_j := \Pr\left[\exists i \in \mathbb{N}_0 : E_i = 0\right]$$

denote the probability that when starting from state $E_0 = j$, state zero is reached in any finite number of steps. We first claim the following.

Lemma A.8. *p_{j+1}/p_j is constant for all j.*

Proof. This is a consequence of the homogeneous nature of the chain. Note that any *path*, i.e. any fixed succession of increasing and decreasing steps, has the same probability of being realized no matter which starting state we select. Now note that p_1 is the sum of the probabilities of all paths which start at one and hit zero at the end for the first time. It is the same set of paths which start at state $j+1$ and hit state j at the end for the first time. Therefore, the probability that when starting from state $j+1$ we ever hit state j is exactly p_1 and thus the probability that we ever hit zero when starting from state $j+1$ is $p_1 \cdot p_j$. The ratio between any two consecutive p_j and p_{j+1} is therefore constantly p_1. □

Lemma A.9. $0 < p_1 < 1$.

Proof. The first inequality is easy as it suffices to exhibit any path with positive probability leading to the zero state. Indeed, there is a probability of at least $\tau(-1) > 0$ that we make one decreasing step and thus arrive at zero.

To see the second inequality, we proceed by contradiction. Suppose $p_1 = 1$. By Lemma A.8, this would imply that $p_j = 1$ for all j. Starting from a state j and reaching zero within N non-resting (increasing or decreasing) evolution steps requires that more than half of these N steps are decreasing steps which have a probability of

$$\frac{\tau(-1)}{(1-\tau(0))} < \frac{1}{2}$$

by the biassing condition. Using Chernoff, the probability that this happens is at most e^{-cN} for a suitable constant c depending only on τ. Now consider some sufficiently large j having

$$j > \frac{1}{c}\left[1 - \ln(1 - e^{-c})\right].$$

Since reaching zero requires $N > j$, the probability that we can reach zero within an arbitrary number of steps becomes

$$p_j \;\leq\; \sum_{N=j}^{\infty} e^{-cN} \;\leq\; e^{-1+\ln(1-e^{-c})} \sum_{N=0}^{\infty} e^{-cN} =$$

$$= \frac{1}{e} \cdot \left(1 - e^{-c}\right) \cdot \frac{1}{1 - e^{-c}} = \frac{1}{e},$$

a contradiction. □

We now know that $p_j = p_1^j$ for all j. It is left to determine p_1.

Lemma A.10. *$p_1 = \lambda$, the unique root of $p(x)$ in the range $(0, 1)$.*

Proof. Each p_j must obviously satisfy the simple recurrence

$$p_j = \tau(-1)p_{j-1} + \tau(0)p_j + \tau(1)p_{j+1} + \ldots + \tau(k-1)p_{j+k-1}.$$

Since we know the solution has the form $p_j = p_1^j$ we plug this in and obtain

$$0 = p(p_1).$$

Since there is exactly one negative coefficient, namely $\tau(0) - 1$, according to Descartes' rule, $p(x)$ has either two or no positive roots. Since $x = 1$ is a trivial solution, the former must be the case and so there is exactly one more positive root. Moreover at $x = 1$ the derivative of the polynomial equals

$$-1 + \tau(0) + 2\tau(1) + 3\tau(2) + \ldots + k\tau(k-1) > 0,$$

where the inequality follows by virtue of the biassing condition. Since therefore values slightly to the left of $x = 1$ are negative while the value at $x = 0$ is $\tau(-1)$ and thus positive again by the biassing condition, the only further positive root λ must be strictly in between and by the previous lemma this must equal p_1. □

We are now interested in the expected time

$$e_j := \mathbb{E}[\min\{i \in \mathbb{N}_0 : E_i = 0\} \mid \exists i \in \mathbb{N}_0 : E_i = 0\}]$$

which it takes when starting from j to get to state zero *under the condition* that zero is reached in finite time. We claim that this is bounded by a linear function in j.

Lemma A.11. *There exists a constant C' depending only on τ such that*

$$e_j \leq C' \cdot j$$

for all j.

Proof. The proof uses similar arguments as the one of Lemma A.9. Let $q_{j,N}$ denote the probability that when starting at state j and making N non-resting (increasing or decreasing) steps we arrive at $E_N = 0$. For this probability, we obtain via Chernoff a bound of the form

$$q_{j,N} \leq e^{-cN}, \tag{A.2}$$

because the decreasing steps which have probability smaller than one half among the non-resting steps must make for at least half of those steps. Hereby, c is a constant depending only on τ.

Let Q_j denote the number of non-resting steps it takes to reach zero starting from state j, but define Q_j to be zero whenever this number is infinite. This relates to e_j as

$$\mathbb{E}[Q_j] = e_j \cdot p_j \cdot (1 - \tau(0)).$$

Let $r \in \mathbb{N}$. We write $\mathbb{E}[Q_j]$ split into parts as

$$\mathbb{E}[Q_j] = \sum_{N=1}^{r} N \cdot \Pr\left[Q_j = N\right] + \sum_{N=r+1}^{\infty} N \cdot \Pr\left[Q_j = N\right].$$

Now we apply different bounds to the two parts. In the first sum we use $N \leq r$ and the fact that the probabilities can not sum up to more than p_j being the probability that Q_j takes any non-zero number. In the second sum, we use (A.2). This yields

$$\mathbb{E}[Q_j] \leq r \cdot p_j + \sum_{N=r+1}^{\infty} N e^{-cN}.$$

If we select r large enough that the second sum is smaller than p_j, then we obtain

$$\mathbb{E}[Q_j] \leq (r+1)p_j$$

and thus

$$e_j \leq \frac{r+1}{1-\tau(0)}.$$

Indeed, there is some constant R_1 depending only on c and thus on τ such that

$$\forall N > R_1 \ : \ N e^{-cN} < e^{-\frac{c}{2} \cdot N},$$

and some constant R_2 depending only on c and on λ and thus only on τ such that

$$\forall N > R_2 \ : \ e^{-\frac{c}{2} \cdot N} < \lambda \cdot (1 - e^{-\frac{c}{2}}),$$

whence it follows that if we select for example $r := \max\{R_1, R_2\} \cdot j$, we obtain

$$\sum_{N=r+1}^{\infty} N e^{-cN} < \left[\lambda(1 - e^{-\frac{c}{2}})\right]^j \cdot \sum_{N=0}^{\infty} e^{-\frac{c}{2}N} < \lambda^j = p_j.$$

Since r depends linearly on j and the rest are constants depending only on τ, the same holds for e_j. □

The main lemma follows like this: using Markov's inequality to obtain another constant C from C', we get a lower bound for the probability of hitting zero within the given number of steps which is a constant away from p_j. On the other hand, p_j is the probability that we hit zero within an arbitrary amount of steps, so we obtain equality. □

In some of our applications, the chain in question has a *reflection point*, which is a state beyond which the process cannot increase and then reflects. We also obtain an estimate on this type of process.

Lemma A.12. *Let $k \geq 2$ be some integer and*

$$\tau : \{-1, 0, 1, 2, \ldots, k-1\} \to [0, 1]$$

be a $(k+1)$-tuple of probabilities which sum up to one, where $\tau(-1) > 0$ and which have the bias

$$b_\tau := \frac{\tau(-1)}{1 - \tau(0)} < \frac{1}{2}.$$

Let furthermore s be a positive integer. Let $\{E_i \in \mathbb{Z}\}_{i \in \mathbb{N}_0}$ be a Markov Chain such that $E_0 = j$ and which then evolves as follows:

$$\forall i \geq 1 : \forall t \in \{-1, 0, 1, 2, \ldots, k-1\} : \; Pr\left[E'_i = E_{i-1} + t\right] \; = \; \tau(t)$$

and then $E_i := \min\{E'_i, s\}$. For such a reflected chain and any number of steps N,

$$Pr\left[\exists i \leq N : E_i = 0\right] \leq \lambda^j + N \cdot \lambda^s.$$

holds, where λ is the unique root in $(0, 1)$ of the polynomial

$$p(x) := -x + [\tau(-1) + \tau(0)x + \tau(1)x^2 + \ldots + \tau(k-1)x^k].$$

Proof. We use a coupling argument of the following type. Consider N Markov chain processes

$$\{M_i^{(t)} \in \mathbb{Z}\}_{i \in \mathbb{N}_0}, \;\; 0 \leq t \leq N,$$

where $M_0^{(0)} = j$ while $M_i^{(k)} = s$ for all $i \leq k$. And then for all $i > k$, we define

$$M_i^{(k)} = M_{i-1}^{(k)} + T_i,$$

where T_i is distributed according to τ and where we use the *same* T_i for all chains. To couple the process in the claim to these chains, we use

$$E_i' := E_{i-1} + T_i$$

as well.

Each chain separately behaves exactly like the type of chain discussed in Lemma A.7, where chain $M^{(0)}$ starts at state j while chain $M^{(k)}$ starts moving only at time $i = k$ from starting state s. Using the proof of Lemma A.7, for each chain separately, the probability that it *ever* reaches zero in an arbitrary amount of steps is λ^j in the case of $M^{(0)}$ and λ^s in the case of $M^{(k)}$ for $k > 0$. By a union bound, the probability that *any* of the chains *ever* hits zero is at most $\lambda^j + N\lambda^s$.

Now the claim follows from observing that clearly $E_i \geq \min_k M_i^{(k)}$ because each time a reflection occurs, we can continue tracing the corresponding of our N chains which starts moving at the reflection point at that very moment. □

A.6 Binomial Coefficients

The following estimate will come handy when working with binomial coefficients. It is a well-known estimate found in many textbooks.

Lemma A.13. *For any fixed constant $0 < \rho < 1$ and integers $n \to \infty$ we have*

$$\binom{n}{\rho n} = \left[\frac{1}{1-\rho}\left(\frac{1-\rho}{\rho}\right)^{\rho}\right]^{n+o(n)}.$$

Proof. The proof is straightforward since we are not interested in the exact constants. We can first derive a much simplified version of Stirling's formula by noting that

$$\int_0^n \log(x)dx \;<\; \log(n!) \;<\; \int_1^{n+1} \log(x)dx$$

from which

$$n\ln n - n < \log(n!) < (n+1)\ln(n+1) - n$$

and by exponentiation

$$n^n \cdot e^{-n} < n! < (n+1)^{n+1} \cdot e^{-n} = n^n \cdot e^{-n} \cdot (n+1) \cdot \left(1 + \frac{1}{n}\right)^n.$$

Since the last factor converges, we have

$$n! = \left(\frac{n}{e}\right)^{n+o(n)}.$$

We plug this equation into the definition of the binomial coefficient and obtain

$$\binom{n}{\rho n} = \frac{n!}{(\rho n)! \cdot ((1-\rho)n)!} = \frac{\left(\frac{n}{e}\right)^{n+o(n)}}{\left(\frac{\rho n}{e}\right)^{\rho n+o(n)} \cdot \left(\frac{(1-\rho)n}{e}\right)^{(1-\rho)n+o(n)}},$$

from which the claim readily follows. □

A.7 Proof of Theorem 2.3 (from Theorem 2.4)

Set $\mu(C) := e/d^k$ for all $C \in F$. Then Theorem 2.4 guarantees satisfiability as long as

$$\frac{1}{d^k} = w(C) \leq \mu_C \prod_{D \in \Gamma_F(C)} (1 - \mu_D) = \frac{e}{d^k}\left(1 - \frac{e}{d^k}\right)^{|\Gamma_F(C)|}.$$

With the hypothesis $|\Gamma_F(C)| \leq \frac{d^k}{e} - 1$ this is provided as long as

$$\frac{1}{d^k} \leq \frac{e}{d^k}\left(1 - \frac{e}{d^k}\right)^{\frac{d^k}{e}-1},$$

or, equivalently, $(1 - 1/x)^{x-1} \geq 1/e$ for $x := d^k/e > 1$, which is Lemma A.3. □

A.8 Proof of Theorem 2.5

A *star* is a graph G in which there exists a center vertex v to which all other vertices are connected, while all vertices other than v form an independent set. Call a ClSP F a *star formula* if its dependency graph is a star. Star formulas which are non-degenerate are satisfiable since one can pick any variable from the center clause and assign it in any way which satisfies the center clause and then the remaining star falls apart into an independent set of non-empty clauses each of which can be separately satisfied.

We claim that the following holds.

Lemma A.14. *Let F be a non-degenerate ClSP. If for all clauses $C \in F$, we have*

$$\sum_{D \in \Gamma_F(C)} w(D) \leq \frac{1}{4},$$

then F can be split as $F = F_0 \dot\cup F_1$ such that F_0 and F_1 are independent, F_0 is an independent union of star formulas and F_1 satisfies the local condition.

If so, then the theorem follows trivially as F_0 is satisfiable because it consists of star formulas while F_1 is satisfiable due to the Local Lemma.

Proof of Lemma A.14. Since the ClSP is non-degenerate, we know that we have $w(C) \leq \frac{1}{4}$ for all $C \in F$. Let F_0 consist of all connected components of F which contain a clause C_0 reaching this value, i.e. that $w(C_0) = \frac{1}{4}$.

Fix any $C_0 \in F_0$ with $w(C_0) = \frac{1}{4}$. Then consider all clauses adjacent to C_0 in the dependency graph: each of these clauses can *only* have C_0 in their neighborhoods since the weight of C_0 already saturates the bound in the hypothesis for each of these clauses. Therefore C_0 and its neighbors form a star independent of the remainder of the formula.

Let $F_1 = F \setminus F_0$ be the remainder. In F_1, all clauses have weight at most $\frac{1}{6}$. This is because this is the next possible value of the weight, arising from a 2-constraint where one variable has a domain of size two and the other one a domain of size three. Set $\mu(C) := ew(C)$ for all $C \in F_1$. Then Theorem 2.4 guarantees satisfiability as long as

$$w(C) \leq ew(C) \prod_{D \in \Gamma_F(C)} (1 - ew(D)).$$

Therefore, we have for all $C \in F_1$, using Lemma A.2 with $\alpha := 1.34$, since then $\beta(\alpha) > 0.46$ while $ew(D) \leq \frac{e}{6} < 0.46$,

$$ew(C) \prod_{D \in \Gamma_F(C)} (1 - ew(D)) \geq w(C) \cdot e \prod_{D \in \Gamma_F(C)} e^{-1.34 ew(D)} \geq$$

$$= w(C) \cdot e^{1 - 1.34e \sum_{D \in \Gamma_F(C)} w(D)} \geq 1.09 \cdot w(C),$$

where the last step uses the hypothesis. As this establishes what we need for Theorem 2.4, the claim readily follows. □

A.9 Proof of Lemma 2.36

This is not difficult to compute because all the branches of the tree develop totally independently. For every vertex $v \in V(T)$, the probability that this vertex is created and labelled $[v]$ at the (unique) moment when there was a chance to do that is just $\mu([v])$. On the other hand, for any vertex $v \in V(T)$ and any $C \in \Gamma_F^+([v])$ such that v does *not* have a child labelled C, the probability that the process really does *not* create that child at the (unique) moment when there was a chance to

create it is $1 - \mu(C)$. So, to get the total probability of outputting exactly the tree T and nothing larger or smaller, we have to multiply for all nodes $v \in V(T)$ the probability $\mu([v])$ and for all children that *could have been created* with a label C but have not the probability $1 - \mu(C)$. Letting

$$W(v) := \{C \in \Gamma_F^+([v]) \mid \text{no child of } v \text{ is labelled } C\}$$

for all $v \in V(T)$, we get that the probability of obtaining exactly T from the branching process equals

$$\underbrace{\prod_{v \in V(T) \setminus \{\text{root}\}} \mu([v])}_{\substack{\text{probability that the process does create all nodes that are there in } T}} \cdot \underbrace{\prod_{v \in V(T)} \prod_{C \in W(v)} (1 - \mu(C))}_{\substack{\text{probability that the process does } \textit{not} \text{ create any node that is } \textit{not} \text{ there in } T}}.$$

In order to combine the two products, let us multiply by μ_A for the root and divide by the same term in front of the whole expression, e.g.

$$\frac{1}{\mu(A)} \prod_{v \in V(T)} \left(\mu([v]) \cdot \prod_{C \in W(v)} (1 - \mu(C)) \right).$$

Instead of multiplying over $C \in W(v)$ in the second inner product, we can multiply over all $C \in \Gamma_F^+([v])$ and then divide again by $1 - \mu(C)$ for all $C \notin W(v)$. Note that all cases with $C \notin W(v)$ are exactly the labels that *do* appear somewhere in T, except for the root for which we have divided once too much, which we compensate for by adding the term $1 - \mu_A$ in front, so we get that the probability equals

$$\frac{1 - \mu(A)}{\mu(A)} \prod_{v \in V(T)} \left(\frac{\mu([v])}{1 - \mu([v])} \prod_{C \in \Gamma_F^+([v])} (1 - \mu(C)) \right).$$

Replacing Γ_F^+ by Γ_F, we will lose one term $1 - \mu([v])$ in the product, which nicely cancels out with the denominator of the fraction, yielding

$$\frac{1 - \mu(A)}{\mu(A)} \prod_{v \in V(T)} \left(\mu([v]) \prod_{C \in \Gamma_F([v])} (1 - \mu(C)) \right),$$

as claimed. □

A.10 Proof of Lemma 2.38

To establish the statement, we have to be more precise with the estimate used in the proof of Lemma 2.3 (see Appendix A.7). We again set

$$\mu(C) := \frac{e}{d^k}$$

for all $C \in F$. Now the formula satisfies the local condition with slack ϵ if

$$\frac{1}{d^k} \leq (1-\epsilon) \cdot \mu(C) \cdot \prod_{D \in \Gamma_F(C)} (1-\mu(D)).$$

The right hand side equals

$$\frac{e}{d^k}(1-\epsilon)\left(1-\frac{e}{d^k}\right)^{|\Gamma_F(C)|}.$$

From the hypothesis $|\Gamma_F(C)| \leq (1-\delta)\left(\frac{d^k}{e}-1\right)$, thus it suffices to prove

$$\frac{1}{e} \leq (1-\epsilon)\left(1-\frac{e}{d^k}\right)^{(1-\delta)\left(\frac{d^k}{e}-1\right)}.$$

Lemma A.3 yields that

$$\left(1-\frac{e}{d^k}\right)^{\left(\frac{d^k}{e}-1\right)} \geq \frac{1}{e} + \frac{1}{2e}\cdot\frac{e}{d^k} = \frac{1}{e}\left(1+\frac{e}{2d^k}\right)$$

and therefore

$$(1-\epsilon)\left(1-\frac{e}{d^k}\right)^{(1-\delta)\left(\frac{d^k}{e}-1\right)} \geq (1-\epsilon)\left(1+\frac{e}{2d^k}\right)^{1-\delta}\cdot\frac{1}{e^{1-\delta}} =$$

$$= \frac{1}{e}\cdot\left[(1-\epsilon)\left(1+\frac{e}{2d^k}\right)\underbrace{\left(\frac{e}{1+\frac{e}{8}}\right)^{\delta}}_{\geq 2^{\delta}}\right]\cdot\underbrace{\left(\frac{1+\frac{e}{8}}{1+\frac{e}{2d^k}}\right)^{\delta}}_{\geq 1}.$$

It is left to show that the term in square brackets is at least one. We apply the simple first inequality of Lemma A.4 to see that this part is at least

$$(1-\epsilon)\left(1+\frac{e}{d^k}+(2^\delta-1)\right)$$

and so, again according to Lemma A.4 (this time using the last inequality), if we set

$$\epsilon := \frac{1}{2}\cdot\left(2^\delta-1+\frac{e}{d^k}\right),$$

we obtain that the local condition is satisfied with slack ϵ. □

A.11 Proof of Lemma 2.39

We first follow the proof of Theorem 2.5 in order to split F into a part F_0 consisting of a disjoint union of stars (which are matched) and an independent part F_1 having $w(C) \leq \frac{1}{6}$ for all $C \in F_1$. We recall from the proof of Theorem 2.5 that when assigning weights $\mu := ew(C)$, we obtain $w_\mu(C) \geq 1.09 \cdot w(C)$ for all $C \in F_1$. Therefore if we let $\epsilon < 0.08$, then we have $(1-\epsilon)w_\mu(C) \geq w(C)$ as required. □

A.12 Proof of Lemma 2.40

We use Lemma A.5 to produce μ'. Each μ-weight have either decreased or changed to $\frac{1}{2}$. The first case is trivial. In the second case, we note that the right-hand side of the slacked hypothesis is $(1-\epsilon/2)\cdot\frac{1}{2}$ and thus, since $\epsilon \leq 1$, well-above the largest possible weight in a non-degenerate ClSP of $\frac{1}{4}$. □

A.13 Proof of Lemma 2.41

First we reshape the hypothesis somewhat. If $\epsilon \leq \delta/2$, then

$$1 - \delta \leq 1 - 2\epsilon \leq 1 - 2\epsilon + \frac{3\epsilon^2 + 2\epsilon^3}{1 + 2\epsilon + \epsilon^2} = \left(\frac{1}{1+\epsilon}\right)^2. \tag{A.3}$$

Furthermore, note that for any $y > 1, z \in (0,1)$, we have $y^z - 1 \leq (y-1)^z$. Combining this with (A.3), we obtain from the hypothesis that

$$|\Gamma_F(C)| \leq \left(\frac{e}{d^k}\right)^{\left(\frac{1}{1+\epsilon}\right)^2} - 1 \leq \left(\left(\frac{e}{d^k}\right)^{\frac{1}{1+\epsilon}} - 1\right)^{\frac{1}{1+\epsilon}}. \tag{A.4}$$

In this form, the hypothesis is suited to the following calculation. We now proceed analogously to the proof of Lemma 2.38, but this time we set

$$\mu(C) := \left(e \cdot d^{-k}\right)^{\frac{1}{1+\epsilon}}$$

for all $C \in F$. Now the formula satisfies the local condition with exponential slack ϵ if

$$\frac{1}{d^k} \leq \left(\mu(C) \prod_{D \in \Gamma_F(C)} (1 - \mu(D))\right)^{1+\epsilon}.$$

The right hand side is at least as large as

$$\frac{e}{d^k} \cdot \left(1 - \left(\frac{e}{d^k}\right)^{\frac{1}{1+\epsilon}}\right)^{(1+\epsilon)|\Gamma_F(C)|}.$$

Plugging in (A.4), we are done if ϵ satisfies

$$\left[\left(1 - \left(\frac{e}{d^k}\right)^{\frac{1}{1+\epsilon}}\right)^{\left(\left(\frac{e}{d^k}\right)^{\frac{1}{1+\epsilon}} - 1\right)^{\frac{1}{1+\epsilon}}}\right]^{1+\epsilon} \geq \frac{1}{e}.$$

After cancelling the exponents, this is Lemma A.3. □

A.14 Proof of Lemma 2.42

Basically, we follow the proof of Theorem 2.5, although the hypothesis is now sufficiently strong such that no additional splitting is necessary. Note that all $C \in F$ with $w(C) \geq 1/6$ must be isolated vertices as otherwise the condition would be violated at all neighbors. By setting $\mu(C)$ arbitrarily close to one for all such C, we can satisfy the slacked condition easily. From now on, we can assume that we are working on $F' \subseteq F$ with $w(C) < 1/6$ for all $C \in F'$ exclusively.

We let $\epsilon := \delta$. For $C \in F'$, we now set

$$\mu(C) := [ew(C)]^{\frac{1}{1+\epsilon}}.$$

Note that from $w(C) < 1/6$ and $\epsilon < 1$ follows that

$$\mu(C) < 0.68 < \beta(1.7), \tag{A.5}$$

where β is defined as in Lemma A.2. The slacked hypothesis is satisfied if

$$\left[\prod_{D \in \Gamma_F(C)} (1 - \mu(C))\right]^{1+\epsilon} \geq \frac{1}{e}$$

and via Lemma A.2 and (A.5), it is sufficient to prove that

$$\sum_{D \in \Gamma_F(C)} [w(C)]^{\frac{1}{1+\epsilon}} \leq \frac{1}{(1+\epsilon) \cdot 1.7 \cdot e^{\frac{1}{1+\epsilon}}}.$$

Differentiating the right hand side, we find that for all $\epsilon \in [0, 1)$, it is at least $1/6$. Since

$$1 - \delta = \frac{1}{1+\epsilon} - \frac{\epsilon^2}{1+\epsilon} < \frac{1}{1+\epsilon},$$

the condition in the hypothesis is stronger than this. □

A.15 Proof of Lemma 2.43

We use Lemma A.5 to produce μ'. Each μ-weight has either decreased or changed to $\frac{1}{2}$. The first case is trivial. In the second case, we note that the right-hand side of the slacked hypothesis is $2^{-(1+\epsilon)} \geq \frac{1}{4}$ and thus, since $\epsilon \leq 1$ by hypothesis, well-above the largest possible weight in a non-degenerate ClSP of $\frac{1}{4}$. □

A.16 Proof of Lemma 2.48

We prove that Algorithm 4 needs an expected logarithmic number of phases until F is satisfied and the claim then follows.

Let us focus on a single star component S first. Note that until S is satisfied, every phase of the algorithm is either a satellite phase for S, i.e. a phase where all violated satellite clauses are picked for resampling, or a center phase where the center clause is selected (there are no other options for maximal independent sets of violated clauses). Each satellite clause has a variable which occurs nowhere else. Therefore, each time a satellite clause is picked for resampling, there is a probability of at least $1/2$ that this variable changes its value whereupon the clause will be satisifed and never turn violated again. Therefore the probability that a fixed satellite clause partakes in more than t phases is at most 2^{-t}. By a union bound, the probability that *any* satellite clause in S partakes in more than t phases (i.e. the probability that there *are* more than t satellite phases for S) is thus at most $|S| \cdot 2^{-t}$.

Now use a union bound over the number s of stars in F to see that the probability that there is *any* star in F for which more than t satellite phases are being conducted is at most $s \cdot |F| \cdot 2^{-t}$. Let Y be the maximum number of satellite phases conducted over all stars in the formula. We have established that $Y \geq t$ with probability at most

$|F|^2 \cdot 2^{-t}$. Via Lemma A.6, we obtain that Y has an expectation which is logarithmic in $|F|$.

Between any two satellite phases, there is a constant expected number of center phases because each such phase satisfies the center clause with constant probability. Thus the expected maximum number of total phases in any star is at most a constant multiple of Y which concludes the proof.

A.17 Proof of Lemma 2.56[2]

We start from Lemma 2.36. Recall that if $\mathcal{T}_A$ is the set of all (regular, full, proper) witness trees of which the root is labelled by A, then Lemma 2.36 yields that

$$\sum_{T \in \mathcal{T}_A} \prod_{v \in V(T)} w_\mu([v]) \;\leq\; \frac{\mu(A)}{1-\mu(A)}. \tag{A.6}$$

So far, we have summed over all full witness trees. What we are really interested in is the set $\mathcal{T}'_A$ of all partial witness trees where the root is labelled with some clause from the splitting hull of A and having a μ-weight in the range $[\gamma^2, \gamma]$. Consider any mapping $\zeta : \mathcal{T}'_A \to T_A$ that associates a full witness tree $T \in \mathcal{T}_A$ with every partial witness tree $T \in T'_A$ by relabelling the root by A. Note that every tree $T \in \mathcal{T}_A$ has a preimage under ζ of size at most $3|A|$ since this was the bound we derived on the size of the splitting hull of A. Moreover, as the definition of μ-weights of partial trees in contrast to that for full trees excludes the root vertex, we get an additional $w_\mu(A)$ correction factor and end up with

$$\sum_{T \in \mathcal{T}'_A} w_\mu(T) \;\leq\; 3|A| \cdot \frac{\mu(A)}{1-\mu(A)} \cdot \frac{1}{w_\mu(A)}.$$

[2] from [CGH09]

Summing over all $A \in F$, this already yields the second claim.

For the first claim, we continue and note that by definition of T'_A, $w_\mu(T) \leq \gamma$, therefore

$$\sum_{T \in \mathcal{T}'_A} \left[w_\mu(T)\right]^{1+\epsilon} \;\leq\; 3|A| \cdot \frac{\mu(A)}{1-\mu(A)} \cdot \frac{\gamma^\epsilon}{w_\mu(A)}.$$

We apply the slacked hypothesis and the definition of γ and the fact $|A| \leq |V|$ to obtain

$$\sum_{T \in \mathcal{T}'_A} w(T) \;\leq\; 3|V| \cdot \frac{\mu(A)}{1-\mu(A)} \cdot \frac{1}{M \cdot w_\mu(A)}.$$

Plugging in the definition of M, this yields

$$\sum_{T \in \mathcal{T}'_A} w(T) \leq \frac{1}{2 \cdot |F|}.$$

Summing over all $A \in F$ yields the claim. □

A.18 Proof of Theorem 3.8[3]

Let us first mention the issue of rounding. In the theorem, there are many rounding brackets. This is necessary if one wants the statement to hold. After all, n might not be divisible by $f(n)$ and $f(n)$ might not be divisible by k, and so forth. Moreover, for the event we define to have any probability other than zero, we require that $l + r = t$ *exactly*, et cetera. While we did this with care in formulating the theorem, we will forget about rounding in the proof we give here. This is not a real issue. After all, our proof goal is that the defined event be *typical*

[3]Preparatory for [MS11] which is joint work with Dominik Scheder.

which means that it needs to have probability

$$\left(\frac{1}{2}\cdot\frac{k}{k-1}\right)^{n+o(n)},$$

which is the success probability we have established for Schöning's algorithm *up to lower order terms* in the exponent. When reading the following proof, we keep the matter of rounding in mind and convince ourselves that any effect it might have can easily be subsumed in the lower order error term we allow. We read the argument as if all quantities involved were simply integers.

For estimating the probability of the given intersection of events, we estimate separate independent parts of which we can finally multiply the probabilities. First, we estimate the probability that initially $E_0 = n/k$. Next, we estimate the probability that in any given window of t evolution steps, exactly r, i.e. a $\frac{1}{k}$-fraction are increasing and the remainder decreasing. By the nature of the E-process which selects a starting state at random, then does evolution steps in which each step is independent of the history, the final probability will arise from multiplying all these estimates.

Let us calculate the probability that $E_0 = n/k$. By the binomial distribution of E_0 and then by using Lemma A.13, we obtain

$$\Pr\left[E_0 = \frac{n}{k}\right] = \left(\frac{1}{2}\right)^n \binom{n}{n/k} = \left[\frac{1}{2}\cdot\frac{1}{1-\frac{1}{k}}\left(\frac{1-\frac{1}{k}}{\frac{1}{k}}\right)^{\frac{1}{k}}\right]^{n+o(n)} =$$

$$= \left[\frac{1}{2}\cdot\frac{k}{k-1}(k-1)^{\frac{1}{k}}\right]^{n+o(n)}.$$

Next, we estimate the probability that within any given window of $f(n)/(k-2)$ evolution steps, exactly $1/k$-fraction are increasing. For this probability we obtain using again Lemma A.13,

$$\Pr\left[\left|\left\{1 \le i \le \frac{f(n)}{k-2} \mid E_i = E_{i-1} - 1\right\}\right| = l\right] =$$

$$= \begin{pmatrix} \frac{f(n)}{k-2} \\ \frac{f(n)}{k(k-2)} \end{pmatrix} \cdot \left(\frac{1}{k}\right)^{\frac{k-1}{k}\cdot\frac{f(n)}{k-2}} \cdot \left(\frac{k-1}{k}\right)^{\frac{1}{k}\cdot\frac{f(n)}{k-2}} =$$

$$= \left[\frac{k}{k-1}\cdot(k-1)^{\frac{1}{k}}\cdot\left(\frac{1}{k}\right)^{\frac{k-1}{k}}\cdot\left(\frac{k-1}{k}\right)^{\frac{1}{k}}\right]^{\frac{f(n)}{k-2}+o(f(n))} =$$

$$= \left((k-1)^{\frac{2-k}{k}}\right)^{\frac{f(n)}{k-2}+o(f(n))} = \left((k-1)^{-\frac{1}{k}}\right)^{f(n)+o(f(n))}.$$

To obtain the total probability, we multiply the first factor and $n/f(n)$ of the second factors and obtain

$$\left[\frac{1}{2}\cdot\frac{k}{k-1}(k-1)^{\frac{1}{k}}\right]^{n+o(n)} \cdot \left((k-1)^{-\frac{1}{k}}\right)^{\frac{n}{f(n)}\cdot f(n)+\frac{n}{f(n)}\cdot o(f(n))},$$

readily yielding the desired quantity. □

Bibliography

[ACO08] Dimitris Achlioptas and Amin Coja-Oghlan, *Algorithmic Barriers from Phase Transitions*, Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008), 2008, pp. 793–802.

[AKS10] Andris Ambainis, Julia Kempe, and Or Sattath, *A Quantum Lovász Local Lemma*, Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010), 2010, pp. 151–160.

[Alo91] Noga Alon, *A Parallel Algorithmic Version of the Local Lemma*, Random Structures and Algorithms **2** (1991), no. 4, 367–378.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan, *A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas*, Information Processing Letters **8** (1979), no. 3, 121–123.

[AS00] Noga Alon and Joel H. Spencer, *The Probabilistic Method*, second ed., Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience [John Wiley & Sons], New York, 2000, With an appendix on the life and work of Paul Erdős.

[Bec91] József Beck, *An Algorithmic Approach to the Lovász Local Lemma. I*, Random Structures and Algorithms **2** (1991), no. 4, 343–365.

[BFPS11] Rodrigo Bissacot, Roberto Fernández, Aldo Procacci, and Benedetto Scoppola, *An Improvement of the Lovász Local Lemma via Cluster Expansion*, Combinatorics, Probability and Computing **20** (2011), no. 5, 709–719.

[BK04] Tobias Brueggemann and Walter Kern, *An Improved Deterministic Local Search Algorithm for 3-SAT*, Theoretical Computer Science **329** (2004), no. 1-3, 303–313.

[BKS03] Piotr Berman, Marek Karpinski, and Alex D. Scott, *Approximation Hardness and Satisfiability of Bounded Occurrence Instances of SAT*, Electronic Colloquium on Computational Complexity (ECCC) **10** (2003), no. 022.

[CGH+96] Robert M. Corless, Gaston H. Gonnet, Dave E.G. Hare, David J. Jeffrey, and Donald E. Knuth, *On the LambertW Function*, Advances in Computational Mathematics **5** (1996), 329–359.

[CGH09] Karthekeyan Chandrasekaran, Navin Goyal, and Bernhard Haeupler, *Deterministic Algorithms for the Lovász Local Lemma*, CoRR **abs/0908.0375** (2009).

[CO10] Amin Coja-Oghlan, *A Better Algorithm for Random k-SAT*, SIAM Journal on Computing (SICOMP) **39** (2010), no. 7, 2823–2864.

[Coo71] Stephen A. Cook, *The Complexity of Theorem-Proving Procedures*, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971), 1971, pp. 151–158.

[CS11] Toby S. Cubitt and Martin Schwarz, *A Constructive Commutative Quantum Lovász Local Lemma and Beyond*, CoRR **abs/1112.1413** (2011).

[DG84] William F. Dowling and Jean H. Gallier, *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae*, Journal of Logic Programming **1** (1984), no. 3, 267–284.

[DGH+02] Evgeny Dantsin, Andreas Goerdt, Ewald A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning, *A Deterministic $(2 - 2/(k+1))^n$ Algorithm for k-SAT Based on Local Search*, Theoretical Computer Science **289** (2002), no. 1, 69–83.

[Dub90] Olivier Dubois, *On the r, s-SAT Satisfiability Problem and a Conjecture of Tovey*, Discrete Applied Mathematics **26** (1990), no. 1, 51–60.

[EL75] Paul Erdős and László Lovász, *Problems and Results on 3-Chromatic Hypergraphs and Some Related Questions*, Infinite and Finite Sets (to Paul Erdős on his 60th birthday), Vol. II (A. Hajnal, R. Rado, and Vera T. Sós, eds.), North-Holland, 1975, pp. 609–627.

[ES73] Paul Erdös and J. L. Selfridge, *On a Combinatorial Game*, Journal of Combinatorial Theory, Series A **14** (1973), no. 3, 298–301.

[ES91] Paul Erdős and Joel Spencer, *Lopsided Lovász Local Lemma and Latin Transversals*, Discrete Applied Mathematics **30** (1991), no. 2-3, 151–154, ARIDAM III (New Brunswick, NJ, 1988).

[For06] Otto Forster, *Analysis 1. Differential- und Integralrechnung einer Veränderlichen (Grundkurs Mathematik)*, Vieweg, January 2006.

[FP07] Roberto Fernández and Aldo Procacci, *Cluster Expansion for Abstract Polymer Models. New Bounds from an old Approach*, Communications in Mathematical Physics **274** (2007), no. 1, 123–140.

[FV98] Tomás Feder and Moshe Y. Vardi, *The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory*, SIAM Journal on Computing **28** (1998), no. 1, 57–104.

[Geb09] Heidi Gebauer, *Disproof of the Neighborhood Conjecture with Implications to SAT*, Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009), 2009, pp. 764–775.

[GH11] William I. Gasarch and Bernhard Haeupler, *Lower Bounds on van der Waerden Numbers: Randomized- and Deterministic-Constructive*, Electronic Journal of Combinatorics **18** (2011), no. 1.

[Giu09] Andrei Giurgiu, *Random Walk Algorithms for SAT*, Master's thesis, ETH Zürich, 2009.

[GST11] Heidi Gebauer, Tibor Szabó, and Gábor Tardos, *The Local Lemma is Tight for SAT*, Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011), 2011, pp. 664–674.

[Hal35] Philip Hall, *On Representatives of Subsets*, Journal of the London Mathematical Society **s1-10** (1935), no. 1, 26–30.

[Her11] Timon Hertli, *3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General*, Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011), 2011, pp. 277–284.

[HMS11] Timon Hertli, Robin A. Moser, and Dominik Scheder, *Improving PPSZ for 3-SAT using Critical Variables*, Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011), 2011, pp. 237–248.

[HS05] Shlomo Hoory and Stefan Szeider, *Computing Unsatisfiable k-SAT Instances with Few Occurrences per Variable*, Theoretical Computer Science **337** (2005), no. 1-3, 347–359.

[HS06] Shlomo Hoory and Stefan Szeider, *A Note on Unsatisfiable k-CNF Formulas with Few Occurrences per Variable*, SIAM Journal on Discrete Mathematics **20** (2006), no. 2, 523–528.

[HSSW02] Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe, *A Probabilistic 3-SAT Algorithm Further Improved*, Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002), 2002, pp. 192–202.

[IT04] Kazuo Iwama and Suguru Tamaki, *Improved Upper Bounds for 3-SAT*, Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), 2004, p. 328.

[Knu73] Donald E. Knuth, *The Art of Computer Programming, Volume i: Fundamental Algorithms, 2nd Edition*, p. 396 (Exercise 11), Addison-Wesley, 1973.

[KR90] Richard M. Karp and Vijaya Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A), Elsevier and MIT Press, 1990, pp. 869–942.

[Kra49] Leon G. Kraft, *A Device for Quantizing, Grouping, and Coding Amplitude-Modulated Pulses*, Master's thesis, Massachusetts Institute of Technology, 1949.

[KS10] Konstantin Kutzkov and Dominik Scheder, *Using CSP to Improve Deterministic 3-SAT*, CoRR **abs/1007.1166** (2010).

[KS11] Kashyap Babu Rao Kolipaka and Mario Szegedy, *Moser and Tardos meet Lovász*, Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011), 2011, pp. 235–244.

[KST93] Jan Kratochvíl, Petr Savický, and Zsolt Tuza, *One more Occurrence of Variables makes Satisfiability jump from Trivial to NP-complete*, SIAM Journal of Computing **22** (1993), no. 1, 203–210.

[Kul99] Oliver Kullmann, *New Methods for 3-SAT Decision and Worst-case Analysis*, Theoretical Computer Science **223** (1999), no. 1-2, 1–72.

[Lev73] Leonid A. Levin, *Universal Sequential Search Problems*, Problems of Information Transmission **9** (1973), no. 3.

[Lub86] Michael Luby, *A Simple Parallel Algorithm for the Maximal Independent Set Problem*, SIAM Journal on Computing (SICOMP) **15** (1986), no. 4, 1036–1053.

[Mil12] Sebastian J. Millius, *Towards a Generalization of the PPSZ Algorithm for Large Domains and Multiple Solutions*, Master's thesis, ETH Zürich, 2012.

[Mos06] Robin A. Moser, *On the Search for Solutions to Bounded Occurrence Instances of SAT*, Semester Thesis, ETH Zürich, 2006.

[Mos08] ———, *Derandomizing the Lovász Local Lemma more Effectively*, CoRR **abs/0807.2120** (2008).

[Mos09] ———, *A Constructive Proof of the Lovász Local Lemma*, Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009), 2009, pp. 343–350.

[MR98] Michael Molloy and Bruce A. Reed, *Further Algorithmic Aspects of the Local Lemma*, Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC 1998), 1998, pp. 524–529.

[MS77] Florence J. MacWilliams and Neil J. A. Sloane, *The Theory of Error-Correcting Codes. II*, North-Holland Publishing Co., Amsterdam, 1977, North-Holland Mathematical Library, Vol. 16.

[MS85] Burkhard Monien and Ewald Speckenmeyer, *Solving Satisfiability in less than* 2^n *Steps*, Discrete Applied Mathematics **10** (1985), 287–295.

[MS11] Robin A. Moser and Dominik Scheder, *A Full Derandomization of Schöning's k-SAT Algorithm*, Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011), 2011, pp. 245–252.

[MT10] Robin A. Moser and Gábor Tardos, *A Constructive Proof of the General Lovász Local Lemma*, Journal of the ACM **57** (2010), no. 2.

[MT11] Jochen Messner and Thomas Thierauf, *A Kolmogorov Complexity Proof of the Lovász Local Lemma for Satisfiability*, Computing and Combinatorics (Bin Fu and Ding-Zhu Du, eds.), Lecture Notes in Computer Science, vol. 6842, Springer Berlin Heidelberg, 2011, pp. 168–179.

[MTY11] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto, *Derandomizing HSSW Algorithm for 3-SAT*, COCOON, 2011, pp. 1–12.

[MW11] Robin A. Moser and Emo Welzl, *Chapter 5: Lovász Local Lemma*, Algorithms, Probability and Computing, Lecture Notes (Thomas Holenstein, Ueli Maurer, Angelika Steger, Emo Welzl, and Peter Widmayer, eds.), ETH Zürich, 2011.

[MW12a] ———, *Chapter 2*: The Algorithmic Lovász Local Lemma*, Boolean Satisfiability – Combinatorics and Algorithms, Lecture Notes, Emo Welzl, 2012.

[MW12b] ———, *Chapter 7*: Derandomizing Schöning's Algorithm*, Boolean Satisfiability – Combinatorics and Algorithms, Lecture Notes, Emo Welzl, 2012.

[Pap91] Christos H. Papadimitriou, *On Selecting a Satisfying Truth Assignment (Extended Abstract)*, Proceedings of the 32nd Annual Symposium of Foundations of Computer Science (FOCS 1991), 1991, pp. 163–169.

[Peg11] Wesley Pegden, *An Improvement of the Moser-Tardos Algorithmic Local Lemma*, CoRR **abs/1102.2853** (2011).

[PPSZ05] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane, *An Improved Exponential-Time Algorithm for k-SAT*, Journal of the ACM **52** (2005), no. 3, 337–364.

[PPZ99] Ramamohan Paturi, Pavel Pudlák, and Francis Zane, *Satisfiability Coding Lemma*, Chicago Journal of Theoretical Computer Science (1999), Article 11, 19 pp. (electronic).

[Rod96] Robert Rodošek, *A New Approach on Solving 3-Satisfiability*, AISMC, 1996, pp. 197–212.

[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) (2003), no. 054.

[Sch99] Uwe Schöning, *A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems*, Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999), 1999, p. 410.

[Sch08] Dominik Scheder, *Guided Search and a Faster Deterministic Algorithm for 3-SAT*, Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN 2008), 2008, pp. 60–71.

[Sch09] Pascal Schweitzer, *Using the Incompressibility Method to Obtain Local Lemma Results for Ramsey-Type Problems*, Information Processing Letters **109** (2009), no. 4, 229–232.

[Sch10] Stefan Schneider, *Random Walk Algorithms for SAT and Constraint Satisfaction Problems*, Master's thesis, ETH Zürich, 2010.

[She85] James B. Shearer, *On a Problem of Spencer*, Combinatorica **5** (1985), no. 3, 241–245.

[Spe77] Joel Spencer, *Asymptotic Lower Bounds for Ramsey Functions*, Discrete Mathematics **20** (1977), no. 0, 69 – 76.

[Sri08] Aravind Srinivasan, *Improved Algorithmic Versions of the Lovász Local Lemma*, Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008) (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2008, pp. 611–620.

[SS05] Alexander Scott and Alan Sokal, *The Repulsive Lattice Gas, the Independent-Set Polynomial, and the Lovász Local Lemma*, Journal of Statistical Physics (2005).

[SS06] Alexander D. Scott and Alan D. Sokal, *On Dependency Graphs and the Lattice Gas*, Combinatorics, Probability and Computing **15** (2006), no. 1-2, 253–279.

[Sze04] Stefan Szeider, *Minimal Unsatisfiable Formulas with Bounded Clause-Variable Difference are Fixed-Parameter Tractable*, Journal of Computer and System Sciences **69** (2004), no. 4, 656–674.

[Sze11] May Szedlák, *The PPSZ Algorithm for Large Domains*, Bachelor's thesis, ETH Zürich, 2011.

[Tov84] Craig A. Tovey, *A Simplified NP-Complete Satisfiability Problem*, Discrete Applied Mathematics **8** (1984), no. 1, 85–89.

[Wel12] Emo Welzl, *Boolean Satisfiability – Combinatorics and Algorithms*, Lecture Notes, ETH Zürich, 2012.

[Yin11] Mingsheng Ying, *Another Quantum Lovász Local Lemma*, CoRR **abs/1010.5577v3** (2011).