Improved Exponential Algorithms for SAT and ClSP

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

Timon Hertli

MSc ETH CS
born on 28 October 1985
citizen of Kleinandelfingen ZH, Switzerland

accepted on the recommendation of

Prof. Dr. Emo Welzl, examiner
Prof. Dr. Kazuo Iwama, co-examiner
Prof. Dr. Ramamohan Paturi, co-examiner

2015
Contents

Contents iv
Abstract v
Zusammenfassung vii
Acknowledgements ix
1 Introduction 1
2 Notation 5
3 The PPSZ Algorithm for Unique $k$-SAT 9
  3.1 General Properties of PPSZ 10
  3.2 Bounding the Guessed Probability of Unique $k$-SAT 15
    3.2.1 Motivation 15
    3.2.2 Building Critical Clause Trees 16
    3.2.3 Connection to Forced Variables 17
  3.3 Random Deletion in $k-1$-ary Trees 19
    3.3.1 Of Infinite and Finite Trees 20
    3.3.2 Of Independent and Dependent Labels 23
    3.3.3 Integrating over the Rank of the Root 25
4 PPSZ for Multiple Satisfying Assignments 29
  4.1 Problem Assessment 30
  4.2 Definition of a Cost Function 33
  4.3 Basic Facts 35
  4.4 Bounding Expectations of Products 39
  4.5 Trading Survival Probability for Cost Savings 41
Abstract

Satisfiability of Boolean formulas (SAT) is one of the most prominent NP-complete problems. We consider $k$-SAT, the decision problem that asks whether formulas in conjunctive normal form (CNF) with clauses of size at most $k$ are satisfiable, and the more general problem called $(d,k)$-ClSP (clause satisfaction problem) where the variables are $d$-valued instead of Boolean.

For $k$-SAT and $(d,k)$-ClSP many algorithms have been presented whose running time is “moderately exponential” in the number of variables of the input formula. One of the fastest randomized algorithm for $k$-SAT is the PPSZ algorithm by Paturi, Pudlák, Saks, and Zane (FOCS 1998). We re-analyze the PPSZ algorithm and show that the bounds shown in the case where the input formula has at most one satisfying assignment (Unique $k$-SAT) hold in general, which was previously only known for $k \geq 5$.

We also show how to generalize PPSZ to $(d,k)$-ClSP, improving on the previous algorithms for most considered values of $(d,k)$.

Furthermore, we present a new algorithm based on PPSZ with exponentially better bounds for 3-SAT. For general $k$ we show that in order to improve on PPSZ for $k$-SAT, it is enough to improve on PPSZ for Unique $k$-SAT.
Zusammenfassung

Das Erfüllbarkeitsproblem boolescher Formeln (SAT) ist eines der wichtigsten NP-vollständigen Probleme. Wir betrachten \( k \)-SAT, das Entscheidungsproblem, das fragt ob Formeln in konjunktiver Normalform (CNF) mit maximaler Klauselgröße von \( k \) erfüllbar sind, und das allgemeinere Problem namens \((d,k)\)-ClSP (Klauselerfüllbarkeitsproblem), bei dem die Variablen \( d \)-wertig anstatt boolesch sind.

Für \( k \)-SAT und \((d,k)\)-ClSP wurden viele Algorithmen präsentiert, deren Laufzeit "moderat exponentiell" von der Zahl der Variablen der Eingabeformel abhängt. Einer der schnellsten randomisierten Algorithmen für \( k \)-SAT ist der PPSZ-Algorithmus von Paturi, Pudlák, Saks, and Zane (FOCS 1998). Wir analysieren den PPSZ-Algorithmus erneut und zeigen, dass die Schranken, welche im Falle von Eingabeformeln mit maximal einer erfüllender Belegung (Unique \( k \)-SAT) gezeigt wurden, auch im Allgemeinen gelten. Dies war zuvor nur für \( k \geq 5 \) bekannt.

Wir zeigen auch, wie man den PPSZ-Algorithmus auf \((d,k)\)-ClSP verallgemeinern kann, und verbessern damit die vorherigen Algorithmen für die meisten betrachteten Werte von \((d,k)\).

Zudem präsentieren wir einen neuen Algorithmus basierend auf dem PPSZ-Algorithmus mit exponentiell besseren Schranken für 3-SAT. Für allgemeines \( k \) zeigen wir, dass es, um den PPSZ-Algorithmus für \( k \)-SAT zu verbessern, genügt, den PPSZ-Algorithmus für Unique \( k \)-SAT zu verbessern.
Acknowledgements

I would like to thank my advisor Emo Welzl for accepting me into his research group, for his support in realizing this thesis, for his guidance, and for his great satisfiability lecture that draw my interest to the topic in the first place. Many thanks also to Robin Moser and Dominik Scheder, advisors of my master thesis, co-authors, and colleagues, guiding my first steps into research.

Thanks to my co-examiners Kazuo Iwama and Mohan Paturi, for inviting me to their universities, for their many suggestions to my research, for spending the time to understand my thesis, and for coming to Zurich just for my examination. Thanks to Stefan Schneider for his hospitality in San Diego.

Thanks to Sebastian Stich, office mate for almost all my time here. Thanks to the other members of my research group during my time here: Yves Brise, Tobias Christ, Andra Francke, Komei Fukuda, Bernd Gärtner, Heidi Gebauer, Anna Gundert, Michael Hoffmann, Martin Jaggi, Vincent Kusters, Malte Milatz, Torsten Mütze, Jerri Nummenpalo, Gabriel Nivasch, Andrea Salow, Marek Sulovský, May Szedlák, Antonis Thomas, Hemant Tyagi, Uli Wagner, and Manuel Wettstein.

Thanks to students that were brave enough to write a thesis under my supervision: Luca Eggemann, Isabelle Hurbain, Sebastian Millius, Nico Neureiter, Qinheping Hu, and Beat Saurenmann.

Thanks to Isabelle Hurbain, Vincent Kusters, Robin Moser, and Jerri Nummenpalo for proofreading my thesis.

In the end, I would greatly thank my parents for everything, especially for keeping me from despair during my time writing this thesis.
Chapter 1

Introduction

Satisfiability of Boolean formulas (SAT) is one of the most prominent NP-complete problems. Among other things, it has been used to establish the notion of NP-completeness in the famous Cook-Levin theorem [3, 18].

A special case of a Boolean formula is a formula in conjunctive normal form (CNF), i.e. a conjunction of disjunctions of literals. Such a disjunction is commonly called a clause. If all clauses have at most \( k \) literals, we say such a formula is in \((\leq k)\)-CNF, and denote by \( k \)-SAT the decision problem that asks whether such a formula admits a satisfying assignment. In [3] it has also been shown that already 3-SAT is NP-complete; later many other problems were shown to be NP-complete by a reduction from 3-SAT. Unless \( P = NP \), there is no polynomial algorithm for 3-SAT (and \( k \)-SAT with \( k \geq 3 \)), but one can still ask how fast 3-SAT can be solved in the worst case.

For an \((\leq k)\)-CNF formula \( F \) over \( n \) variables, the trivial algorithm to decide satisfiability would simply try all possible \( 2^n \) assignments, and thus would run in time poly\(|F| \cdot 2^n\). This motivates considering the running time of the algorithm not in terms of the length of \( F \), but in terms of \( n \). The best algorithms for \( k \)-SAT found so far are “moderately exponential algorithms”, i.e. algorithms running in time \( O(c^n) \) for \( c < 2 \) (or equivalently \( O(2^{bn}) \), for \( b < 1 \)). Especially for 3-SAT, much effort has been put into improving this constant, and 3-SAT has become some sort of benchmark problem for moderately exponential algorithms. The so-called exponential time hypothesis (ETH) [14] conjectures that there is some fixed lower bound to the above constants \( b \) and \( c \). While for
$k$-SAT many improvements over the trivial algorithm have been made, for the general satisfiability problem, even if the formulas are to be in CNF but with arbitrary clause size, $O(\text{poly}(|F|) \cdot 2^n)$ is essentially the best known. There are some improvements depending on the number of clauses of $F$ by Schuler [34]. When we consider the running time in terms of the length of the formula (the sum of all clause sizes), the best known algorithm is by Wahlström [36]. Although this complexity parameter might seem more natural, it has not received as much attention as the number of variables.

The goal of this thesis is to give faster randomized algorithms for $k$-SAT, and for a more general problem called $(d,k)$-CSP where values are $d$-valued instead of Boolean. We consider only algorithms whose running time depends on the number of variables $n$.

**History of Moderately Exponential Time Algorithms for $k$-SAT.** The first non-trivial algorithm for $k$-SAT is by Monien and Speckenmeyer from 1985 [22]. It runs in time $O(1.619^n)$ for 3-SAT, and in time $2^{(1-O(\frac{1}{\sqrt{k}}))n}$ for $k$-SAT. This algorithm is deterministic, and a so-called branching algorithm, i.e. it covers the space of possible satisfying assignments in a more clever way than trying all $2^n$ assignments. For 3-SAT, there were several improvements on the branching rules, culminating in an $O(1.476^n)$ algorithm in 1996 [26].

The next improvements were obtained by a sequence of inherently randomized algorithms, in the sense that with a certain probability, no satisfying assignment of a satisfiable formula would be found. The first such algorithm, commonly called PPZ after its inventors Paturi, Pudlák, and Zane [25] was presented in 1997 and runs in time $\text{poly}(n)2^{(1-k/\mu_k)n}$ for $k$-SAT (and thus $O(1.588^n)$ for 3-SAT). Despite not being faster for 3-SAT, the improvement for $k$-SAT was huge compared to the branching algorithms; the so-called savings increased from $\Theta(\frac{1}{2^k})$ to $\frac{1}{k}$.

In 1998, the so-called PPSZ algorithm was presented [24] by the inventors of PPZ together with Michael Saks. PPSZ is a direct improvement to the PPZ algorithm, although with a much more complicated analysis. PPSZ for 3-SAT runs in time $O(1.364^n)$; if the input formula is promised to have at most one satisfying assignment (called Unique 3-SAT), the running time is even $O(1.308^n)$. For $k$-SAT, the running time is $\text{poly}(n)2^{(1-k/\mu_k)n}$, with $\mu_k \to \frac{\pi^2}{6}$ for $k \to \infty$.

In 1999, Schöning [33] presented a very simple random walk algorithm, running in $O(1.334^n)$ for 3-SAT, better than PPSZ; however,
worse than PPSZ for both Unique 3-SAT and \( k \)-SAT if \( k \geq 4 \). Further improvements for 3-SAT tried to close this gap. For more details, see Chapter 4, where we also show how to completely close the gap.

Interestingly, the currently fastest deterministic algorithms are based on the randomized algorithms above. Already PPZ [25] was presented together with its own derandomization with worse running time, but savings of the same magnitude (running in time \( 2^{(1-\Theta(\frac{1}{k}))n} \) for \( k \)-SAT). For Unique \( k \)-SAT, PPSZ was derandomized by Rolf [29], but for general \( k \)-SAT no such result is known. The currently fastest deterministic algorithm for \( k \)-SAT is the derandomization of Schöning’s algorithm by Moser and Scheder [23]; for 3-SAT Makino, Tamaki, and Yamamoto [20] improve on this and obtain a running time of \( O(1.3303^n) \). In this thesis, we will only consider randomized algorithms.

**Our Contributions.** In Chapter 3, we re-analyze the PPSZ algorithm for Unique \( k \)-SAT as a foundation for the later chapters. We changed and simplified some aspects compared to the original analysis of PPSZ [24]. This chapter is based on lecture notes produced for a course on Boolean satisfiability by Emo Welzl [10], joint work with Robin Moser and Dominik Scheder.

In Chapter 4, we show that the bounds for Unique \( k \)-SAT of PPSZ also hold for \( k \)-SAT, which was not known for \( k = 3, 4 \). In particular, this gives an algorithm for 3-SAT running in time \( O(1.308^n) \), with a much simpler analysis compared to the previous best 3-SAT algorithms. The results of this chapter are based on [7] with additions by Isabelle Hurbain [13] and Sebastian Millius [21]. The presentation is again based on the lecture notes with Robin Moser and Dominik Scheder [10].

In Chapter 5, we consider the following generalization of \( k \)-SAT: We replace the Boolean domain of the variables with \( d \) possible values. Instead of literals \( x \) or \( \bar{x} \), we have literals disallowing exactly one value, i.e. \( x \neq c \) for some \( c \in \{1, 2, \ldots, d\} \). If every clause has at most \( k \) literals, and every variable has \( d \) possible values, we call the resulting problem \((d,k)\)-CISP. We adapt PPSZ to \((d,k)\)-CISP, obtaining an improvement compared to previous algorithms for \((d,k)\)-CISP for most values \((d,k)\) we considered. This chapter is based on a yet unpublished manuscript, joint work with Isabelle Hurbain, Sebastian Millius, Robin Moser, Dominik Scheder, and May Szedlák.

In Chapter 6, we consider the possibility of algorithms faster than PPSZ. We present an algorithm that is exponentially faster than PPSZ for 3-SAT. For general \( k \) we show that in order to improve on PPSZ
for $k$-SAT, it is enough to improve on PPSZ for Unique $k$-SAT. This chapter is based on [8] and a yet unpublished manuscript.
Chapter 2

Notation

The notation in this thesis is based on the notational framework introduced in [37].

**General Notation.** By $\mathbb{N} = \{1, 2, 3, \ldots \}$ we denote the positive integers (natural numbers), by $\mathbb{N}_0 = \{0, 1, 2, \ldots \}$ the non-negative integers, and by $[d] = \{1, 2, \ldots, d\}$ the set of positive integers from 1 to $d$.

For a set $W$, we denote by $x \leftarrow_{\text{u.a.r.}} W$ the operation of choosing an element $x$ u.a.r. (uniformly at random) from $W$. By choosing an element u.a.r. from an closed real interval, we mean choosing it according to the continuous uniform distribution over this interval. Unless otherwise stated, all random choices are mutually independent.

We consider a permutation on $V$ to be a bijection $\pi : V \to |V|$, ordering $x \in V$ in ascending value of $\pi(x)$. By a random permutation on $V$ we denote a permutation chosen uniformly at random from the set of all permutations on $V$.

We denote by log the logarithm to the base 2. For the logarithm to the base $e$, we write $\ln$, and for a general base $d$ we write $\log_d$. When writing a statement with $\text{poly}(n)$, we mean that there exists a polynomial depending on $n$ for which the statement holds; this will be commonly used to denote polynomial factors in the running time of our algorithms.

By $o(1)$ we denote a quantity dependent on $n$ going to 0 with $n \to \infty$.

The randomized algorithms for the SAT problem we consider here are so-called Monte-Carlo algorithms with one-sided error: They have a fixed running time bound and are always correct on unsatisfiable for-
Chapter 2. Notation

formulas, but they might declare a satisfiable formula unsatisfiable with some probability, say at most $\frac{1}{3}$.

**Satisfiability Notation.** Let $V$ be a finite set of propositional variables. A literal $u$ over $x \in V$ is a variable $x$ or a negated variable $\bar{x}$. If $u = \bar{x}$, then $\bar{u}$, the negation of $u$, is defined as $x$. We mostly use $x, y, z$ for variables and $u, v, w$ for literals. We assume that all literals are distinct.

A clause over $V$ is a finite set of literals over pairwise distinct variables from $V$. By $\text{vbl}(C)$ we denote the set of variables that occur in $C$, i.e. \{x \in V \mid x \in C \lor \bar{x} \in C\}. C$ is a $k$-clause if $|C| = k$ and it is a $(\leq k)$-clause if $|C| \leq k$. A formula $F$ in CNF (conjunctive normal form) over $V$ is a finite set of clauses over $V$. We define $\text{vbl}(F) := \bigcup_{C \in F} \text{vbl}(C)$. $F$ is a $k$-CNF formula (an $(\leq k)$-CNF formula) if all clauses of $F$ are $k$-clauses ($(\leq k)$-clauses). In running time bounds, we commonly denote by $n$ the number of variables of the input formula.

A (truth) assignment on $V$ is a function $\alpha : V \to \{0, 1\}$ which assigns a Boolean value to each variable. $\alpha$ extends to negated variables by letting $\alpha(\bar{x}) := 1 - \alpha(x)$. A literal $u$ is satisfied by $\alpha$ if $\alpha(u) = 1$. A clause is satisfied by $\alpha$ if it contains a satisfied literal and a formula is satisfied by $\alpha$ if all of its clauses are. A formula $F$ is satisfiable if there exists an assignment satisfying $F$. A formula that is not satisfiable is called unsatisfiable.

Given an assignment $\alpha$ on $U$ and $\beta$ on $V$, we call $\alpha$ and $\beta$ to be compatible if $\alpha(x) = \beta(x)$ for all $x \in U \cap V$. We will write $\alpha \subseteq \beta$ if furthermore $U \subseteq V$.

A partial assignment $\alpha$ on $V$ is a partial function from $V$ to $\{0, 1\}$ (sometimes denoted $\alpha : V \not\rightarrow \{0, 1\}$), i.e. a function that might leave some variables unset. A partial assignment $V$ can be considered as an assignment on some subset $W \subseteq V$ and will be treated as such.

For an assignment $\alpha$ on $V$ and $W \subseteq V$, we denote by $\alpha|_{W}$ the restriction of $\alpha$ to $W$, that is the (unique) assignment on $W$ compatible with $V$. For an assignment $\alpha$ on $V$, we define $\text{dom}(\alpha) := V$, the domain of $\alpha$. For a partial assignment $\beta$ the domain consists of the variables assigned by $\beta$ (this is consistent with how we treat partial assignments as assignments on a subset of the variables).

For an assignment $\alpha$ on $V$ and $\beta$ on $W$, we denote by $\alpha[\beta]$ the assignment on $V \cup W$ that agrees with $\beta$ on $W$ and with $\alpha$ on $V \setminus W$ (i.e. we start with $\alpha$ and add the assignment $\beta$, overwriting the values of the variables also set by $\alpha$).

Given a CNF formula $F$ and a variable set $V$, we denote by $\text{sat}_{V}(F)$
the set of assignments on $V$ that satisfy $F$. If $V$ is clear from the context, we also write sat($F$).

$k$-SAT is the decision problem of deciding whether an ($\leq k$)-CNF formula has a satisfying assignment. (Promise) Unique $k$-SAT is the promise problem of deciding whether an ($\leq k$)-CNF formula $F$ over $V$ has either exactly one or no satisfying assignment on $V$. We will always consider $k$ to be constant. As 2-SAT is solvable in polynomial time and we are interested in exponential time algorithms, we only consider $k$-SAT for $k \geq 3$.

If $F$ is a CNF formula and $x$ a variable (not necessarily in vbl($F$)), we write $F[x \mapsto \downarrow 1]$ ($F[x \mapsto \downarrow 0]$) for the formula arising from removing all clauses containing $x$ ($\bar{x}$) and truncating all clauses containing $\bar{x}$ ($x$) to their remaining literals. This corresponds to assigning 1 (0) to $x$ in $F$ and removing trivially satisfied clauses. If $\gamma = \{x \mapsto 0, y \mapsto 1, \ldots \}$ is a (partial) assignment, we write $F[\gamma]$ as a shorthand for $F[x \mapsto 0][y \mapsto 1][\ldots]$, the restriction of $F$ to $\gamma$.

**ClSP Notation.** In Chapter 5 we will consider so called clause satisfaction problems (ClSP). Many notions here have the same name as the corresponding notion for SAT, however it will be clear from the context whether we are considering ClSP or SAT.

Let $V$ be a finite set of propositional variables. A literal over $x \in V$ is of the form ($x \neq c$) for some value $c \in \mathbb{N}$. A clause over $V$ is a finite set of literals over pairwise distinct variables from $V$.

We denote both the decision problem and its instances by clause satisfaction problem (ClSP); sometimes we call the latter formulas. A ClSP (formula) $F$ over $V$ is a a finite set of clauses over $V$. By vbl($F$) we denote the set of variables appearing in $F$. We say that ClSP $F$ is a $(d,k)$-ClSP if every clause of $F$ has size at most $k$ and if the values appearing in the literals are all in $\{1,2,\ldots,d\}$.

A $(d$-valued) assignment on $V$ is a function $\alpha : V \rightarrow \{1,2,\ldots,d\}$ which assigns a value to each variable. A literal $u = (x \neq c)$ is satisfied by $\alpha$ if $\alpha(x) \neq c$. A clause is satisfied by $\alpha$ if it contains a satisfied literal. A formula is satisfied by $\alpha$ if all its clauses are satisfied.

The decision problem $(d,k)$-ClSP asks whether a $(d,k)$-ClSP formula has a satisfying $d$-valued assignment. Unique $(d,k)$-ClSP is the promise problem of deciding whether a $(d,k)$-ClSP $F$ over $V$ has exactly one or no $d$-valued satisfying assignment (on $V$).

An assignment that assigns a single variable is called a value assignment, e.g. ($x \mapsto c$). In the Boolean case there is a unique value
assignment satisfying a literal, giving rise to a natural correspondence. Here this is not the case anymore. Abusing notation, we will consider an assignment as a set of value assignments, exactly one per assigned variable.

The remaining notation is analogous to the satisfiability notation. For sat$_V(F)$ and sat($F$), $d$ will be clear from the context.
Chapter 3

The PPSZ Algorithm for Unique $k$-SAT

In this chapter we will introduce the PPSZ algorithm [24] and state some general properties. Then we reprove the results of [24] for Unique $k$-SAT with some differences. The goal is to make the proof both easier to understand and the results easier to work with. We will make the differences explicit; unless noted this chapter is directly based on [24]. This new analysis is joint work with Robin Moser and Dominik Scheder [10]. The analysis for general $k$-SAT is considered in the next chapter.

One possible (randomized) approach to find a satisfying assignment of an ($\leq k$)-CNF formula $F$ is the following: Choose a random order $\pi$ on the variables and fix them one by one in this order. Set each variable $x$ to your best knowledge under the assumption that all previous choices can still be extended to a satisfying assignment. Here, “to your best knowledge” means that you are allowed to look at all possible subformulas of (at most) $D$ clauses $G \subseteq F$, for some number $D$. For each such subformula $G$, you can check which of $x \mapsto 1$ and $x \mapsto 0$ occur in a satisfying assignment of $G$, compatible with your previous choices. If only one assignment to $x$ is possible then any compatible satisfying assignment of $G$ and hence $F$ must set $x$ to that value. If for all subformulas $G$ both $x \mapsto 1$ and $x \mapsto 0$ occur in a compatible satisfying assignment, then we don’t know anything about $x$ and we guess the value of $x$ uniformly at random from $\{0,1\}$. For constant $D$, this so-called $D$-implication can be checked in polynomial time by brute-force (we see shortly how exactly).
Note that if \( G \) has no compatible satisfying assignment, any literal is \( D \)-implied; however, in this case we have no possibility to extend the current assignment to a satisfying assignment of \( G \) and hence \( F \) anymore (violating the above assumption), and the algorithm should return failure for the current try.

For \( D = 1 \), this algorithm is called PPZ after its inventors Paturi, Pudlák, and Zane from 1997 [25]. For general \( D \) this algorithm is essentially the algorithm PPSZ by Paturi, Pudlák, Saks, and Zane from 1998 [24], an extension of PPZ. It is noteworthy that these algorithms were presented in the context of lower bounds on so-called depth 3 circuits. We will just consider the algorithmic aspect, however.

### 3.1 General Properties of PPSZ

More formally, we define \( D \)-implication and general implication as follows:

**Definition 3.1.** Let \( F \) be a CNF formula, let \( D \) be a number, and let \( u \) be a literal. We say that \( F \) implies \( u \), in writing \( F \models u \), if any assignment \( \alpha \) satisfying \( F \) also satisfies \( u \). We say \( F \) \( D \)-implies \( u \), in writing \( F \models_D u \), if there is some \( G \subseteq F \) with \(|G| \leq D \) having \( G \models u \).

In this terminology, in PPZ we check whether \( x \) or \( \bar{x} \) is 1-implied in \( F \). The PPSZ algorithm applies an obvious improvement: Check whether \( x \) or \( \bar{x} \) is \( D \)-implied for some \( D \). To analyze PPSZ formally, we think of it choosing a random permutation \( \pi \) of \( \text{vbl}(F) \) and an assignment \( \beta \in \{0,1\}^n \) in advance. Then it processes the variables according to \( \pi \) and assigns them values according to \( \beta \), unless they turn out to be \( D \)-implied (by \( F \) restricted to the assignment built so far), in which case it assigns them accordingly. If at the step of \( x \), both \( x \) and \( \bar{x} \) are \( D \)-implied, then the current run cannot find a satisfying assignment anymore. Instead of directly reporting failure (as a practical implementation would), to make notation easier, we arbitrarily assign 1 to \( x \) and continue, so that PPSZ always returns some assignment on \( V \). This procedure is listed in Algorithms 1 and 2.

In the following we assume as usual that \( k \geq 3 \) is constant. First we observe that we can check \((\log n)\)-implication in subexponential time:

**Observation 3.2.** For an \((\leq k)\)-CNF formula \( F \) over \( n \) variables and a literal \( u \), \( F \models_D u \) can be decided in time

\[
O(n^{kD} \cdot 2^{kD} \cdot \text{poly}(n)).
\]
3.1. General Properties of PPSZ

Algorithm 1 PPSZ(CNF formula $F$, variable set $V$, integer $D$)

Choose $\beta$ u.a.r. from all assignments on $V$
Choose $\pi$ u.a.r. from all permutations on $V$
return $\text{PPSZ}(F, V, \beta, \pi, D)$

Algorithm 2 PPSZ(CNF formula $F$, variable set $V$, assignment $\beta$, permutation $\pi$, integer $D$)

Let $(x_1, x_2, \ldots, x_{|V|})$ be the set $V$ ordered according to $\pi$
Let $\gamma$ be a partial assignment on $V$, initially empty
for $i \leftarrow 1$ to $|V|$ do
if $F[\gamma] \models_D x_i$ then
$\gamma(x_i) \leftarrow 1 \{x_i$ is forced to 1}$
else if $F[\gamma] \models_D \neg x_i$ then
$\gamma(x_i) \leftarrow 0 \{x_i$ is forced to 0}$
else
$\gamma(x_i) \leftarrow \beta(x_i) \{x_i$ is guessed}$
end if
end for
return $\gamma$

If $D$ is constant, this is polynomial in $n$. If $D = \log n$, this is $2^{o(n)}$ (i.e. subexponential in $n$).

Proof. We can use the following brute-force approach: Choose $D$ clauses of the $O(n^k)$ many, then try all at most $2^{kD}$ assignments on the variables of the chosen clauses. The other bounds are an immediate consequence.

In the end we will let $D = \log n$. This choice is arbitrary; for the analysis any $D$ growing in $n$ will do. On the other hand, any choice $D = o\left(\frac{n}{\log n}\right)$ will result in subexponential running time; by being a bit more clever even $D = o(n)$ is possible. However, as far as our analysis goes we do not get any additional benefit compared to choosing $D$ to be an arbitrary (slowly) growing function in $n$ (the subexponential factors will change but we ignore these, as we are only interested in the exponential complexity).

Note that in [24], a stronger concept called bounded resolution was used in the place of $D$-implication. However, for all known purposes of the analysis, $D$-implication is already sufficient. It has the advantage
that it is based on logical implication instead on the syntactic resolution proof system; we think that this makes some arguments easier to understand. For future research it should be kept in mind, however, that stronger definitions of “to your best knowledge” exist.

The bad thing to happen in PPSZ is if we have to guess the value of a variable, as in this case we might guess wrongly (sometimes both choices are viable, more about this in the next chapter). The main statement of this chapter is that for \((\leq k)\)-CNF formulas with a unique satisfying assignment we won’t have to guess too often. From this, as we see shortly, we obtain a lower bound on the probability that PPSZ finds the unique satisfying assignment.

To make this formal we need to define the concept of forced and guessed variables.

**Definition 3.3.** Let \(F\) be a CNF formula over variables \(V\), \(\alpha \in \{0,1\}^V\) a satisfying assignment, \(\pi = (x_1,\ldots,x_{|V|})\) a permutation of \(V\), and \(D\) be a number. A variable \(x_i\) is called forced (with respect to \(F, V, \alpha, \pi, \) and \(D\)) if \(F[\{x_1 \mapsto \alpha(x_1), \ldots, x_{i-1} \mapsto \alpha(x_{i-1})\}]\) \(D\)-implies \(x_i\) or \(\bar{x}_i\). Otherwise the variable is called guessed. We denote the set of forced (guessed) variables within \(V\) by \(\text{Forced}(F,V,\alpha,\pi,D)\) (\(\text{Guessed}(F,V,\alpha,\pi,D)\), resp.).

It is important to note that in the above definition the forced and guessed variables are not with respect to a random PPSZ execution, but to an execution where the random assignment \(\beta\) has been fixed to a satisfying assignment \(\alpha\) (of which there might be multiple). In this definition \(\pi\) is fixed, but later we consider random \(\pi\). The naive definition that would retain the randomness of \(\beta\) seems useless here since the following analysis requires that all variables have been chosen w.r.t. a fixed satisfying assignment. Especially if \(F\) becomes unsatisfiable during the execution of the algorithm, the analysis doesn’t say anything, as it only considers satisfiable formulas.

The crucial observation here is that if, for some satisfying assignment \(\alpha\) of \(F\), all guessed variables (w.r.t. \(\alpha\) and \(\pi\)) have the same value in \(\alpha\) and \(\beta\), then PPSZ indeed returns \(\alpha\). The intuition for this is that forced variables are always forced “correctly” in the sense that all satisfying assignments compatible with the assignment built so far in PPSZ agree on the forced variable, and namely assign it the value it is forced to.

**Observation 3.4.** Let \(F\) be a CNF formula over \(V\) and \(\alpha\) be a satisfying assignment on \(V\). Then PPSZ\((F,V,\beta,\pi,D)\) returns \(\alpha\) if and only if \(\alpha(x) = \beta(x)\) for all \(x \in \text{Guessed}(F,V,\alpha,\pi,D)\).
3.1. General Properties of PPSZ

Proof. Let \( \pi = (x_1, x_2, \ldots, x_{|V|}) \). We proceed by induction on the steps \( i = 1, \ldots, |V| \) of PPSZ. First, suppose \( \alpha(x) = \beta(x) \) for all \( x \in \text{Guessed}(F, V, \alpha, \pi, D) \). Assume that for all \( j < i \), \( \gamma(x_j) = \alpha(x_j) \), where \( \gamma \) is the partial assignment built by PPSZ just before step \( i \). If \( x_i \) is guessed by PPSZ, then clearly \( x_i \in \text{Guessed}(F, V, \alpha, \pi, D) \); hence \( x_i \) is assigned \( \beta(x_i) = \gamma(x_i) \). Otherwise \( x_i \) is forced by PPSZ. However, \( \alpha \) satisfies \( F \) and since \( \gamma \subseteq \alpha \), \( \alpha \) also satisfies \( F[\gamma] \), and any subformula \( G \) of it. Therefore \( x_i \) must be forced to \( \alpha(x_i) \), as every subformula \( G \) has an assignment that sets \( x_i \) to \( \alpha(x_i) \), namely \( \alpha \). Thus, by induction, every variable is assigned according to \( \alpha \) and PPSZ returns \( \alpha \).

For the other direction, suppose that \( \alpha(x) \neq \beta(x) \) for some \( x \in \text{Guessed}(F, V, \alpha, \pi, D) \), and let \( x_i \) be the first such variable in \( \pi \). From the previous paragraph, we know that all \( x_j \) for \( j < i \) have indeed been assigned according to \( \alpha \). Hence \( x_i \) is guessed in PPSZ here and assigned \( \beta(x) \); since PPSZ does not change variables once fixed, it cannot return \( \alpha \) anymore.

By the above observation, PPSZ returns \( \alpha \) if and only if \( \beta \) agrees with \( \alpha \) on all variables from \( \text{Guessed}(F, V, \alpha, \pi, D) \); for fixed \( \pi \) this happens with probability \( 2^{-|\text{Guessed}(F, V, \alpha, \pi, D)|} \). When considering \( \beta \) and \( \pi \) to be random (as in PPSZ) then the probability that PPSZ returns \( \alpha \) is the expectation of the above probability:

\[
\Pr_{\beta, \pi} (\text{PPSZ}(F, V, \beta, \pi, D) \text{ returns } \alpha) = E_{\pi} \left[ 2^{-|\text{Guessed}(F, V, \alpha, \pi, D)|} \right]
\]

This expression is rather difficult to work with. Fortunately there is a nice trick we can do here: we can apply Jensen’s inequality (see Appendix, Theorem A.3) on the convex function \( x \mapsto 2^{-x} \) (or equivalently \( x \mapsto 2^x \)). Doing this, we obtain

\[
\Pr_{\beta, \pi} (\text{PPSZ}(F, V, \beta, \pi, D) \text{ returns } \alpha) \geq 2^{-E_{\pi}[|\text{Guessed}(F, V, \alpha, \pi, D)|]}.
\]  

(3.1)

Now we are left to analyze \( E_{\pi}[|\text{Guessed}(F, V, \alpha, \pi, D)|] \), i.e. the expected size of a certain set. A standard application of linearity of expectation is the fact that the expected size of a set is just the sum of the probabilities that each potential member is in this set. Thus, applying Jensen’s inequality allows us to ignore all correlations between the variables; we will use this rather often in the following chapters. As noted,

\[
E_{\pi}[|\text{Guessed}(F, V, \alpha, \pi, D)|] = \sum_{x \in V} \Pr_{\pi}(x \in \text{Guessed}(F, V, \alpha, \pi, D)),
\]
and we are left to compute $\Pr_\pi (x \in \text{Guessed}(F, V, \alpha, \pi, D))$, the probability that a fixed variable is guessed. If a unique satisfying assignment is promised, these probabilities can be bounded from above, resulting in a lower bound on the probability that PPSZ returns the unique satisfying assignment. First we define $S_k$, which will be the (idealized) upper bound that a variable is guessed in a uniquely satisfiable ($\leq k$)-CNF.

Definition 3.5. $S_k := \int_0^1 \frac{t^{1/(k-1)} - t}{1-t} dt$

For $k = 3$, one can show that $S_3 = 2 \ln 2 - 1$. For small $k$, $S_k$ and $2^{S_k}$ are approximately (rounded up):

<table>
<thead>
<tr>
<th>$k$</th>
<th>$S_k$</th>
<th>$2^{S_k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.3862944</td>
<td>1.307032</td>
</tr>
<tr>
<td>4</td>
<td>0.5548182</td>
<td>1.468984</td>
</tr>
<tr>
<td>5</td>
<td>0.6502379</td>
<td>1.569427</td>
</tr>
<tr>
<td>6</td>
<td>0.7118243</td>
<td>1.637874</td>
</tr>
</tbody>
</table>

Theorem 3.6 ([24]). If $F$ is an ($\leq k$)-CNF over $V$ with a unique satisfying assignment, then

$$\Pr_\pi (x \in \text{Guessed}(F, V, \alpha, \pi, D)) \leq S_k + \epsilon_k^{(D)},$$

where $\epsilon_k^{(D)}$ goes to 0 for $D \to \infty$ (for fixed $k$).

This theorem together with (3.1) and Observation 3.2 about the running time gives the following corollary:

Corollary 3.7 ([24]). If $F$ is an ($\leq k$)-CNF over $n$ variables $V$ with a unique satisfying assignment $\alpha$, then

$$\Pr (\text{PPSZ}(F, V, \log n) = \alpha) \geq 2^{-(S_k + o(1))n}.$$  

Furthermore, PPSZ($F, V, \log n$) runs in time $2^{o(n)}$ (i.e. in subexponential time).

So far, PPSZ is an algorithm that runs in subexponential time and finds, for a uniquely satisfiable formula, the satisfying assignment with exponentially small probability. To turn this into a randomized algorithm for Unique $k$-SAT, we need to repeat PPSZ, proportional to the inverse success probability. The runs are all independent; once a satisfying assignment is found, it is returned and the algorithm stops. If no satisfying assignment has been found, then the formula is declared
unsatisfiable. As PPSZ cannot erroneously produce a satisfying assignment of an unsatisfiable formula (there exists none), we only need to consider PPSZ for satisfiable formulas. For a formal argument, see Lemma A.2 in the appendix.

**Corollary 3.8.** PPSZ can be turned into a randomized algorithm for Unique $k$-SAT running in time $2^{(S_k + o(1))n}$.

The only difference to [24] in this section is the usage of $D$-implication instead of bounded resolution.

### 3.2 Bounding the Guessed Probability of Unique $k$-SAT

In the remainder of this chapter, we (re-)prove Theorem 3.6. In the following let $F$ be an $(\leq k)$-CNF formula over $V$ with a unique satisfying assignment $\alpha$.

The first step is to define so-called critical clause trees that represent the structure of a uniquely satisfiable $(\leq k)$-CNF formula ensuring that $x$ is $D$-implied with large enough probability (and hence guessed with small enough probability).

#### 3.2.1 Motivation

Let us motivate the following steps and critical clause trees with the following example. Suppose $F$ over $V = \{x, \ldots\}$ is uniquely satisfied by the all-one assignment $\alpha$. At the step of $x$, one might ask the question why $x$ cannot be set to 0. Let us try to set $x$ to 0 in $\alpha$ and see what happens. Let $\alpha_x$ be the assignment that assigns 0 to $x$ and 1 to all other variables. As $\alpha$ is the unique satisfying assignment, $\alpha_x$ does not satisfy $F$, meaning that there is clause $C_x \in F$ not satisfied by $\alpha_x$ (but satisfied by $\alpha$). It is not hard to see that $C_x$ must consist of a positive literal $x$ and possibly some negative literals over other variables, i.e. $C_x$ has a unique satisfied literal. We call such a clause $C_x$ a critical clause (critical clauses will be formally defined and used in Chapter 6, here they only serve to motivate critical clause trees). If all other variables of $C_x$ come before $x$ in PPSZ and are set to 1 (i.e. according to $\alpha$), then $x$ will be 1-implied at the step of $x$. This happens with probability $\frac{1}{|C_x|}$; thus giving an upper bound of $1 - \frac{1}{k}$ to $\Pr_\pi (x \in \text{Guessed}(F, V, \alpha, \pi, 1))$; this is the basic idea of the PPZ algorithm [25], the predecessor of PPSZ.
However, PPSZ is allowed to look at more than just one clause. Suppose the critical clause $C_x$ found above is $\{x, \overline{y}, \overline{z}\}$. This clause tells us that we cannot set $x$ to 0, unless either $y$ or $z$ are also set to 0. If both $y$ and $z$ come before $x$ in PPSZ and are assigned 1 (i.e. according to $\alpha$), then we are fine. However, suppose $y$ comes after $x$ in PPSZ (but $z$ still comes before $x$ and is assigned 1). The correct question to ask is now “why cannot both $x$ and $y$ be set to 0?”, as if either one of $x$ and $y$ would be set to 1, we will be fine. Analogous to before, let $\alpha_{xy}$ be the assignment setting $x$ and $y$ to 0 and all other variables to 1, and let $C_{xy}$ be a clause of $F$ not satisfied by $\alpha_{xy}$. $C_{xy}$ must have some positive literals over $\{x, y\}$, and might have negative literals over other variables; for example $C_{xy} = \{x, y, \overline{a}\}$. If here in addition to $z$, also $a$ comes before $x$ in PPSZ and is assigned 1, then we can infer from $C_x$ and $C_{xy}$ that $x$ must be set to 1: By $C_{xy}$ either $x$ or $y$ must be 1. If $x$ is 1, we are done; if $y$ is 1, together with the assumption that $z$ is 1, $C_x$ implies that $x$ must be set to 1. Hence $x$ is 2-implied. Critical clause trees formalize this type of argument and give sufficient conditions that a variable $x$ is $D$-implied. Note that $C_{xy}$ is not a critical clause. The name critical clause tree comes from the slightly different approach used to analyze PPSZ originally [24]; there $C_x$ and $C_{xy}$ would be combined by resolution over $y$ to create the new critical clause $\{x, \overline{z}, \overline{a}\}$. Apart from this, we could call the trees just as well $D$-implication trees.

### 3.2.2 Building Critical Clause Trees

We now construct a collection of $k-1$-ary trees $\{T_x\}_{x \in V}$, with $T_x$ called a critical clause tree of $x$, which in some way represents multiple ways $x$ can become $D$-implied during a run of the PPSZ algorithm. For a tree $T$, we denote by $V(T)$ the set of nodes of $T$. The depth of a node is its distance to the root (where the depth of the root itself is 0). The height of $T$ is the maximum depth of a node of $T$.

$T_x$ is a rooted $k-1$-ary tree (where every node has at most $k-1$ children) where every node $u \in V(T)$ is labeled both with a variable $x \in V$, which we denote by var-label($u$), and with a clause $C \in F$, denoted by clause-label($u$). Even though there is some choice in how these trees are built, we make such choices arbitrarily once and for all and then consider the collection $\{T_x\}_{x \in V}$ to be fixed for the remainder of the chapter. Here is how $T_x$ is built for a fixed $x \in V$:

1. Start with $T_x$ consisting of a single root. This root has variable label $x$, and does not have a clause label yet.
2. As long as there is a leaf $u \in V(T)$ that does not yet have a clause label, do the following:

(a) Define $W := \{\text{var-label}(v) \mid v \in V(T) \text{ ancestor of } u \text{ in } T\}$, where an ancestor of $u$ is a node on the path from $u$ to the root, including $u$ itself.

(b) Define the assignment $\beta$ as

$$\beta : \text{vbl}(F) \to \{0, 1\}, \quad z \mapsto \begin{cases} 1 - \alpha(z) & \text{if } z \in W, \\ \alpha(z) & \text{otherwise.} \end{cases}$$

(c) Let $C \in F$ be a clause not satisfied by $\beta$. Since $\beta \neq \alpha$ and $\alpha$ is the unique satisfying assignment, such a clause exists. Set clause-label($u$) := $C$.

Observe that this means that all literals of $C$ satisfied by $\alpha$ must be over $W$.

(d) For each literal $w \in C$ which has $\alpha(w) = 0$, create a new leaf, label it with the variable underlying $w$, and attach it to $u$ as a child. The new leaf does not yet have a clause label.

We denote the resulting tree by $T_x$. Note that every clause in $F$ has at most $k - 1$ literals violated by $\alpha$, and thus in step (d) we append at most $k - 1$ children. Suppose $v$ is an ancestor of $u$ and var-label($v$) = $y$. Since $\beta(y) \neq \alpha(y)$, there is no $\alpha$-violated literal over $y$ contained in clause-label($u$). Therefore,

**Observation 3.9.** In $T_x$, no node has the same var-label as one of its ancestors.

This also implies that the height of the tree cannot exceed $n - 1$, thus the process terminates, making $T_x$ well-defined.

### 3.2.3 Connection to Forced Variables

We now establish a connection between critical clause trees and forced variables useful for estimating the probability with which a variable is forced.

To simplify the calculations that await us, we introduce the notion of placements. A placement of the variables in $V$ is a function $\pi : V \to [0, 1]$. Placements can be seen as continuous permutations. Especially, if the values $\pi(x)$ are chosen independently and uniformly
at random from $[0, 1]$ for each $x \in V$, then with probability 1, $\pi$ is injective, and by sorting $V$ according to the values $\pi(x)$ we obtain a uniformly distributed permutation of $V$ by symmetry. The reason why we introduced placements is that for a (uniformly) random placement the places $\pi(x)$ for $x \in V$ are mutually independent; in contrast to the positions of the variables in a random permutation.

Let $\gamma \in [0, 1]$ and $T_x$ be the critical clause tree of some fixed variable. We call a node $u \in T_x$ reachable at time $\gamma$ w.r.t. $\pi$ if there exists a path from the root to $u$: root = $v_0, v_1, \ldots, v_m = u$ in $T_x$, such that $\pi(\text{var-label}(v_i)) \geq \gamma$ for all $1 \leq i \leq m$. Let us denote by Reachable($T_x, \gamma, \pi$) the set of all nodes in $T$ reachable at time $\gamma$ w.r.t. $\pi$.

**Lemma 3.10.** If we have $|\text{Reachable}(T_x, \pi(x), \pi)| \leq D$, then $x \in \text{Forced}(F, V, \alpha, \pi, D)$.

The following proof is different compared to the proof of [24] as we use $D$-implication instead of bounded resolution. This allows us to consider the underlying semantic properties of critical clause trees directly.

**Proof.** Let $\alpha'$ be the restriction of $\alpha$ to the variables $y \in V$ with $\pi(y) < \pi(x)$. By definition, $x$ is forced if there is a formula $F' \subseteq F[\alpha']$ with $|F'| \leq D$ that implies $x$ or $\bar{x}$. Let $G$ be the clause labels of Reachable($T_x, \pi(x), \pi$), i.e. the subformula of $F$ consisting of all clause labels of reachable nodes in $T_x$. Clearly $|G| \leq D$. Let $l$ be the literal over $x$ satisfied by $\alpha$ ($x$ if $\alpha(x) = 1$ and $\bar{x}$ if $\alpha(x) = 0$). We claim that $G[\alpha']$ implies $l$. This proves the statement, as $G[\alpha'] \subseteq F[\alpha']$.

Suppose, for the sake of contradiction, that $G[\alpha']$ does not imply $l$. Then we can fix an assignment $\beta: V \rightarrow \{0, 1\}$ which is compatible with $\alpha'$, which has $\beta(x) \neq \alpha(x)$ and which satisfies $G$. Choose a maximal path in $T_x$ starting at the root and containing only nodes $v$ such that $\beta(\text{var-label}(v)) \neq \alpha(\text{var-label}(v))$. Since $\beta(x) \neq \alpha(x)$, this path is non-empty. Let $u$ be its endpoint. The node $u$ is reachable as $\beta$ is compatible with $\alpha$ on all variables ranking before $x$ in $\pi$. All children of $u$ are labeled with some variable $z$ for which $\beta(z) = \alpha(z)$, and all ancestors with some variable $y$ for which $\beta(y) \neq \alpha(y)$. By construction, all literals of clause-label($u$) satisfied by $\alpha$ must be over some variable occurring on the path from $u$ to the root. Also the variables of literals of clause-label($u$) not satisfied by $\alpha$ are exactly the var-labels of the children of $u$. Thus clause-label($u$) is unsatisfied by $\beta$, a contradiction. $\square$
The lemma tells us how to proceed with bounding the probability that a variable is forced. Namely, it follows immediately that over uniform choice of $\pi$, we have

$$\Pr[x \in \text{Forced}(F, V, \alpha, \pi, D)] \geq \Pr[|\text{Reachable}(T_x, \pi(x), \pi)| \leq D]. \quad (3.2)$$

This reduces the problem to a probabilistic calculation on $k-1$-ary trees: what is the probability that when sorting the nodes of a fixed $k-1$-ary tree according to a random permutation (caveat: some nodes have the same label and are prescribed to get assigned the same rank) and deleting all nodes which rank after the root, there will be at most $D$ nodes reachable? We will prove:

**Theorem 3.11.** For any $\epsilon > 0$ there exists a $D_k(\epsilon) \in \mathbb{N}$ such that the following holds. Let $X_1, X_2, \ldots, X_r \in [0, 1]$ be real random variables distributed uniformly and mutually independently. Let $T$ be any finite $k-1$-ary tree and $\sigma : V(T) \to \{1 \ldots r\}$ a labeling of the nodes of $T$ such that on each path from the root to a leaf of $T$, $\sigma$ is injective. Consider the experiment of drawing $X_1, X_2, \ldots, X_r$ according to their distribution and then deleting all nodes $u$ from $T$ (along with the corresponding subtrees) for which $X_{\sigma(u)} < X_{\sigma(\text{root})}$. Then for the probability that the resultant tree $T'$ contains more than $D_k(\epsilon)$ nodes we have

$$\Pr(|V(T')| > D_k(\epsilon)) \leq S_k + \epsilon.$$

For later use, let us also introduce the notation

$$S_k^{(D)} := \max\{S_k, \sup_T (\Pr(|V(T')| > D))\},$$

where the supremum is over all choices of finite trees with labels $T$ (as in the theorem) and $T'$ is the random tree arising from $T$ by conducting the experiment described in the theorem. In this language, the theorem states that $\lim_{D \to \infty} S_k^{(D)} \leq S_k$, where the limit exists because $S_k^{(D)}$ is trivially monotonic and bounded. We require $S_k^{(D)} \geq S_k$ for technical reasons in the next chapter.

### 3.3 Random Deletion in $k-1$-ary Trees

This section is aimed at proving Theorem 3.11. Note that this theorem speaks purely about probabilities in a combinatorial random experiment on $k-1$-ary trees some of whose vertices get deleted. There is no reference to the PPSZ algorithm, so this is a self-contained proof.
3.3.1 Of Infinite and Finite Trees

To prove Theorem 3.11, we will proceed in four high-level steps. First, we consider a much simpler case: an infinite full \( k-1 \)-ary tree from which every node (with subtree) is deleted independently with a fixed probability \( p \), and ask what is the probability that what is remaining is of finite size. Then we show that the probability that the remaining tree has at most height \( H \) is not much smaller for large \( H \).

Obviously we can embed any finite \( k-1 \)-ary tree into the infinite one, so the only remaining problem are the dependencies between the nodes. We show that these dependencies can only help our case.

In the end, we ask what happens if the deletion probability \( p \) is also random, namely if it is chosen u.a.r. from \([0, 1]\). Also, we observe that a \( k-1 \)-tree with bounded height also has a bounded number of nodes.

For the first step, consider an infinite \( k-1 \)-ary tree \( T \), where each node (along with its subtree) is deleted with probability \( p \), independently. Call the remaining tree \( T' \). What is the probability that \( T' \) is finite? Denote this probability by \( q \).

Observe that \( T' \) is finite if and only if for each child \( u \) of the root of \( T' \) either (1) has been deleted or (2) the subtree rooted at \( u \) is finite. (1) happens with probability \( p \), and (2) happens by symmetry with the probability \( q \). Hence for one child \( u \) the condition holds with probability \( p + (1 - p)q \), and by independence we establish the relation

\[
q = (p + (1 - p)q)^{k-1}. \tag{3.3}
\]

Possible values for \( q \) are solutions to this polynomial equality.

**Definition 3.12.** Denote \( R_k(p) \) to be the smallest non-negative solution to (3.3). This is well-defined, as \( q = 1 \) is a solution, and a non-trivial polynomial equation can only have finitely many solutions.

**Observation 3.13.** Let \( T_\infty \) be the infinite rooted full \( k-1 \)-ary tree. Consider the following random experiment: each non-root node from \( T_\infty \) is deleted (along with its subtree) independently from all other nodes with probability \( p \). Then the probability that the resultant tree \( T' \) is of finite size is

\[
\Pr (T' \text{ finite}) \geq R_k(p).
\]

\(^{1}\)The existence of this probability might be unclear. One approach is to model the number of remaining children at height \( i \) by a Markov chain; the corresponding event will be whether the Markov chain reaches 0. For such Markov chains probability theoretic foundations are well established.
3.3. Random Deletion in \( k - 1 \)-ary Trees

**Proof.** The above probability satisfies (3.3) and it must be non-negative, so the statement follows from the definition of \( R_k \).

We can get a closed form of \( R_3(p) \). For general \( k \), we can characterize \( R_k(p) \) by its inverse:

**Lemma 3.14.** For \( k = 3 \), we have

\[
R_3(p) = \begin{cases} 
\frac{p^2}{(1-p)^2} & \text{if } p < \frac{1}{2}, \\
1 & \text{otherwise}.
\end{cases}
\]

In general, define \( S_k(t) = \frac{t^{\frac{1}{k-1}} - t}{1-t} \) for \( t \in [0, 1) \), and \( S_k(1) = \frac{k-2}{k-1} \). Then

- \( S_k \) is monotonically non-decreasing and continuous on \([0, 1]\).
- For \( p \in [0, \frac{k-2}{k-1}] \), \( R_k(p) \) is the inverse of \( S_k(t) \).
- For \( p \in [\frac{k-2}{k-1}, 1] \), \( R_k(p) = 1 \).
- \( R_k \) is monotonically non-decreasing and continuous on \([0, 1]\).

The statement for \( R_3 \) is easy to check; for general \( k \) the proof is rather technical and we defer it to the appendix.

If \( T \) is any (finite or infinite) tree, let us denote by \( h(T) \) the height of \( T \); if \( T \) is infinite we let \( h(T) = \infty \). We can extend the previous lemma as follows.

**Lemma 3.15.** Let \( T_\infty \) be the infinite rooted full \( k - 1 \)-ary tree. Consider the following random experiment: each non-root node from \( T_\infty \) is deleted (along with its subtree) independently from all other nodes with probability \( p \). Then the probability that the resultant tree \( T' \) has height at most \( H \geq 1 \) converges as

\[
\lim_{H \to \infty} \Pr (h(T') \leq H) = \Pr (T' \text{ finite}).
\]

For the proof, we need the theorem of monotone convergence from the appendix:

**Theorem A.4.** Let \( B_1, B_2, \ldots \) be an infinite sequence of events such that \( B_i \supseteq B_{i+1} \) for all \( i \geq 1 \). Then we have

\[
\lim_{n \to \infty} \Pr (B_n) = \Pr \left( \bigcap_{i=1}^{\infty} B_i \right).
\]
Proof of Lemma 3.15. Let $B_i$ be the event that there exists a path from the root to some node at depth $i$. We first claim that

$$\bigcap_{i \geq 1} B_i = \{ T' \text{ infinite} \}.$$ 

Suppose that an outcome is contained in the left hand side. This means that $T'$ contains finite paths of arbitrary length. Of course, by consequence, $T'$ cannot be finite. The other direction is trivial.

From Theorem A.4, it now follows that

$$\lim_{i \to \infty} \Pr (B_i) = \Pr (T' \text{ infinite}).$$

Taking complements,

$$\lim_{H \to \infty} \Pr (h(T') \leq H) = \lim_{H \to \infty} \overline{B_H} = 1 - \lim_{H \to \infty} B_H = 1 - \Pr (T' \text{ infinite}),$$

the claim readily follows. \qed

The two last lemmas can now be combined to obtain a result on finite trees $T$.

Lemma 3.16. For each $p \in [0, 1]$, there exists a sequence $\epsilon_k^{(1)}(p), \epsilon_k^{(2)}(p), \ldots \in \mathbb{R}_0^+$, having $\epsilon_k^{(H)}(p) \to 0$ for $H \to \infty$ such that the following holds. Let $T$ be any finite (and not necessarily full) $k-1$-ary tree. Consider the following random experiment: each non-root node from $T$ is deleted (along with its subtree) independently from all other nodes with probability $p$. Then the probability that the resultant tree $T'$ has height at most $H \geq 1$ satisfies

$$\Pr (h(T') \leq H) \geq R_k(p) - \epsilon_k^{(H)}(p).$$

Please note an important subtlety in this statement: instead of using a limit in the resulting inequality, we have introduced the sequence of errors $\epsilon_k^{(H)}(p)$. This is crucial in order to stress that the rate with which the probability converges to the numbers given is independent of which tree $T$ we are considering ("uniform convergence"). On the other hand, this rate must depend on $p$, as anything stronger would not be provable from the statements we made above.

Proof. The random experiment we are conducting on $T$ can be coupled to a random experiment conducted on $T_\infty$ in the obvious way: $T$
is embeddable into $T_\infty$ with the two roots coinciding and then if we delete every node from $T_\infty$ independently with probability $p$, the same experiment is taking place on $T$. Note that for the tree $T''$ resulting from the deletions in $T_\infty$ and the tree $T'$ resulting from the deletions in $T$, since $T'$ is a subtree of $T''$ we have $h(T') \leq h(T'')$ and therefore $\Pr(h(T') \leq H) \geq \Pr(h(T'') \leq H)$. Define, for all $H \geq 1$,

$$
\epsilon_H(p) := \max\{R_k(p) - \Pr(h(T'') \leq H), 0\}.
$$

Then we find that

$$
\Pr(h(T') \leq H) \geq \Pr(h(T'') \leq H) \geq R_k(p) - \epsilon_k^{(H)}(p)
$$

and from the previous lemma,

$$
\lim_{H \to \infty} \epsilon_k^{(H)}(p) = 0,
$$

as required. \(\square\)

### 3.3.2 Of Independent and Dependent Labels

What we have obtained so far does not yet totally reflect what we are after: in the experiments studied so far, each node was deleted independently of all other nodes. This is not the case in the result we are eventually after, there dependencies can occur of the sort that nodes of the tree are linked to one another and if one of these linked vertices is deleted, all of them are deleted. The following lemma shows that this can only improve the probability to have a small remaining tree.

**Lemma 3.17.** Let $Z_1, Z_2, \ldots, Z_r \in \{0, 1\}$ be mutually independent binary random variables, each of which takes value one with probability $p$. Let $T$ be any finite (and not necessarily full) $k-1$-ary tree with a labeling $\sigma : V(T) \setminus \{\text{root}\} \to \{1, \ldots, r\}$ of the non-root nodes of $T$ with indices with the property that on each path from the root to a leaf, $\sigma$ is injective. Consider the experiment of drawing $Z_1, \ldots, Z_r$ according to their distribution and then deleting all nodes $u$ from $T$ (along with their subtrees) for which $Z_\sigma(u) = 1$. Call the resulting tree $T'$.

Juxtapose the experiment where in $T$, every non-root node is deleted independently from all other nodes with probability $p$. Call the random tree resulting from this experiment $T''$. Then for any $H$,

$$
\Pr(h(T') \leq H) \geq \Pr(h(T'') \leq H).
$$
To prove this theorem we need the FKG correlation inequality due to Fortuin, Kasteleyn and Ginibre. It basically states that monotone events are positively correlated. The proof is found in the appendix.

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_r\}$ be a collection of independent binary random variables. An event $E$ is said to be determined by $\mathcal{A}$, if there exists a fixed list $S_E \subseteq 2^\mathcal{A}$ such that $E = \{\{A \in \mathcal{A} | A = 1\} \in S_E\}$, or, informally speaking, if knowing the values of $\mathcal{A}$ leads to knowing whether $E$ occurs. Moreover, if $S_E$ is upwards hereditary, i.e. if

$$\forall \mathcal{A} \supseteq U \supseteq V : V \in S_E \Rightarrow U \in S_E,$$

then $E$ is called monotonically increasing in $\mathcal{A}$.

**Theorem A.7. (FKG Inequality)** Let $\mathcal{A} = \{A_1, A_2, \ldots, A_r\}$ be a collection of mutually independent binary random variables and $E_1$ and $E_2$ events which are determined by $\mathcal{A}$ and monotonically increasing in $\mathcal{A}$. Then

$$\Pr(E_1 \land E_2) \geq \Pr(E_1) \cdot \Pr(E_2).$$

Furthermore, for $l$ such events $E_1, E_2, \ldots, E_l$,

$$\Pr\left(\bigwedge_{i=1}^l E_i\right) \geq \prod_{i=1}^l \Pr(E_i).$$

**Proof of Lemma 3.17.** The statement is trivial if $\sigma$ is globally injective. We now use the FKG inequality to demonstrate that correlations arising from duplicate labels cannot decrease this probability if none of these duplicates are in an ancestor-descendant relation.

To achieve this, we proceed by induction on $H$. For $H = 0$ the statement is trivial.

For the induction step, let $H > 0$. If the root of $T$ has no child, the statement is trivial. Now suppose the root has $0 < l \leq k - 1$ children; call them $u_1, \ldots, u_l$. Let $i \in \{1, \ldots, l\}$ and let us look at each of the root’s subtrees separately first. Let $T_i$ be the subtree rooted at $u_i$ and let $Z_i = \sigma(u_i)$. Let $T_i'$ denote the subtree of $T'$ rooted at $u_i$ and $T_i''$ the subtree of $T''$ rooted at $u_i$ (empty trees if $u_i$ is deleted). The hypothesis on $\sigma$ entails that no other node in $T_i$ is labeled with $Z_i$, so whatever happens in the non-root nodes of $T_i$ is independent of whether $u_i$ itself is being deleted or not. Let

$$E_i := \{Z_i = 1 \lor (Z_i = 0 \land h(T_i') \leq H - 1)\}$$
We have that
\[ \{h(T') \leq H\} = \bigwedge_{i=1}^{l} E_i. \]
Moreover, the \( E_i \) are events which are determined by \( \{Z_1, \ldots, Z_r\} \) and, as one can easily check, monotonically increasing in these. Therefore we can use the FKG Inequality as in Theorem A.7 to obtain that
\[
\Pr (h(T') \leq H) = \Pr \left( \bigwedge_{i=1}^{l} E_i \right) \geq \prod_{i=1}^{l} \Pr (E_i).
\]
For these separate events, we have, due to the independence of \( T_i \) from \( Z_i \)
\[
\Pr (E_i) = p + (1-p) \Pr (h(T''_i) \leq H - 1) \geq p + (1-p) \Pr (h(T''_i) \leq H - 1)
\]
where we have used the induction hypothesis for the inequalities. Combining the last two inequalities, we obtain
\[
\Pr (h(T') \leq H) \geq \prod_{i=1}^{l} (p + (1-p) \Pr (h(T''_i) \leq H - 1))
= \Pr (h(T''') \leq H).
\]
Moreover, we will need the following simple observation for technical reasons:

**Observation 3.18.** *In the setting of Lemma 3.17, \( \Pr (h(T') \leq H) \) as a function of \( p \) is continuous and monotonically non-decreasing.*

### 3.3.3 Integrating over the Rank of the Root

Using the previous lemma we can now finally approach the proof of Theorem 3.11. Fix \( \epsilon > 0 \).

An arbitrary tree \( T \) is labeled with real-valued random variables as \( \sigma : V(T) \to \{X_1, X_2, \ldots, X_r\} \) in such a way that \( \sigma \) is injective on any path from the root to a leaf of \( T \). Now each \( X_i \) is drawn independently and uniformly from \([0, 1]\). Then all nodes are deleted (along with their subtrees) whose corresponding random variable is smaller than that of the root, producing the resultant random tree \( T' \).
Without loss of generality, suppose $\sigma(\text{root}) = X_r$. Note that this value is independent of all other values used because the root is part of all paths and $\sigma$ is injective on each path, so $X_r$ does not occur a second time as a label. Now once we condition on $X_r = \gamma$ for some fixed value $\gamma \in [0, 1]$, we can consider for $1 \leq i \leq r - 1$ the binary random variables

$$Z_i := \begin{cases} 1 & \text{if } X_i < \gamma \\ 0 & \text{otherwise.} \end{cases}$$

The $\{Z_1, Z_2, \ldots, Z_{r-1}\}$ are mutually independent binary random variables, each of which takes value 1 with probability exactly $\gamma$. This is the situation we have in Lemma 3.17 and so we can use this together with Lemma 3.16 result to conclude that

$$\Pr(h(T') \leq H \mid X_r = \gamma) \geq R_k(\gamma) - \epsilon_k^{(H)}(\gamma),$$

and so

$$\lim_{H \to \infty} \Pr(h(T') \leq H \mid X_r = \gamma) \geq R_k(\gamma). \quad (3.4)$$

To convert this conditional into an unconditional probability, we invoke the law of total probability, which, since $Z_r$ is uniformly distributed, reads

$$\Pr(h(T') \leq H) = \int_0^1 \Pr(h(T') \leq H \mid X_r = \gamma) d\gamma.$$ 

$\Pr(h(T') \leq H \mid X_r = \gamma)$ and $R_k(\gamma)$ are monotonically non-decreasing (Observation 3.18 and Lemma 3.14), so by Lemma A.6 we can go from pointwise convergence to convergence of the integral and obtain

$$\lim_{H \to \infty} \Pr(h(T') \leq H) = \lim_{H \to \infty} \int_0^1 \Pr(h(T') \leq H \mid X_r = \gamma) d\gamma \geq \int_0^1 R_k(\gamma) d\gamma.$$

Using basic integration rules (see appendix for a proof), we obtain

**Lemma 3.19.**

$$\int_0^1 R_k(x) dx = 1 - S_k,$$

$$\int_0^1 R_3(x) dx = 2 - 2 \ln 2.$$
3.3. Random Deletion in $k - 1$-ary Trees

Note that a tree with more than $D$ nodes needs to have height more than $\log_{k-1}(D) - 1$, and thus

$$\Pr(|V(T')| > D) \leq \Pr(h(T') > \log_{k-1}(D) - 1),$$

so

$$\lim_{D \to \infty} \Pr(|V(T')| > D) \leq \lim_{D \to \infty} \Pr(h(T') > \log_{k-1}(D) - 1)$$

$$= \lim_{H \to \infty} \Pr(h(T') > H)$$

$$= 1 - \lim_{H \to \infty} \Pr(h(T') \leq H)$$

$$\leq 1 - \int_{0}^{1} R_k(x) \, dx$$

$$\leq S_k.$$

Choosing $D_k(\epsilon)$ appropriately concludes the proof of of Theorem 3.11. For $k = 3$, we obtain $S_3 = 2 \ln 2 - 1$.

Note that its possible to omit the usage of Lemma A.6 by exchanging the integral with the convergence of $H$; however, this would require us to carry the integral through the whole proof and furthermore it does not work in Chapter 5. With our approach the integral is contained to its own subsection.

The difference to [24] is how we treat the convergence for increasing $D$. In [24], this was done analytically by estimating a very complicated integral. Here we used a measure theoretic approach by using the monotone convergence theorem. This makes things much simpler and in the non-Boolean case treated in Chapter 5 it appears to be the only feasible way. The disadvantage of our approach is that we do not know the exact convergence rates, however, this would only affect subexponential factors (which from our theoretical standpoint are considered negligible).

A purely combinatorial analysis of critical clause trees can be found in [38].
Chapter 4

PPSZ for Multiple Satisfying Assignments

In the previous chapter we dealt with the case where $F$ has a unique satisfying assignment. In general, we cannot rely on such a strong assumption. This makes the whole analysis considerably more complicated and for roughly ten years, it was open whether it could at all be carried out without a loss in terms of an inferior running time.

In fact, the original analysis by Paturi et al. [24] attempts at circumventing the problem of multiple satisfying assignments by partitioning the solution space into disjoint subregions within which the formula may be considered ‘uniquely satisfiable’ and then averaging success probabilities over these regions. This results in an algorithm running in time $O(1.364^n)$ for general 3-SAT (compared to $O(1.308^n)$ for Unique 3-SAT). The analysis is also very complicated; in the published journal version [24] a simplified proof is used that giving a running time of $O(1.435^n)$.\footnote{The proof of the original bound can be found at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1134, see also the author’s master thesis [6]}

Such a gap also exists for $k = 4$, although smaller ($O(1.469^n)$ vs. $O(1.479^n)$). For $k \geq 5$, the bounds match. It was conjectured in [24] that this gap can be closed.

Subsequent work on moderately exponential time algorithms for 3-SAT can be seen as an attempt to close this gap, however by using a different algorithm. Schöning’s random walk algorithm [33], running in time $O(1.334^n)$ for 3-SAT was published just one year after PPSZ. After some improvements of Schöning’s algorithm to
Iwama and Tamaki [17] found out how to combine both PPSZ and Schöning to obtain running time $O(1.328^n)$ for general 3-SAT. Subsequent improvements managed to lower the running time to $O(1.321^n)$ [9, 16, 30], at the cost of increasing the complexity of the algorithm considerably.

Our contribution is to close the gap by showing that PPSZ itself has the same running time for $k$-SAT as for Unique $k$-SAT (or more accurate, the probability to find some satisfying assignment is at least as good as to find a unique satisfying assignment). This improves the best known running time for 3-SAT and 4-SAT. Our analysis replaces the analysis of [24] for general $k$-SAT and it is simpler compared to original analysis of [24].

Remember the definition $S_k := \int_0^1 \frac{t^{1/(k-1)} - t}{1-t} dt$ from the previous chapter. In this chapter we extend the results of the previous chapter to general $k$-SAT:

**Theorem 4.1.** If $F$ is an $(\leq k)$-CNF over $n$ variables $V$ then

$$\Pr(\text{PPSZ}(F, V, \log n) \in \text{sat}_V(F)) \geq 2^{-\left(S_k + o(1)\right)n}.$$ 

Furthermore, PPSZ$(F, V, \log n)$ runs in time $2^{o(n)}$ (i.e. in subexponential time).

**Corollary 4.2.** PPSZ can be turned into a randomized algorithm for $k$-SAT running in time $2^{\left(S_k + o(1)\right)n}$.

As mentioned, these bounds were known for $k \geq 5$ in [24]. The original proof for $k = 3, 4$ has appeared in [7]. This write-up of this chapter is based on [10], joint work with Robin Moser and Dominik Scheder, containing many didactic improvements, and integrating a result by Sebastian Millius [21] showing how to deal with forced variables (in [7], PPSZ had to be modified slightly). Additionally, in Section 4.6 we mention a result from the master thesis of Isabelle Hurbain [13], giving a lower bound on the probability that an individual satisfying assignment is returned.

### 4.1 Problem Assessment

Where does the analysis go wrong when we have multiple satisfying assignments?

Recall the tree construction process on page 16. Suppose we are assuming $\alpha$ to still be $a$ (not anymore the) satisfying assignment and
suppose we are trying to build a critical clause tree $T_x$ for $x$. If the current intermediate tree $T$ has a leaf $u$ that does not yet have a clause label, it defines $U$ to be the set of all variable labels of ancestors of $u$. Then it defines an assignment $\beta$ via

$$\beta : \text{vbl}(F) \to \{0, 1\}, \quad z \mapsto \begin{cases} 1 - \alpha(z) & \text{if } z \in W, \\ \alpha(z) & \text{otherwise}, \end{cases}$$

chooses a clause $C \in F$ that is not satisfied by $\beta$ and sets $C$ as clause label of $u$. The problem is that if $\alpha$ is just some but not the unique satisfying assignment, $\beta$ might as well satisfy $F$, and thus such a clause $C$ need not exist and we are stuck.

Still, most of the arguments from the first section go through with a milder assumption than uniqueness of the satisfying assignment. Note that in the present case $\beta(x) \neq \alpha(x)$ according to definition, because $x \in W$ holds always in this tree. So in order to construct a full tree without getting stuck, we only require that there are no satisfying assignments $\beta$ with $\beta(x) \neq \alpha(x)$, i.e. that all satisfying assignments assign the same value to $x$.

In the remainder of the chapter we let $D$ be some number denoting how many clauses PPSZ is allowed to look at. This choice is arbitrary; for the final result we will again use $D = \log n$.

**Definition 4.3.** Let $F$ be any satisfiable $(\leq k)$-CNF formula over $V$ and $x \in V$. $x$ is said to be frozen in $F$ if either $F \models x$ or $F \models \bar{x}$, that is all satisfying assignments of $F$ send $x$ to the same value. Otherwise $x$ is said to be non-frozen.

Note that variables that do not appear in $F$ (i.e. $x \in V \setminus \text{vbl}(F)$) are always non-frozen; such variables might seem unnecessary but we consider them due to technical reasons.

For any satisfiable formula $F$ over $V$, this partitions the variables into three categories

$$V_{fo}(F, V) \cup V_{fr}(F, V) \cup V_{nf}(F, V) = V,$$

where

- $V_{fo}(F, V) = \{x \in V \mid F \models_D x\}$ are the variables that are currently forced, that is which are frozen and follow from $F$ via $D$-implication,
• $V_{fr}(F, V) := \{x \in V \mid F \models x, F \not\models_D x\}$ are the variables that are frozen (but not forced) and

• $V_{nf}(F, V) := \{x \in V \mid F \not\models x\}$ are the remaining variables that are non-frozen, i.e. those which you can assign either way keeping the formula satisfiable.

The arguments of the last section show that at least if a variable is frozen, the probability that it needs to be guessed during a run of PPSZ is bounded.

**Lemma 4.4.** Let $F$ be an $(\leq k)$-CNF formula over $V$ and $x \in V_{fo}(F, V) \cup V_{fr}(F, V)$ be a variable that is frozen. Let furthermore $\alpha$ be any satisfying assignment. Then $\Pr (x \in \text{Guessed}(F, V, \alpha, \pi, D)) \leq S_k^{(D)}$, with $S_k^{(D)}$ defined as in Theorem 3.11.

This lemma is actually the only statement we require here from the previous chapter. The proof is completely analogous to the previous chapter. After having constructed the critical clause trees, the analysis of the trees is exactly the same; giving the exact same bound $S_k^{(D)}$ on the probability that a variable is guessed (w.r.t. $\alpha$).

One could be inclined to think that we are already done now. The non-frozen variables are totally harmless: we can assign them either way and stay satisfiable. And since the previous lemma, we now know that whenever a variable becomes frozen during execution, the probability that it needs to be guessed in the further execution of the algorithm is bounded by the same number it was bounded in the case of uniquely satisfiable formulas. Or not?

The big problem arising is that we were able to bound this probability only for a fixed satisfying assignment $\alpha$. So as long as the algorithm chooses to assign values according to $\alpha$, the probability that a frozen variable needs to be guessed is appropriately bounded. But there could be many satisfying assignments around and which of them the algorithm is going to steer towards is decided only while it executes and assigns values to the non-frozen variables.

In particular, the probability with which each of the assignments is being selected can depend heavily on the ordering in which we process variables. And although each satisfying assignment $\alpha$ has, according to the lemma, the property that frozen variables need to be guessed with probability no higher than $S_k^{(D)}$ for a uniformly random ordering $\pi$, the probability of $\alpha$ at all becoming relevant for us depends heavily on
4.2 Definition of a Cost Function

To be able to formalize the above tradeoff we need to look “inside” PPSZ, as the number of non-frozen variables might change with every new variable assigned (a newly assigned variable might rule out some previously compatible satisfying assignments, making some variables frozen). An “intermediate state” of the algorithm can be represented by a partial assignment $\alpha_0$ over $V$ containing all the values we have already selected in previous steps. We will now define two things: A “cost”, measuring how hard it is to find a satisfying assignment when we already know $\alpha_0$, and a “likelihood”, measuring how likely a given satisfying assignment $\alpha$ is produced (among all satisfying assignments) when starting with $\alpha_0$. The purpose of this definition is measure what happens inside PPSZ, giving us the flexibility to accomodate non-frozen variables using the mentioned tradeoff.

For the remainder of this chapter, let $F$ be a fixed satisfiable ($\leq k$)-CNF formula over $V$. To do the calculation, we will associate a cost to each state of the algorithm as follows. For notational convenience we write

- $V(\alpha_0)$ as a shorthand for $V \setminus \text{dom}(\alpha_0)$,
- $n(\alpha_0) := |V(\alpha_0)|$,
- $V_{fo}(\alpha_0) := V_{fo}(F^{[\alpha_0]}, V(\alpha_0))$,
• $V_{fr}(\alpha_0) := V_{fr}(F^{[\alpha_0]}, V(\alpha_0))$,
• $V_{nf}(\alpha_0) := V_{nf}(F^{[\alpha_0]}, V(\alpha_0))$.

We start defining the cost for a fixed variable $x$ and satisfying assignment $\alpha$ to be a number between 0 and 1. Intuitively this denotes the probability that something bad happens with variable $x$ when building $\alpha$ (where something bad means that we have to guess $x$ while it is frozen). To make notation easier, we will allow $x$ and $\alpha$ to be arbitrary, and define the corresponding cost to be 0 in degenerate cases.

**Definition 4.5.** Let $\alpha_0$ be a partial and $\alpha$ be a total assignment on $V$ and let $x \in V$ be any variable. We define the cost of $x$ when completing $\alpha_0$ to $\alpha$, in writing $\text{cost}(\alpha_0,\alpha,x)$, as follows.

• If $x \not\in V(\alpha_0)$, then $\text{cost}(\alpha_0,\alpha,x) := 0$.

• If $\alpha_0$ and $\alpha$ are incompatible, i.e. $\exists x : \{\alpha_0(x), \alpha(x)\} = \{0,1\}$, then $\text{cost}(\alpha_0,\alpha,x) := 0$.

• If $\alpha$ does not satisfy $F$, then $\text{cost}(\alpha_0,\alpha,x) := 0$.

• Otherwise, if
  - $x \in V_{fo}(\alpha_0)$, then $\text{cost}(\alpha_0,\alpha,x) := 0$.
  - $x \in V_{fr}(\alpha_0)$, then
    \[ \text{cost}(\alpha_0,\alpha,x) := \Pr_{\pi}\left(x \in \text{Guessed}(F^{[\alpha_0]}, V(\alpha_0), \alpha, \pi, D)\right), \]
    where $\pi$ is chosen u.a.r. from all permutations on $V(\alpha_0)$.
  - $x \in V_{nf}(\alpha_0)$, then $\text{cost}(\alpha_0,\alpha,x) := S_k^{(D)}$.

For frozen variables, the cost is exactly the probability that $x$ is guessed w.r.t. $\alpha$, when starting with $F^{[\alpha_0]}$. For a non-frozen variable $x$, we cannot say anything about the probability that it is guessed. Since $x$ might become frozen at some point and we want our cost to be non-increasing over time, we set the cost to $S_k^{(D)}$, the upper bound of the guessing probability of Theorem 3.11 and Lemma 4.4.

Next, we define the likelihood of each assignment. The correct distribution over the satisfying assignments matches (in the Boolean case) the distribution PPSZ would give were it allowed to look at the whole formula instead of just all small subformulas. This motivates the following definition:
Definition 4.6. Let $\alpha_0$ be a partial and $\alpha$ a total assignment over $V$. The likelihood of completing $\alpha_0$ to $\alpha$, $\text{lkhd}(\alpha_0,\alpha)$, is defined as the probability that the assignment returned by PPSZ($F^{[\alpha_0]}$, $V(\alpha_0), |F^{[\alpha_0]}|$) (i.e. for $D = |F^{[\alpha_0]}|$), combined with $\alpha_0$, is $\alpha$.

Note that the version of PPSZ used in this definition is very inefficient and only used for the analysis. Since $D$ is as large as the whole formula, every implication is a $D$-implication and thus whenever a variable is frozen, it is automatically forced. The likelihood of $\alpha$ is thus in a certain sense the probability that PPSZ selects $\alpha$ during execution, under the assumption that it never makes mistakes and that there is no gap between frozen and forced variables. For this reason, if $\alpha_0$ is incompatible with $\alpha$ or if $\alpha$ does not satisfy $F$, then $\text{lkhd}(\alpha_0,\alpha) = 0$ (unless $F^{[\alpha_0]}$ is not satisfiable to begin with). Thus if $F^{[\alpha_0]}$ is satisfiable, $\text{lkhd}(\alpha_0,\alpha)$ defines a probability distribution over the satisfying assignments of $F$. Also observe that this probability distribution does only depend on the set of satisfying assignments of $F^{[\alpha_0]}$, not on the formula itself. It is important to note that the likelihood distribution is not the uniform distribution. Consider e.g. $F = \{\{x,y\}\}$ and observe that $\text{lkhd}(\emptyset,\{x \mapsto 1, y \mapsto 1\}) = \frac{1}{4}$, but $\text{lkhd}(\emptyset,\{x \mapsto 0, y \mapsto 1\}) = \frac{3}{8}$. This means that some satisfying assignments are more likely to be returned by PPSZ in some sense and thus are also more important. In Section 4.6 we see that this is indeed also the case in a normal execution of PPSZ.

To now define the cost of completing $\alpha_0$ to any satisfying assignment (for all variables and assignments combined), we sum over all possible assignments weighted by the likelihood and then over all variables:

$$
\text{cost}(\alpha_0, x) := \sum_{\alpha \in \{0,1\}^V} \text{lkhd}(\alpha_0,\alpha) \cdot \text{cost}(\alpha_0,\alpha,x) \quad \text{wcost}(\alpha_0,\alpha,x)
$$

and

$$
\text{cost}(\alpha_0) := \sum_{x \in V} \text{cost}(\alpha_0, x).
$$

4.3 Basic Facts

Next gather some basic facts about the cost and the likelihood. To begin with, consider the following.

Observation 4.7. For any $\alpha_0$, $\alpha$, $x$, we have $\text{cost}(\alpha_0,\alpha,x) \leq S_k^{(D)}$. Furthermore $\text{cost}(\alpha_0) \leq S_k^{(D)} n(\alpha_0)$. 

Proof. The statement follows directly from the definition of the cost and the probability bound of Lemma 4.4.

Lemma 4.8. Let $\alpha_0$ be a partial assignment, and $\alpha$ be an assignment on $V$. For any fixed variable $x \in V(\alpha_0)$, if we set $x$ according to $\alpha$,

(i) the likelihood of $\alpha$ can only increase, i.e.
$$lkhd(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \geq lkhd(\alpha_0, \alpha)$$
with equality if $x$ is frozen in $F^{[\alpha_0]}$, and

(ii) the cost of a fixed variable $y \in V$ w.r.t. $\alpha$ can only decrease, i.e.
$$cost(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y) \leq cost(\alpha_0, \alpha, y).$$

On the other hand, when choosing $x \in V(\alpha_0)$ uniformly at random (assume $V(\alpha_0)$ is nonempty) and setting it according to $\alpha$,

(iii) the likelihood of $\alpha$ increases on average as
$$E[lkhd(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)] = \left(1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}\right) lkhd(\alpha_0, \alpha),$$
whereas

(iv) the cost of a fixed variable $y \in V(\alpha_0)$ decreases on expectation as
$$E[cost(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)] \leq cost(\alpha_0, \alpha, y) - \frac{s}{n(\alpha_0)},$$
where
$$s = \begin{cases} 
1 & \text{if } y \in V_{fr}(\alpha_0) \\
S_k^{(D)} & \text{if } y \in V_{nf}(\alpha_0) \\
0 & \text{if } y \in V_{fo}(\alpha_0). 
\end{cases}$$

Proof. (i) Assume for the moment that the permutation $\pi$ is fixed. What is $lkhd(\alpha_0, \alpha)$? To output $\alpha$, every non-frozen variable that PPSZ encounters has to be set to the right value, which happens with probability $1/2$. If $D = |F|$, all frozen variables are automatically forced. Hence, for a fixed $\pi$, $lkhd(\alpha_0, \alpha)$ is $2^{-n_{f}(\pi, \alpha_0, \alpha)}$, where $n_{f}(\pi, \alpha_0, \alpha)$ denotes the number of non-frozen variables encountered. Therefore we have
$$lkhd(\alpha_0, \alpha) = E_{\pi} \left[ 2^{-n_{f}(\pi, \alpha_0, \alpha)} \right]$$
Moving \( x \) to the beginning in \( \pi \) can only decrease the number of non-frozen variables. Furthermore, if \( x \) is frozen in \( F^{[\alpha_0]} \), the number of non-frozen variables remains the same.

Now observe that if we remove \( x \) from \( \pi \), the resulting permutation \( \pi' \) has a uniform distribution from permutations over \( V(\alpha_0) \setminus \{x\} \). The number of non-frozen variables can only decrease in \( V(\alpha_0) \setminus \{x\} \); removal of \( x \) might decrease this even further. Therefore

\[
E_\pi \left[ 2^{-\text{nf}(\pi,\alpha_0,\alpha)} \right] \leq E_{\pi'} \left[ 2^{-\text{nf}(\pi',\alpha_0 \cup \{x \mapsto \alpha(x)\},\alpha)} \right],
\]

with equality if \( x \) is frozen, as in this case \( x \) is always assigned \( \alpha(x) \). The latter term is equal to \( \text{lkh}(\alpha_0 \cup \{x \mapsto \alpha(x)\},\alpha) \), and we are done.

(ii) Whenever \( \text{cost}(\alpha_0,\alpha,x) = 0 \) because we are in a degenerate case (the first three cases) of Definition 4.5, it is easy to see \( \text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\},\alpha,y) = 0 \) because the same degenerate case applies again.

Otherwise, if \( x \) is non-frozen, then \( \text{cost}(\alpha_0,\alpha,y) = S_k^{(D)} \) and by Observation 4.7, \( \text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\},\alpha,y) \) is at most \( S_k^{(D)} \).

If \( y \in V_{nf}(\alpha_0) \), then \( \text{cost}(\alpha_0,\alpha,y) = S_k^{(D)} \). By Observation 4.7 the cost for any variable and assignment is at most \( S_k^{(D)} \), so \( \text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\},\alpha,y) \leq S_k^{(D)} \).

If \( y \in V_{fr}(\alpha_0) \) or \( y \in V_{fo}(\alpha_0) \), then \( \text{cost}(\alpha_0,\alpha,y) \) is the probability that \( y \) is guessed in the remainder of PPSZ. If we now fix another variable \( x \) to \( \alpha(x) \), it is easily seen that this probability cannot decrease, so \( \text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\},\alpha,y) \leq \text{cost}(\alpha_0,\alpha,y) \).

(iii) First we give some intuitive reasoning. Observe that variables that are frozen now in \( F^{[\alpha_0]} \) can be ignored in the remainder of the likelihood definition: Their value is the same in all satisfying assignments, and assigning them will not change the set of satisfying assignments (up to the removal of the frozen variable).

We conveniently used PPSZ to define the likelihood in Definition 4.6, and the order in which the frozen variables are set corresponds to how PPSZ would do it, by choosing the next variable uniformly. The disadvantage of this approach is that were we to condition that the next variable is indeed assigned according to
\( \alpha \), as in the statement we are proving, then the choice of the next variable is not uniform anymore. Namely non-frozen variables are picked next with half the probability that frozen variables are.

To change that, we could have defined the likelihood equivalently by picking non-frozen variables with weight 2 and frozen variables with weight 1, resulting in a uniform choice after conditioning that the next variable is set according to \( \alpha \). The probability that the next variable is indeed set according to \( \alpha \) is

\[
\frac{|V_{nf}(\alpha_0)| + |V_{fo}(\alpha_0)|}{2|V_{nf}(\alpha_0)| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)|} = \frac{n(\alpha_0)}{|V_{nf}(\alpha_0)| + n(\alpha_0)},
\]

and the claimed statement is not hard to see.

In the following we give a more formal proof: We have

\[\text{lkhd}(\alpha_0, \alpha) = E[\text{PPSZ}(F^{[\alpha_0]}, V(\alpha_0), |F^{[\alpha_0]}|) \cup \alpha_0 = \alpha].\]

Let \( x \) be the variable selected next in PPSZ (\( x \) is obviously u.a.r. in \( V(\alpha_0) \)). If \( x \) is frozen, then PPSZ always sets \( x \) to \( \alpha(x) \). If \( x \) is non-frozen, then PPSZ sets \( x \) to \( \alpha(x) \) with probability \( 1/2 \). \( x \) is non-frozen with probability \( |V_{nf}(\alpha_0)|/n(\alpha_0) \). Therefore

\[\text{lkhd}(\alpha_0, \alpha) = \Pr (x \in V_{nf}(\alpha_0)) \cdot \frac{1}{2} \cdot E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \}) | x \in V_{nf}(\alpha_0)]
+ \Pr (x \notin V_{nf}(\alpha_0)) E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \}) | x \notin V_{nf}(\alpha_0)]
= \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \cdot \frac{1}{2} \cdot E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \}) | x \in V_{nf}(\alpha_0)]
+ \left( 1 - \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \right) \cdot E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \}) | x \notin V_{nf}(\alpha_0)]
= \frac{1}{2} \cdot E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \})]
+ \frac{1}{2} \left( 1 - \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \right) \cdot E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \}) | x \notin V_{nf}(\alpha_0)].\]

From (i), if \( x \notin V_{nf}(\alpha_0) \), then \( \text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \}) = \text{lkhd}(\alpha_0) \), and so we have

\[\text{lkhd}(\alpha_0, \alpha) = \frac{1}{2} \cdot E[\text{lkhd}(\alpha_0 \cup \{ x \mapsto \alpha(x) \})] + \frac{1}{2} \left( 1 - \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \right) \cdot \text{lkhd}(\alpha_0, \alpha).\]
Multiplying the equation by 2 and collecting the lkhd($\alpha_0, \alpha$)-terms on the left-hand side yields the statement.

(iv) If $y \in V_{fo}(\alpha_0)$, this follows from (ii). For the case $y \in V_{nf}(\alpha_0)$ observe that $x = y$ happens with probability $1/n(\alpha_0)$ and in that event the cost of $y$ reduces to 0. If $y$ is not selected, then by (ii) the cost does not increase. As $\text{cost}(\alpha_0, \alpha, y) = S_k^{(D)}$, the average decrease is hence at least $S_k^{(D)}/n(\alpha_0)$.

The interesting case is if $y \in V_{fr}(\alpha_0)$, where $x$ is frozen but not forced. In that case, the cost of $y$ is the probability that $y$ is guessed, and the statement tells us that this probability is reduced by $1/n(\alpha_0)$ after one step. This is because with probability $1/n(\alpha_0)$, $y$ comes next in $\pi$ and is guessed now (as $x$ is not forced now). This $1/n(\alpha_0)$ is counted in $\text{cost}(\alpha_0, \alpha, y)$, but not in $E[\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)]$, which is the probability that $y$ is guessed after the next step.

\[ \square \]

4.4 Bounding Expectations of Products

We have seen how the cost and the likelihood change during the execution of PPSZ. However, the total cost is the sum of the weighted cost, that is, the product of the likelihood with the cost. The expectation of a product of random variables depends on the correlation between these variables; this correlation corresponds exactly to the aforementioned problem that which assignment PPSZ steers to depends on the ordering of the variables, which in turn determines which variables are forced. The facts from the previous section allow us to bound the above expectation and thus the influence of the correlation.

For forced and non-frozen variables the cost does not depend on the satisfying assignment and thus matters are simple here. Forced variables have cost zero, and the cost will remain zero at all times, independent of the satisfying assignment.

A similar fact is true for the non-frozen variables. We have bounded the expected drop in cost for a non-frozen variable in Lemma 4.8 as $S_k^{(D)}/n(\alpha_0)$, and this drop stems exclusively from the case when the variable in question is being assigned and eliminated, causing its cost to drop to zero. If this happens, it happens in all assignments at once and so we do not care if at the same time there are shifts in the likelihood.
distribution. Note that the non-frozen variables might become frozen after one step but for our estimate we only used that the cost is at most \(S_k^{(D)}\), which is true for all assignments and all variables.

The difficult case arises for the variables which are frozen but not currently forced, \(y \in V_{fr}(\alpha_0)\). There, each assignment \(\alpha\) contributes a potentially different cost for \(y\) and so if at the same time the likelihood distribution over the various satisfying assignments changes, there can be potentially harmful correlations. The following simple correlation inequality will come handy in bounding these possibly negative effects.

**Lemma 4.9.** Let \(A, B \in \mathbb{R}\) be random variables and \(a, b, \bar{a}, \bar{b} \in \mathbb{R}\) fixed numbers such that \(A \geq a\) and \(B \leq b\) always and \(E[A] = \bar{a}\) and \(E[B] = \bar{b}\). Then
\[
E[A \cdot B] \leq a\bar{b} + b\bar{a} - ab.
\]

**Proof.** We can write
\[
E[A \cdot B] = E[(A - a) \cdot B] + aE[B]
\]
and then use \(B \leq b\) and \(A \geq a\) to obtain
\[
E[A \cdot B] \leq bE[A - a] + aE[B] = b\bar{a} - ba + \bar{b},
\]
as claimed. \qed

Applying this result, we can prove the following extension of Lemma 4.8 for the frozen variables.

**Lemma 4.10.** Let \(\alpha\) be a fixed satisfying assignment and \(\alpha_0 \subseteq \alpha\). Let \(y \in V_{fr}(\alpha_0)\) be a fixed frozen, non-forced variable. If we select \(x \in V(\alpha_0)\) uniformly at random and assign it according to \(\alpha\), then
\[
E[wcost(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)] \leq
\]
\[
\leq \left(1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}\right)wcost(\alpha_0, \alpha, y) - \frac{lkhd(\alpha_0, \alpha)}{n(\alpha_0)}.
\]

Note that the weighted cost might actually increase after one step, which might seem counterintuitive. However, the above expectation considers only setting \(x\) according to \(\alpha\). In the next section we also consider setting \(x\) differently (in which case the cost becomes 0) and see how this balances the increase of the weighted cost observed here.
4.5. Trading Survival Probability for Cost Savings

Proof. To apply Lemma 4.9 to the random experiment indicated, we set $A := \text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, y)$, $a := \text{lkhd}(\alpha_0, \alpha)$ and $\bar{a} := a(1 + |V_{nf}(\alpha_0)|/n(\alpha_0))$. We have $A \geq a$ and $E[A] = \bar{a}$ from Lemma 4.8. Further, we set $B := \text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, y, \alpha, y)$, $b := \text{cost}(\alpha_0, \alpha, y)$ and $\bar{b} := \text{cost}(\alpha_0, \alpha, y) - 1/n$. Again, we have $B \leq b$ and $E[B] = \bar{b}$ from Lemma 4.8. Invoking Lemma 4.9, we deduce that

$$E[\text{wcost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)] \leq ab + b\bar{a} - ab,$$

which matches the claim. \qed

4.5 Trading Survival Probability for Cost Savings

Suppose $\alpha_0$ is the current state of the algorithm. In a single following step, PPSZ selects a variable $x \in V(\alpha_0)$ uniformly at random. If $x$ is $D$-implied, it is being set accordingly, if not, it is assigned a value uniformly at random. Call $b \in \{0, 1\}$ the random variable representing the value PPSZ chooses to assign to $x$. Two things happen during such a step.

Firstly, there is a certain survival probability with which PPSZ will make an assignment that preserves satisfiability such that $\alpha_0 \cup \{x \mapsto b\}$ can still be completed to some satisfying assignment and we stay in the game. Let us write

$$S(\alpha_0) := \{(x', b') \in V(\alpha_0) \times \{0, 1\} \mid F[\alpha_0 \cup \{x' \mapsto b'\}] \text{ is satisfiable}\}$$

for the set of all variable-value pairs $(x', b')$ such that $x' \mapsto b'$ preserves satisfiability. If $\alpha_0$ is clear from the context, we write $S$ for $S(\alpha_0)$. Then the survival probability of this step of the algorithm is the probability that $(x, b) \in S$. This survival probability is easy to compute: whenever $x$ is a forced variable, we choose the correct value by $D$-implication and so the probability is one, whenever $x$ is a non-frozen variable, both values are okay and we survive as well with probability one. Only if $x$ is a frozen but not yet forced variable, we have a probability of $\frac{1}{2}$ to make a mistake.

Secondly, there is a drop in the costs: an additional variable $x$ gets a value which, for one, removes the cost of $x$ itself, and may additionally help to force other variables, such that the costs of frozen variables go down. How large are the new costs on average?
In principle, we are interested in the average drop in costs given that the current step makes the algorithm survive. However, since the survival probability as outlined above is different for different types of variables, entering such a conditional space would introduce a bias in the choice of \( x \) and \( b \) which is difficult to handle and would complicate matters. Instead, we here suggest to analyse the uniform distribution over \( S \). This distribution neither ideally reflects what the algorithm is actually doing, nor is the processed variable selected uniformly such that our cost decrease estimates from the last subsection could be applied seamlessly. But it is a handy intermediate distribution that can be related sufficiently easily to both aspects. We now claim the following.

**Lemma 4.11.** If \((X, B) \in S\) is selected uniformly at random, then

\[
\mathbb{E}[\text{cost}(\alpha_0 \cup \{X \mapsto B\})] \leq \text{cost}(\alpha_0) - \frac{|V_{fr}(\alpha_0)|}{|S|} - \frac{2S_k^{(D)} \cdot |V_{nf}(\alpha_0)|}{|S|}.
\]

**Proof.** We analyze the cost decrease contribution of the different types of variables separately. Each variable already assigned by \( \alpha_0 \) contributes zero cost both before and after the step. The same holds for currently forced variables.

For a non-frozen variable \( y \in V_{nf}(\alpha_0) \), note that \( S \) features two pairs containing \( y \), in contrast to the other types of variables of which \( S \) features only one pair each. This means that with probability \( 2|V_{nf}(\alpha)|/|S| \), a pair featuring \( y \) is selected. In that case the cost contribution of \( y \) drops from \( S_k^{(D)} \) to zero in all assignments. No matter what happens to these costs otherwise (they can certainly not increase by definition), the non-frozen variables hence contribute the last term in the claimed inequality.

Now consider the frozen, non-forced variables \( V_{fr}(\alpha_0) \). If we fix some satisfying and \( \alpha_0 \)-compatible assignment \( \alpha \in \{0, 1\}^V \) and condition the experiment on the event that \( \alpha(X) = B \), then \( X \) becomes uniformly at random among \( V(\alpha_0) \) because \( S \) features exactly one pair \((x', b')\) per variable \( x' \in V(\alpha_0) \) such that \( \alpha(x') = b' \). This is where the uniform choice over \( S \) comes very handy. As now, we can directly apply our previous Lemma 4.10 to find that conditioning on \( \alpha(X) = B \), the cost
of each frozen variable drops on average as
\[
E \left[ \text{wcost}(\alpha_0 \cup \{X \mapsto B\}, \alpha, y) \mid \alpha(X) = B \right] \leq \left( 1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \right) \text{wcost}(\alpha_0, \alpha, y) - \frac{\text{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)} \cdot \frac{|S|}{n}.
\]

The condition itself is satisfied with probability \( n/|S| \). If it does not apply, then the cost contribution of \( \alpha \) drops to zero altogether. Therefore, the unconditional change can be obtained by multiplying the right-hand side with \( n/|S| \) which then yields
\[
E \left[ \text{wcost}(\alpha_0 \cup \{X \mapsto B\}, \alpha, y) \right] \leq \text{wcost}(\alpha_0, \alpha, y) - \frac{\text{lkhd}(\alpha_0, \alpha)}{|S|},
\]
and the factor in front of \( \text{wcost}(\alpha_0, \alpha, y) \) conveniently cancels out.

If we, on both sides, take the sum over all assignments \( \alpha \in \{0, 1\}^V \) and all variables \( x \in V \), the claim follows.

Now we finally state the main lemma relating the cost with the success probability. This lemma immediately implies Theorem 4.1, as \( \text{cost}(\alpha_0) \leq S_k^{(D)} n(\alpha_0) \) by Observation 4.7.

**Lemma 4.12.** Let \( \alpha_0 \) s.t. \( F^{[\alpha_0]} \) is satisfiable. Then the overall probability of \( \text{PPSZ}(F^{[\alpha_0]}, V(\alpha_0), D) \) to output some satisfying assignment of \( F^{[\alpha_0]} \) is at least \( 2^{-\text{cost}(\alpha_0)} \).

**Proof.** We apply induction and suppose that the claim holds for all \( \alpha_0 \) that fix a larger number of variables. If \( \alpha_0 \) is total, then the statement holds trivially.

Let us denote by \( p(\alpha_0) \) the probability that \( \text{PPSZ}(F^{[\alpha_0]}, V(\alpha_0), D) \) outputs some satisfying assignment of \( F^{[\alpha_0]} \). Let \( x \) and \( b \) be random variables as defined before, i.e. \( x \in V(\alpha_0) \) u.a.r.; and \( b \in \{0, 1\} \) the forced value by \( D \)-implication if \( x \in V_{fo}(\alpha_0) \) and u.a.r. otherwise. Let
\[
\begin{align*}
w(x, b) := \begin{cases} 2, & x \in V_{fo}(\alpha_0) \\ 1, & \text{otherwise} \end{cases}
\end{align*}
\]
be the weight of \((x, b)\) in \( \text{PPSZ} \) (note that if and only if \( x \) is forced, \( b \) is not chosen u.a.r.). For possible \((x', b')\), we have \( \Pr(x = x' \land b = b') = \)
chapter 4

PPSZ for Multiple Satisfying Assignments

\[ \frac{w(x', b')}{|S| + |V_f(o(\alpha_0))|} \]

Observe that \((x, b)\) are chosen with exactly the distribution as PPSZ\((F[\alpha_0], V(\alpha_0), D)\) would choose the first variable \(x\) and its value \(\gamma(x)\). Furthermore,

\[ \text{PPSZ}(F[\alpha_0 \cup \{x \mapsto b\}], V(\alpha_0 \cup \{x \mapsto b\}), D) \cup \{x \mapsto b\} \]

has the same distribution as PPSZ\((F[\alpha_0], V(\alpha_0), D)\); in particular the probability to satisfy \(F[\alpha_0]\) is the same. Thus we have

\[ p(\alpha_0) = \mathbb{E}[p(\alpha_0 \cup \{x \mapsto b\})] \]

\[ = \Pr((x, b) \in S) \mathbb{E}[p(\alpha_0 \cup \{x \mapsto b\}) \mid (x, b) \in S] \]

\[ = \frac{|S| + |V_f(o(\alpha_0))|}{2n(\alpha_0)} \mathbb{E}[p(\alpha_0 \cup \{x \mapsto b\}) \mid (x, b) \in S]. \]

Now we change the expectation to \((X, B) \in S\) u.a.r. Apart from the forced variables, conditioning on \((x, b) \in S\) exactly gives uniform distribution; this also matches the distribution of Lemma 4.11.

\[ p(\alpha_0) = \frac{|S| + |V_f(o(\alpha_0))|}{2n(\alpha_0)}. \]

\[ \sum_{(X', B') \in S} \frac{w(X', B')}{|S| + |V_f(o(\alpha_0))|} p(\alpha_0 \cup \{X' \mapsto B'\}) \]

\[ = \frac{1}{2n(\alpha_0)} \sum_{(X', B') \in S} w(X', B') p(\alpha_0 \cup \{X' \mapsto B'\}) \]

\[ = \frac{|S|}{2n(\alpha_0)} \mathbb{E}[w(X, B)p(\alpha_0 \cup \{X \mapsto B\})]. \]

\[ \frac{|S|}{2n(\alpha_0)} \] corresponds to the probability that when choosing both \(x' \in V(\alpha_0)\) and \(b' \in \{0, 1\}\) uniformly at random, then \((x', b') \in S\). The weight \(w(X, B)\) comes from the fact that forced variables contribute twice as much to the success probability \(p(\alpha_0)\). It might seem problematic to carry the weight around in the expectation.

The original proof [7] circumvented this problem by automatically fixing variables the moment they became forced, and by now there wouldn’t be any forced variables anymore. However, Sebastian Millius [21] observed that Jensen’s inequality and linearity of expectation allows us to pull the product apart, and afterwards forced variables look just like other frozen variables. Interestingly, the exact weighting
w(X, B) does not matter; the only thing that is important is the expectation of the logarithm of w(X, B). Using Jensen’s inequality, we have

\[
p(\alpha_0) \geq 2^{\log\left(\frac{|S|}{2|n(\alpha_0)|}\right)} 2E[\log(w(X,B)p(\alpha_0\cup\{X\mapsto B\})] \\
= 2^{\log\left(\frac{|S|}{2|n(\alpha_0)|}\right)} 2E[\log(w(X,B))] - E[-\log(p(\alpha_0\cup\{X\mapsto B\})].
\]

Observe that

\[
E[\log(w(X,B))] = \Pr(w(X,B) = 2) = \frac{|V_{fo}(\alpha_0)|}{|S|},
\]

and by induction and Lemma 4.11

\[
E[-\log(p(\alpha_0 \cup \{X \mapsto B\})] \leq E[cost(\alpha_0 \cup \{X \mapsto B\})] \\
\leq cost(\alpha_0) - \frac{|V_{fr}(\alpha_0)|}{|S|} - \frac{2S_k^{(D)} \cdot |V_{nf}(\alpha_0)|}{|S|}.
\]

Putting things together, we obtain

\[
p(\alpha_0) \geq 2^{\log\left(\frac{|S|}{2|n(\alpha_0)|}\right)} + \frac{|V_{fo}(\alpha_0)|}{|S|} - cost(\alpha_0) + \frac{|V_{fr}(\alpha_0)|}{|S|} + 2S_k^{(D)} \cdot \frac{|V_{nf}(\alpha_0)|}{|S|}.
\]

As we want to show that \(p(\alpha_0) \geq 2^{-cost(\alpha_0)}\), we need to show

\[
\log\left(\frac{|S|}{2n(\alpha_0)}\right) + \frac{|V_{fo}(\alpha_0)|}{|S|} - cost(\alpha_0) + \\
\frac{|V_{fr}(\alpha_0)|}{|S|} + 2S_k^{(D)} \cdot \frac{|V_{nf}(\alpha_0)|}{|S|} \geq -cost(\alpha_0).
\]

\(cost(\alpha_0)\) conveniently cancels and we are left to show

\[
\log\left(\frac{|S|}{2n(\alpha_0)}\right) + \frac{|V_{fo}(\alpha_0)|}{|S|} + \frac{|V_{fr}(\alpha_0)|}{|S|} + 2S_k^{(D)} \cdot \frac{|V_{nf}(\alpha_0)|}{|S|} \geq 0.
\]

On one hand, we have the logarithmic summand; this stems from the “success probability” of one step (ignoring forced variables); it is
between \(-1\) and 0, the larger the more non-frozen variables there are. To balance this out we have a cost reduction of \(\frac{1}{|S|}\) for each frozen variable (forced variables have the same reduction from the weight function). However, for non-frozen variables, the cost reduction is only twice \(\frac{|S|}{|S|}\) (twice as non-frozen variables \(X\) have two pairs \((X, B) \in S\)). Hence the more non-frozen variables there are, the smaller the cost reduction is (this is not linear, as the number of non-frozen variables also affects the denominator \(|S|\)). We can estimate the logarithm using the following inequality from the appendix:

Lemma A.1. For \(x \geq 0\),

\[
\log(1 + x) \geq \log(e) \frac{x}{1 + x}.
\]

With this, we have

\[
\log \left( \frac{|S|}{2n(\alpha_0)} \right) = -1 + \log \left( \frac{|S|}{n(\alpha_0)} \right)
\]

\[
= -1 + \log \left( 1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \right)
\]

\[
\geq -1 + \log(e) \frac{|V_{nf}(\alpha_0)|}{1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}}
\]

\[
= -1 + \log(e) \frac{|V_{nf}(\alpha_0)|}{n + |V_{nf}(\alpha_0)|}
\]

\[
= -1 + \log(e) \frac{|V_{nf}(\alpha_0)|}{|S|}.
\]

This means that for every non-frozen variable, we are \(\frac{\log(e)}{|S|}\) above \(-1\). It remains to show

\[-1 + \log(e) \frac{|V_{nf}(\alpha_0)|}{|S|} + \frac{|V_{fo}(\alpha_0)|}{|S|} + \frac{|V_{fr}(\alpha_0)|}{|S|} + \frac{2S_k^{(D)} \cdot |V_{nf}(\alpha_0)|}{|S|} \geq 0.\]

Multiplying by \(|S|\) gives equivalently

\[-|S| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)| + (\log(e) + 2S_k^{(D)} \cdot |V_{nf}(\alpha_0)|) \geq 0.\]

As long as \(\log(e) + 2S_k^{(D)} \geq 2\), things work out nicely. However, as \(S_k^{(D)} \geq S_k \geq S_3 > 0.38\) (by definition \(S_k^{(D)} \geq S_k; S_k \geq S_3\) follows
from the definition and is also intuitively clear), so \( \log(e) + 2S_k^{(D)} > 1.44 + 2 \cdot 0.38 > 2 \), and the left-hand side of the inequality is at least 0 as \(|S| = 2|V_{nf}(\alpha_0)| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)|\).

Our analysis works as long as \( S_k^{(D)} \geq 1 - \frac{\log(e)}{2} \approx 0.2786 \), corresponding to a running time of \( O(1.214^n) \). A gap would reappear if a new analysis would be able to show that an even smaller \( S_k^{(D)} \) works for Unique \( k \)-SAT.

\[ \square \]

### 4.6 Return Probability of a Single Satisfying Assignment

In this section we discuss a result from Isabelle Hurbain’s master thesis [13], which was proposed as an open problem in [7]. What is the probability that PPSZ returns a fixed satisfying assignment \( \alpha \)? It turns out that this is just \( \text{lkhd}() \cdot 2^{-(S_k + o(1))n} \), i.e. the probability to return some satisfying assignment multiplied by the likelihood. This proof might give more insight to the PPSZ algorithm for general \( k \)-SAT and was developed in the context of finding a derandomization of PPSZ. We show the following extension to Theorem 4.1:

**Theorem 4.13 ([13]).** If \( F \) is an \((\leq k)\)-CNF over \( n \) variables \( V \) and \( \alpha \) a satisfying assignment on \( V \), then

\[
\Pr \left( \text{PPSZ}(F, V, \log n) \in \text{sat}_V(F) \right) \geq \text{lkhd}(\emptyset, \alpha) \cdot 2^{-(S_k + o(1))n}.
\]

We prove this using an extension to Lemma 4.12:

**Lemma 4.14 ([13]).** Let \( \alpha_0 \) s.t. \( F^{[\alpha_0]} \) be satisfiable, and \( \alpha \) be a satisfying assignment of \( F \) on \( V \). Then

\[
\Pr \left( \text{PPSZ}(F^{[\alpha_0]}, V(\alpha_0), D) \cup \alpha_0 = \alpha \right) \geq \text{lkhd}(\alpha_0, \alpha) \cdot 2^{-\text{cost}(\alpha_0, \alpha)}
\]

where

\[
\text{cost}(\alpha_0, \alpha) = \sum_{x \in V} \text{cost}(\alpha_0, \alpha, x).
\]

Theorem 4.13 is an easy consequence of this lemma. Furthermore, this lemma implies Lemma 4.12 by a straightforward application of Jensen’s inequality.
Proof. The first steps follow the proof of Lemma 4.12, with the difference that we are interested in the probability that a fixed satisfying assignment is returned.

We apply induction and suppose that the claim holds for all $\alpha_0$ that fix a larger number of variables. If $\alpha_0$ is total, then the statement holds trivially (note that if $\alpha_0$ and $\alpha$ are not compatible, then $\text{lkhd}(\alpha_0, \alpha) = 0$).

Let $p(\alpha_0, \alpha) := \Pr\left(\text{PPSZ}(F^{|\alpha_0|}, V(\alpha_0), D) \cup \alpha_0 = \alpha\right)$, the probability that the output of PPSZ on $F^{|\alpha_0|}$, together with $\alpha_0$, is $\alpha$.

Let $x$ and $b$ be random variables as in Lemma 4.12, i.e. $x \in V(\alpha_0)$ u.a.r.; and $b \in \{0, 1\}$ the forced value by $D$-implication if $x \in V_{\text{fo}}(\alpha_0)$ and u.a.r. otherwise. Let

$$w(x) := \begin{cases} 2, & x \in V_{\text{fo}}(\alpha_0) \\ 1, & \text{otherwise} \end{cases}$$

be the corresponding weight. First we change the expectation from $(x, b)$ to $X \in V(\alpha_0)$ u.a.r. Previously there was also a random value $B$ replacing $b$, but if we are to obtain a (fixed) $\alpha$, then $X$ must be set to $\alpha(X)$. We have

$$p(\alpha_0, \alpha) = \mathbb{E}[p(\alpha_0 \cup \{x \mapsto b\}, \alpha)]$$

$$= \Pr(b = \alpha(x)) \mathbb{E}[p(\alpha_0 \cup \{x \mapsto b\}, \alpha) \mid b = \alpha(x)]$$

$$= \frac{n(\alpha_0) + |V_{\text{fo}}(\alpha_0)|}{2n(\alpha_0)} \mathbb{E}[p(\alpha_0 \cup \{x \mapsto b\}, \alpha) \mid b = \alpha(x)]$$

$$= \frac{n(\alpha_0) + |V_{\text{fo}}(\alpha_0)|}{2n(\alpha_0)} \cdot \sum_{X' \in V(\alpha_0)} \frac{w(X')}{n(\alpha_0) + |V_{\text{fo}}(\alpha_0)|} p(\alpha_0 \cup \{X' \mapsto \alpha(X')\}, \alpha)$$

$$= \frac{1}{2} \sum_{X' \in V(\alpha_0)} \frac{1}{n(\alpha_0)} w(X') p(\alpha_0 \cup \{X' \mapsto \alpha(X')\}, \alpha)$$

$$= \frac{1}{2} \mathbb{E}_X [w(X) p(\alpha_0 \cup \{X \mapsto \alpha(X)\}, \alpha)].$$

Now a naive continuation would be to apply Jensen’s inequality and proceed similarly as before. However, this is unfortunately not possible.
The problem is that certain choices of $X$ are much more likely to produce $\alpha$ than others (and thus the probability $p(\alpha_0 \cup \{X \mapsto \alpha(X)\}, \alpha)$ varies wildly). On the other hand, Jensen’s inequality gives only good estimates when the value under the expectation is very concentrated. A remedy to this situation is to alter the variable the expectation is over, in the sense that the new random variable assumes variables $X'$ proportional to the increase of the probability of obtaining $\alpha$ after fixing $X'$ to $\alpha(X')$. It turns out that the correct distribution is proportional to $\text{lkd}(\alpha_0, \alpha_0 \cup \{X' \mapsto \alpha(X')\})$, the new likelihood of $\alpha$ after setting $X'$ to $\alpha(X')$.

For shorter notation define $l(X') := \frac{\text{lkd}(\alpha_0 \cup \{X' \mapsto \alpha(X')\}, \alpha)}{\text{lkd}(\alpha_0, \alpha)}$, the relative increase of new likelihood.

Let $\xi$ be a random variable in $V(\alpha)$ taking values $\xi' \in V(\alpha_0)$ with probability distribution $p_\xi(\xi') = \frac{l(\xi')}{|S|}$. The probabilities sum up to 1 due to Lemma 4.8 (iii).

Now we change the expectation to $\xi$, starting from the penultimate line above:

$$p(\alpha_0, \alpha) = \frac{1}{2} \sum_{X' \in V(\alpha_0)} \frac{1}{n(\alpha_0)} w(X') p(\alpha_0 \cup \{X' \mapsto \alpha(X')\}, \alpha)$$

$$= \frac{1}{2n(\alpha_0)} \sum_{\xi' \in V(\alpha_0)} \frac{l(\xi') |S|}{l(\xi')} w(\xi') p(\alpha_0 \cup \{\xi' \mapsto \alpha(\xi')\}, \alpha)$$

$$= \frac{1}{2n(\alpha_0)} \mathbb{E}_\xi \left[ \frac{|S|}{l(\xi')} w(\xi) p(\alpha_0 \cup \{\xi \mapsto \alpha(\xi)\}, \alpha) \right].$$

Now apply the induction hypothesis stating that $p(\alpha_0 \cup \{\xi \mapsto \alpha(\xi)\}, \alpha) \geq \text{lkd}(\alpha_0 \cup \{\xi \mapsto \alpha(\xi)\}) 2^{-\text{cost}(\alpha_0 \cup \{\xi \mapsto \alpha(\xi)\}, \alpha)}$. This gives

$$p(\alpha_0, \alpha) = \text{lkd}(\alpha_0, \alpha) \frac{|S|}{2n(\alpha_0)} \mathbb{E}_\xi \left[ w(\xi) 2^{-\text{cost}(\alpha_0 \cup \{\xi \mapsto \alpha(\xi)\}, \alpha)} \right],$$

as the new likelihood cancels out. Now inside the expectation we have something very similar to Lemma 4.12. Also the likelihood is pulled to the front of the expectation, matching the statement we need to prove. The only unusual thing is that the expectation is over $\xi$ and not over $X$. We are now ready to use Jensen’s inequality, and obtain

$$p(\alpha_0, \alpha) \geq \text{lkd}(\alpha_0, \alpha) 2^{\log \frac{|S|}{2n(\alpha_0)} + \mathbb{E}[\log w(\xi)] + \mathbb{E}[\text{cost}(\alpha_0 \cup \{\xi \mapsto \alpha(\xi)\}, \alpha)]},$$

(4.1)
As assigning frozen (including frozen) variables does not change the likelihood (Lemma 4.8 (i)), $\xi$ is uniform on the frozen variables and each is attained with probability $\frac{1}{|S|}$, exactly as in $X$ from Lemma 4.12; only the non-frozen variables are skewed. Thus

$$E \left[ \log w(\xi) \right] = \Pr (w(\xi = 2)) = \frac{|V_{fo}(\alpha_0)|}{|S|}. \quad (4.2)$$

Dealing with the expectation of the new cost is trickier, but essentially we just apply the statements from in the previous sections in the right way. We write as a shorthand $\alpha_0(\xi') := \alpha_0 \cup \{\xi' \mapsto \alpha(\xi')\}$, and have

$$E \left[ -\text{cost}(\alpha_0(\xi), \alpha) \right] =$$

$$= - \sum_{\xi' \in V(\alpha_0)} \frac{l(\xi')}{|S|} \text{cost}(\alpha_0(\xi'), \alpha)$$

$$= \frac{1}{\text{lkhd}(\alpha_0, \alpha)|S|} \sum_{\xi' \in V(\alpha_0)} \text{lkhd}(\alpha_0(\xi'), \alpha) \text{cost}(\alpha_0(\xi'), \alpha) \quad (4.3)$$

$$= \frac{1}{\text{lkhd}(\alpha_0, \alpha)|S|} \sum_{\xi' \in V(\alpha_0)} \sum_{x' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x')$$

$$= \frac{1}{\text{lkhd}(\alpha_0, \alpha)|S|} \sum_{x' \in V(\alpha_0)} \sum_{\xi' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x').$$

The inner sum actually corresponds to the expectation of Lemma 4.10, and we can use the lemma for frozen, non-forced variables $x'$:

$$\sum_{\xi' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x') \leq |S| \text{wcost}(\alpha_0, \alpha, x') - \text{lkhd}(\alpha_0, \alpha).$$

For non-frozen variables $x'$, we have by Lemma 4.8 (iii) and the defini-
\[ \sum_{\xi' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x') = \]
\[ = \sum_{\xi' \in V(\alpha_0) \setminus x'} \text{wcost}(\alpha_0(\xi'), \alpha, x') \]
\[ \leq \sum_{\xi' \in V(\alpha_0) \setminus x'} \text{lkhd}(\alpha_0(\xi'), \alpha) S_k^{(D)} \]
\[ = |S| \text{lkhd}(\alpha_0, \alpha) S_k^{(D)} - \text{lkhd}(\alpha_0(x'), \alpha) S_k^{(D)} \]
\[ = |S| \text{wcost}(\alpha_0, \alpha, x') - \text{lkhd}(\alpha_0(x'), \alpha) S_k^{(D)}. \]

For forced variables \( x' \) the cost is always 0, so trivially
\[ \sum_{\xi' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x') \leq |S| \text{wcost}(\alpha_0, \alpha, x') \]

Summing over all variables \( x' \) gives
\[ \sum_{x' \in V(\alpha_0)} \sum_{\xi' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x') \leq \]
\[ \leq |S| \text{lkhd}(\alpha_0, \alpha) \text{cost}(\alpha_0, \alpha) \]
\[ - |V_{fr}(\alpha_0)| \text{lkhd}(\alpha_0, \alpha) \]
\[ - \sum_{x' \in V_{nf}(\alpha_0)} \text{lkhd}(\alpha_0(x'), \alpha) S_k^{(D)}, \]

and as by Lemma 4.8 (i) and (iii)
\[ \sum_{x' \in V_{nf}(\alpha_0)} \text{lkhd}(\alpha_0(x'), \alpha) = 2|V_{nf}(\alpha_0)| \text{lkhd}(\alpha_0, \alpha), \]

we have
\[ \sum_{x' \in V(\alpha_0)} \sum_{\xi' \in V(\alpha_0)} \text{wcost}(\alpha_0(\xi'), \alpha, x') \leq \]
\[ \leq |S| \text{lkhd}(\alpha_0, \alpha) \left( \text{cost}(\alpha_0, \alpha) - \frac{|V_{fr}(\alpha_0)|}{|S|} - 2S_k^{(D)} \frac{|V_{nf}(\alpha_0)|}{|S|} \right). \]
Inserting this back into (4.3), we have
\[ \mathbf{E}_\xi [-\text{cost}(\alpha_0(\xi'), \alpha)] \leq \text{cost}(\alpha_0, \alpha) - \frac{|V_{fr}(\alpha_0)|}{|S|} - 2S_k^{(D)} \frac{|V_{nf}(\alpha_0)|}{|S|}, \]
and inserting this and (4.2) into (4.1) gives
\[ p(\alpha_0, \alpha) \geq \]
\[ \geq \text{lkhd}(\alpha_0, \alpha) 2^{\log \left\{ \frac{|S|}{2n(\alpha_0)} + \frac{|V_{fr}(\alpha_0)|}{|S|} - \text{cost}(\alpha_0, \alpha) + \frac{|V_{fr}(\alpha_0)|}{|S|} + 2S_k^{(D)} \frac{|V_{nf}(\alpha_0)|}{|S|} \right\}}. \]
The last steps of the proof are completely analogous to Lemma 4.12. \( \Box \)
In the previous chapters, we considered only assignments that assigned binary (i.e. Boolean) values to the variables. In this chapter we see what happens when the variables can assume values from \( \{1, \ldots, d\} \), for some \( d \in \mathbb{N} \). The corresponding formulas are called \((d, k)\)-clause satisfaction problems (ClSPs), where we retain the condition that clauses can have size at most \( k \). We also call the corresponding decision problem \((d, k)\)-ClSP. For a formal definition see Chapter 2.

It turns out that we can adapt the PPSZ algorithm to ClSPs in a straightforward manner. The only difference is when we set the variables. The definition of \( D \)-implication \( F \models_D (x \neq c) \) is analogous to before; the only difference is that this forbids just one out of \( d \) possible values for \( x \). Thus, when setting a variable \( x \), we check for each \( c \in [d] \) whether \( F \models_D (x \neq c) \), and then choose the value of \( x \) uniformly at random from the remaining values. This means that a variable can be partially guessed, respectively partially forced. This makes the analysis more complicated and introduces new potentially harmful correlations. However, for Unique \((d, k)\)-ClSP we will see that everything turns out well.

For general \((d, k)\)-ClSP, we will proceed as in Chapter 4. Here variables might be partially frozen or non-frozen. We do not completely know how to deal with these; in our analysis we will treat all (partially) non-frozen variables the same. However, this leads to a gap between Unique \((d, k)\)-ClSP and \((d, k)\)-ClSP for large \( d \) and small \( k \), and to the open problem whether this gap can be closed.

In this chapter we will assume that \( d \) and \( k \) are constants, and that
both \( d \geq 2 \) and \( k \geq 2 \) and either \( d \geq 3 \) or \( k \geq 3 \) (as \((2,2)\)-ClSP corresponds to 2-SAT and is solvable in polynomial time).

This chapter is based on a yet unpublished manuscript, joint work with Isabelle Hurbain, Sebastian Millius, Robin Moser, Dominik Scheder, and May Szendlá, which in turn is based on the bachelor thesis of May Szendlá [35], the master thesis of Sebastian Millius [21], and the semester thesis of Isabelle Hurbain [12].

**Previous Results.** As in this thesis we consider only randomized algorithms, we will not mention the fastest deterministic algorithms separately. For \( k \)-SAT, the currently fastest known algorithm is the PPSZ algorithm by Paturi, Pudlák, Saks and Zane [24], as discussed in Chapters 3 and 4. By so-called downsampling this algorithm can be used to solve \((d,k)\)-ClSP: For every variable, we simply remove all but two values, and interpret the result as a \((\leq k)\)-CNF formula. Any satisfying assignment is preserved with probability \(\left(\frac{2}{d}\right)^n\), so the running time is the running time for \(k\)-SAT multiplied by \(\left(\frac{d}{2}\right)^n\). The \((d,k)\)-ClSP algorithm from Schöning [33] has exactly the same running time as achieved by downsampling to its \(k\)-SAT variant, and thus is slower than downsampling to PPSZ, as it is also slower than PPSZ for \(k\)-SAT.

There have been a few moderately exponential algorithms addressing \((d,k)\)-ClSP specifically. Beigel and Eppstein give an algorithm for \((d,2)\)-ClSP running in time \(O((0.4518d)^n)\) for \(d > 3\), and in time \(O(1.365^n)\) for \(d = 3\). Feder and Motwani [4] give a \((d,2)\)-ClSP algorithm based on the PPZ algorithm [25], the predecessor of the PPSZ algorithm, improving on the algorithm by Beigel and Eppstein for large \(d\). Li, Li, Liu, and Xu [19] generalized this to \((d,k)\)-ClSP, but with a weaker analysis. Scheder [31] showed how to use the full power of PPZ for \((d,k)\)-ClSP.

**Our Contribution.** We show how to use the PPSZ algorithm for \((d,k)\)-ClSP. We need the following integral to state the running time formally:
Definition 5.1.

Let $S_{(d,k)} := \sum_{j=0}^{d-1} \binom{d-1}{j} \log_d(1+j)$

\[ \cdot \int_0^1 \left( s^{d-1-j} (1-s)^j \cdot \frac{(1-s^{d-1}) \left( \frac{1}{k-1} s^{k-1} - (d-1)s^{d-2} \right)}{(1-s^{d-1})^2} + \frac{(s^{d-1} - s^{d-1}) (d-1)s^{(d-2)}}{(1-s^{d-1})^2} \right) ds. \]

For Unique $(d,k)$-ClSP we obtain the following bound:

**Theorem 5.2.** There exists a randomized algorithm for Unique $(d,k)$-ClSP with one-sided error that runs in time $O(d^{S_{(d,k)} n + o(n)})$.

For $k = 2$ and $d \geq 5$, and for $k \geq 3$, we are faster than the previous algorithms in the unique case for all $(d,k)$-values we checked.

The bound of the unique case holds for the general case as long as $S_{(d,k)} \leq 1 - \frac{\log_d e}{2}$. For $k = 2$, this makes our analysis useless, as it is slower than downsampling to 2-SAT. For $k = 3$, the inequality holds only for $d \leq 5$. For $k \geq 4$, the inequality always holds for the values we checked; we conjecture that it holds for all such values, but we were not able to bound the above integral correspondingly. Let $G_{(d,k)} := \max\{S_{(d,k)}, 1 - \frac{\log_d e}{2}\}$. For general $(d,k)$-ClSP, we show:

**Theorem 5.3.** There exists a randomized algorithm for $(d,k)$-ClSP with one-sided error that runs in time $O(d^{G_{(d,k)} n + o(n)})$.

For general $(d,k)$-ClSP, we are faster for $k = 3$ and $d \leq 10$, and for $k \geq 4$ for all values of $d$ we checked. Comparison of our results (PPSZ Unique, PPSZ Gen.) with the previous results and downsampling to PPSZ is shown in Tables 5.1 and 5.2.

### 5.1 Unique $(d,k)$-ClSP

In this section we will consider the case of Unique $(d,k)$-ClSP and prove Theorem 5.2. This section is structured similarly to Chapter 3, and we will refer to it when proofs can be directly adapted. First we define $D$-implication formally for $(d,k)$-ClSP:
### Table 5.1:
Constant $c$ so that the algorithm for $\left(d, 2\right)$-ClSP runs in time $c^{n+o(n)}$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1.434</td>
<td>1.820</td>
<td>1.365</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.849</td>
<td>2.427</td>
<td>1.808</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2.254</td>
<td>3.033</td>
<td>2.259</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2.652</td>
<td>3.640</td>
<td>2.711</td>
<td>2.994</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4.208</td>
<td>6.066</td>
<td>4.518</td>
<td>4.529</td>
<td>5</td>
</tr>
</tbody>
</table>

### Table 5.2:
Constant $c$ so that the algorithm for $\left(d, k\right)$-ClSP runs in time $c^{n+o(n)}$.

<table>
<thead>
<tr>
<th>$d$</th>
<th>$k$</th>
<th>PPSZ Unique</th>
<th>PPSZ Gen.</th>
<th>PPZ [31]</th>
<th>Downsampl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>1.901</td>
<td>1.901</td>
<td>2.162</td>
<td>1.961</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2.479</td>
<td>2.479</td>
<td>2.729</td>
<td>2.615</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3.049</td>
<td>3.049</td>
<td>3.291</td>
<td>3.268</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3.614</td>
<td>3.640</td>
<td>3.850</td>
<td>3.922</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4.175</td>
<td>4.246</td>
<td>4.407</td>
<td>4.575</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>4.733</td>
<td>4.853</td>
<td>4.963</td>
<td>5.229</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5.289</td>
<td>5.459</td>
<td>5.516</td>
<td>5.882</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>5.844</td>
<td>6.066</td>
<td>6.069</td>
<td>6.536</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>6.397</td>
<td>6.672</td>
<td>6.621</td>
<td>7.189</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>8.602</td>
<td>9.098</td>
<td>8.821</td>
<td>9.803</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2.153</td>
<td>2.153</td>
<td>2.351</td>
<td>2.204</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2.823</td>
<td>2.823</td>
<td>3.014</td>
<td>2.938</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3.487</td>
<td>3.487</td>
<td>3.672</td>
<td>3.673</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>6.761</td>
<td>6.761</td>
<td>6.935</td>
<td>7.345</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>10.006</td>
<td>10.006</td>
<td>10.176</td>
<td>11.018</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2.310</td>
<td>2.310</td>
<td>2.471</td>
<td>2.355</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3.040</td>
<td>3.040</td>
<td>3.195</td>
<td>3.139</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3.764</td>
<td>3.764</td>
<td>3.915</td>
<td>3.924</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>7.348</td>
<td>7.348</td>
<td>7.490</td>
<td>7.848</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>10.906</td>
<td>10.906</td>
<td>11.045</td>
<td>11.771</td>
</tr>
</tbody>
</table>
Definition 5.4. Let $F$ be a ClSP, let $D$ be a number, and let $l = (x \neq c)$ be a literal. We say that $F$ implies $l$, in writing $F \models l$, if any assignment $\alpha$ satisfying $F$ also satisfies $l$ (i.e. $\alpha(x) \neq c$). We say $F$ $D$-implies $l$, in writing $F \models_D l$, if there is some $G \subseteq F$ with $|G| \leq D$ having $G \models l$.

Algorithm 3 PPSZ((d,k)-ClSP $F$, variable set $V$, integer $D$)

Choose $\pi$ u.a.r. from all permutations on $V$

return PPSZ($F, V, \pi, D$)

Algorithm 4 PPSZ((d,k)-ClSP $F$, variable set $V$, permutation $\pi$, integer $D$)

Let $(x_1, x_2, \ldots, x_{|V|})$ be the set $V$ ordered according to $\pi$

Let $\gamma$ be a partial assignment on $V$, initially empty

for $i \leftarrow 1$ to $|V|$ do
  $C = [d] \setminus \{c \in [d] \mid F^{[\gamma]} \models_D (x_i \neq c)\}$
  \{C contains all values of $x_i$ not forbidden by $D$-implication\}
  $\gamma(x_i) \leftarrow$ u.a.r. $C$ \{let $\gamma(x_i) \leftarrow 1$ if $|C| = 0$\}

end for

return $\gamma$

We define the set of allowed values of $x$ w.r.t. $\pi$, which is similar to forced and guessed variables in the Boolean case.

Definition 5.5. Let $F$ be a (d,k)-ClSP over $V$, $\alpha$ a satisfying assignment of $F$ on $V$, $\pi = (x_1, \ldots, x_{|V|})$ a permutation on $V$, and $D$ be a number. The set of allowed values of $x_i$ is defined as

$$\mathcal{A}(x_i, F, V, \alpha, \pi, D) = [d] \setminus \{c \in [d] \mid F^{[\alpha']} \models_D (x \neq c)\},$$

where $\alpha' = \{x_1 \mapsto \alpha(x_1), \ldots, x_{i-1} \mapsto \alpha(x_{i-1})\}$.

For PPSZ($F, V, \pi, D$) to return the satisfying assignment $\alpha$ it needs to pick $\gamma(x) = \alpha(x)$ for all $x$. Given $\pi$, the probability that $\gamma(x) = \alpha(x)$ is $\frac{1}{|\mathcal{A}(x, F, V, \alpha, \pi, D)|}$. As all choices are independent, we have

$$\Pr(\text{PPSZ}(F, V, \pi, D) \text{ returns } \alpha) = \prod_{x \in V} \frac{1}{|\mathcal{A}(x, F, V, \alpha, \pi, D)|}.$$
and for a permutation \( \pi \) chosen uniformly at random, we get

\[
\Pr(\text{PPSZ}(F, V, D) \text{ returns } \alpha) = E_\pi \left[ \prod_{x \in V} \frac{1}{|A(x, F, V, \alpha, \pi, D)|} \right].
\]

The formal reasoning is analogous to Observation 3.4 and the paragraph following it. We apply Jensen’s inequality (Theorem A.3) with the convex function \( x \mapsto d - x \) and obtain

\[
\Pr(\text{PPSZ}(F, V, D) \text{ returns } \alpha) \geq d^{-\sum_{x \in V} E_\pi[\log_d |A(x, F, V, \alpha, \pi, D)|]]. \quad (5.1)
\]

In the remainder of this section, fix a \((d, k)\)-ClSP \( F \) over \( n \) variables \( V \) with a unique satisfying assignment \( \alpha \). We will give an upper bound to \( E_\pi[\log_d |A(x, F, V, \alpha, \pi, D)|] \); as in the Boolean case we use so-called critical clause trees. Instead of a probability we have to bound the expectation of a logarithm here, which will make things more complicated.

The critical clause trees considered here are similar to the ones in Chapter 3. However, here every variable will have \( d - 1 \) such trees, one for each possible value except \( \alpha(x) \). In the Boolean case, whenever a critical clause tree had at most \( D \) reachable nodes (as defined in Chapter 3), the variable would be forced. Here we will ask how many trees have at most \( \frac{D}{d - 1} \) reachable nodes; every such tree will forbid one possible value of \( x \). We show that if all but \( j \) trees have at most \( \frac{D}{d - 1} \) reachable nodes then there will be at most \( j + 1 \) allowed values for PPSZ to choose from; ultimately we need to bound the logarithm of the expectation of this number. In the following we will assume without loss of generality that \( \alpha \) is the all-\( d \) assignment, i.e. \( \alpha(x) = d \) for all \( x \in V \).

We construct a collection of trees \( \{T_x^{(c)}\}_{x \in V, c \in [d - 1]} \); for a given variable \( x \), the set of trees \( \{T_x^{(c)}\}_{c \in [d - 1]} \) is called the set of critical clause trees of \( x \), and \( T_x^{(c)} \) is called the critical clause tree of \( x \) for (value) \( c \). As in the Boolean case there is some choice how these trees are built; we make these choices arbitrarily and consider the collection \( \{T_x^{(c)}\}_{x \in V, c \in [d - 1]} \) to be fixed in the remainder.

We call \( T \) a rooted tree with children into \( j \) directions if the following holds. \( T \) is a tree with a designated root, \( \text{root}(T) \). The children of a vertex \( v \) are partitioned into \( j \) groups which we denote \( \text{Children}_1(v), \text{Children}_2(v), \ldots, \text{Children}_j(v) \). Each child belongs to exactly one group, i.e. \( \text{Children}_a(v) \) and \( \text{Children}_b(v) \) are disjoint sets whenever \( a \neq b \).

We build, for every \( x \in V \) and for every \( c \in [d - 1] \), a rooted tree with children into \( (d - 1) \) directions \( T_x^{(c)} \), where every node \( u \in V(T) \renc3
is labeled both with a variable $x \in V$, which we denote by var-label($u$), and a list of clauses $C \in F$, denoted by clause-label($u$). Here is how $T_x^{(c)}$ is built for a fixed $x \in V$ and a fixed $c \in [d - 1]$

1. Start with $T_x^{(c)}$ consisting of a single root. The var-label of this root is $x$, and its clause-label is empty.

2. Let $C$ be a clause that is not satisfied by $\alpha[x \mapsto c]$. Let the clause-label of the root be $C$. For each literal $(y \neq d) \in C$, add a child to the root, which is var-labeled with the variable the literal is over. We define these nodes to be in Children$_c(x)$.

Note that the root has only one clause-label and has only children in direction $c$.

3. As long as there is a leaf $u \in V(T)$ that has an empty clause-label, do the following:

(a) Define $W := \{\text{var-label}(v) \mid v \in V(T) \text{ ancestor of } u \text{ in } T\}$, where ancestor includes $u$ itself and the root.

(b) Consider the path from $u$ to the root. Define $\ell_1, \ldots, \ell_m$ such that $y_0 = \text{root}(T)$, $y_1 \in \text{Children}_{\ell_1}(y_0)$, $y_2 \in \text{Children}_{\ell_2}(y_1)$, $\ldots$, $y_{m-1} \in \text{Children}_{\ell_{m-1}}(y_{m-2})$, $u \in \text{Children}_{\ell_m}(y_{m-1})$, where $m$ is the depth of $u$. Define the partial assignment $\mu_0$ as follows: For $x_i = \text{var-label}(y_i)$, let $\mu_0(x_i) = \ell_{i+1}$, for $i \in \{0, \ldots, m - 1\}$.

(c) For $j = 1$ to $d - 1$, we define $\alpha_j := \alpha[\mu_0 \cup \{\text{var-label}(u) \mapsto j\}]$. For each $j$, let $C_j$ be a clause that is not satisfied by $\alpha_j$ (this exists as $\alpha$ is the unique satisfying assignment). Add $C_j$ to clause-label($u$).

Observe that literals $(y \neq b) \in C_j$ for $b \neq d$ must be over variables $y$ in $W$. Furthermore $b$ corresponds to the direction at the node of $y$ that leads to $u$ (if the node of $y$ is the current leaf $u$, then $b = j$).

(d) For $j = 1$ to $d - 1$, add for each literal $(y \neq d)$ in $C_j$ a node to Children$_j(v)$ with var-label $y$.

The critical clause trees have the following properties (similar to the Boolean case):

- The root has at most $(k - 1)$ children; any other node has at most $(d - 1)(k - 1)$ children (every clause has at most $k - 1$ literals $(y \neq d)$, otherwise it would not be satisfied by $\alpha$).
• No node has the same var-label as one of its proper ancestors (a literal \((x \neq d)\) of a var-label of a proper ancestor would make \(C_j\) satisfied by \(\alpha_j\)).

• Thus the height of the tree cannot exceed \(n - 1\), so the process terminates, and the trees are well-defined.

As in Chapter 3 we now consider \(\pi\) as a placement: the values \(\pi(x)\), called place of \(x\), are chosen independently and uniformly at random from \([0, 1]\) for each \(x \in V\); the permutation is obtained by ranking the variables according to their values. Ties happen with probability 0 and can be ignored. In the following, we denote by \(\pi\) both the placement and the induced permutation.

**Definition 5.6.** Let \(\gamma \in [0, 1]\) and \(T_x^{(c)}\) be a critical clause tree. A node \(u \in T_x^{(c)}\) is reachable at time \(\gamma\) w.r.t. \(\pi\) if there exists a path \(v_0, v_1, \ldots, v_m\) such that \(v_0\) is the root of the tree, \(v_m = u\) and \(\pi(\text{var-label}(v_i)) \geq \gamma\) for all \(0 \leq i \leq m\). Let us denote by \(\text{Reachable}(T_x^{(c)}, \gamma, \pi)\) the set of all nodes in \(T_x^{(c)}\) reachable at time \(\gamma\) w.r.t. \(\pi\).

The following analogue to Lemma 3.10 connects reachability with \(D\)-implication.

**Lemma 5.7.** If we have \(\left| \text{Reachable}(T_x^{(c)}, \pi(x), \pi) \right| \leq \frac{D}{d-1}\), then \((x \neq c)\) is \(D\)-implied by \(F[\alpha']\) where \(\alpha'\) is the restriction of \(\alpha\) to the variables \(y \in V\) with \(\pi(y) < \pi(x)\).

**Proof.** Remember that \(\alpha\) is the all-\(d\) assignment. Let \(G\) be the subformula of \(F\) consisting of the union of all the clause-label sets of reachable nodes in \(T_x^{(c)}\). Since \(\left| \text{Reachable}(T_x^{(c)}, \pi(x), \pi) \right| \leq \frac{D}{d-1}\), and each clause-label contains at most \((d-1)\) clauses, we have \(|G| \leq D\). We claim that \(G[\alpha'] \subseteq F[\alpha']\) implies \((x \neq c)\).

Suppose that \(G[\alpha']\) has a satisfying assignment that sets \(x\) to \(c\). Then we can fix an assignment \(\nu: V \rightarrow [d]\) which is compatible with \(\alpha'\), which has \(\nu(x) = c\) and which satisfies \(G[\alpha']\). Choose a path in \(T_x^{(c)}\), starting at the root (labeled \(x\)) as follows: For the current node \(v\), consider children in direction of \(\nu(\text{var-label}(v))\). If some such child \(v'\) has \(\nu(\text{var-label}(v')) \neq \alpha(\text{var-label}(v'))\) (i.e. \(\nu(\text{var-label}(v')) \neq d\), add \(v'\) to the path and continue at \(v'\). Otherwise stop.

Let \(u\) be the endpoint of the path. Since \(\nu\) is compatible with \(\alpha\) on all the variables with place before \(x\), \(u\) must be reachable. For all
5.1. Unique \((d, k)-\text{CISP}\)

children \(z\) of \(u\), we have that \(\nu(\text{var-label}(z)) = d\), and all ancestors \(y\) of \(u\) are such that \(\nu(\text{var-label}(y)) \neq d\).

Let \(j := \nu(\text{var-label}(u))\), and \(C_j\) be the \(j\)-th clause of \(\text{clause-label}(u)\) (if \(u\) is the root, take the single clause in \(\text{clause-label}(u)\)). \(C_j\) is not satisfied by \(\nu\): Any literal \((y \neq d) \in C_j\) corresponds to a child of \(u\) in direction \(j\), and thus \(\nu(y) = d\) (otherwise we continued the path). By the observation at 3(d) any other literal \((y \neq b)\) corresponds to a node \(v\) on the path, and \(b\) is the direction taken at \(v\) (if the node of \(v\) is the endpoint \(u\), then \(b = j\), this holds also if \(u\) is the root). However, we constructed the path such that \(b = \nu(y)\); thus these literals are also not satisfied. \(\square\)

It follows immediately that

\[
\Pr_{\pi}(c \notin A(x, F, V, \alpha, \pi, D)) \geq \Pr_{\pi}\left(|\text{Reachable}(T_x^{(c)}, \pi(x), \pi)| \leq \frac{D}{d-1}\right).
\]

For \(1 \leq c \leq d - 1\), we define

\[
Y_x^{(c)} = \begin{cases} 
1 & \text{if } |\text{Reachable}(T_x^{(c)}, \pi(x), \pi)| > \frac{D}{d-1} \\
0 & \text{otherwise}
\end{cases}
\]

Since we always have \(d \in A(x, F, V, \alpha, \pi, D)\), we have

\[
|A(x, F, V, \alpha, \pi, D)| \leq 1 + \sum_{c=1}^{d-1} Y_x^{(c)},
\]

and so we want to bound

\[
\mathbb{E}[\log_d |A(x, F, V, \alpha, \pi, D)|] \leq \mathbb{E}\left[\log_d \left(1 + \sum_{c=1}^{d-1} Y_x^{(c)}\right)\right]. \quad (5.2)
\]

This reduces the problem to a probabilistic calculation on trees, similar to the Boolean case: when sorting the nodes of a set of trees according to a random permutation and deleting all nodes whose place is after the root, what is the logarithm of the expected number of trees that have more than \(\frac{D}{d-1}\) nodes reachable? In what follows, we will first fix a place \(p\) for the root, and compute said expectation conditioned on the place of the root being equal to \(p\). We will then integrate that result to remove the conditioning by the law of conditional expectation.
First we consider a single, infinite tree, and a lemma corresponding to Observation 3.13 and Lemma 3.15, with the difference the degree of the root differs from the degree of the other nodes. We need the following definition similar to Definition 3.12.

**Definition 5.8.** Define

\[
\tilde{R}_{(d,k)}(p) := (p + (1 - p)R_{(d,k)}(p))^{(k-1)},
\]

where \(R_{(d,k)}(p)\) is the smallest non-negative \(q\) satisfying the equation

\[
q = (p + (1 - p) \cdot q)^{(d-1)(k-1)}.
\]

Observe that \(R_{(d,k)} = R_{(d-1)(k-1)+1}\) from Chapter 3, and thus is also well-defined.

**Lemma 5.9.** Let \(T_\infty\) be an infinite rooted full tree, where the root has degree \(k - 1\), and all other nodes have degree \((d-1)(k-1)\). Consider the following random experiment: each non-root from \(T_\infty\) is deleted (along with its subtree) independently from all other nodes with probability \(p\). Then

\[
\Pr(T' \text{ finite}) \geq \tilde{R}_{(d,k)}(p)
\]

and

\[
\lim_{H \to \infty} \Pr(h(T') \leq H) = \Pr(T' \text{ finite}).
\]

**Proof.** Let \(y_i\) be the \(i\)-th child of the root of \(T_\infty\) and let \(Q_i\) be the subtree rooted at \(y_i\). Let \(Q'_i\) be the tree \(Q_i\) after deleting all nodes deleted in \(T_\infty\) except for \(y_i\). Since what happens to \(y_i\) is independent of what happens in \(Q'_i\), we have

\[
\Pr(T' \text{ finite}) = \Pr \left( \bigwedge_{i=1}^{k-1} y_i \text{ deleted} \lor (y_i \text{ not deleted} \land Q'_i \text{ finite}) \right)
\]

\[
\geq \prod_{i=1}^{k-1} (p + (1 - p) \cdot \Pr(Q'_i \text{ finite})).
\]

\(Q_i\) is an infinite rooted full tree of degree \((d-1)(k-1)\), and by Observation 3.13, \(\Pr(Q'_i \text{ is finite}) \geq R_{(d,k)}(p)\). Thus,

\[
\Pr(T' \text{ finite}) \geq (p + (1 - p)R_{(d,k)}(p))^{(k-1)} = \tilde{R}_{(d,k)}(p).
\]

The second statement has the same proof as Lemma 3.15, as how exactly \(T_\infty\) (and \(T'\)) look is irrelevant for the proof. \(\square\)
Now we consider what happens for multiple trees, where all nodes are deleted independently. Apart from this, the following lemma corresponds to Lemma 3.16.

**Lemma 5.10.** For each \( p \in [0, 1] \), there exists a sequence \( \epsilon_1^{(1)}(p), \epsilon_2^{(2)}(p), \ldots \in \mathbb{R}_0^+ \) of numbers, having \( \epsilon^{(H)}_{(d, k)}(p) \to 0 \) for \( H \to \infty \) such that the following holds: Let \( \{T_{\infty}^{(c)}\}_{c \in [d-1]} \) be a set of \( (d - 1) \) infinite rooted full trees of root degree \((k - 1)\) and non-root degree \((d - 1)(k - 1)\). For all \( c \in [d - 1] \), each non-root node from \( T_{\infty}^{(c)} \) is deleted (along with its subtree) independently from all other nodes with probability \( p \); denote the resulting tree \( T'^{(c)}_{\infty} \). Define \( \mathcal{B}_H \) as the set of values \( c \) such that \( T'^{(c)}_{\infty} \) has a height greater than \( H \). We have

\[
\mathbb{E}[\log_d(1 + |\mathcal{B}_H|)] \leq \sum_{j=1}^{d-1} \log_d(1 + j) \binom{d-1}{j} \tilde{R}_{(d, k)}(p)^{d-1-j}(1 - \tilde{R}_{(d, k)}(p))^j + \epsilon^{(H)}_{(d, k)}(p).
\]

**Proof.** Let \( q_H := \Pr(c \not\in \mathcal{B}_H) \) for some fixed \( c \) (this is the same for all \( c \)), let \( q_\infty := \lim_{H \to \infty} q_H \), and let \( q^* := \tilde{R}_{(d, k)}(p) \). From Lemma 5.9 we know \( q_\infty \geq q^* \). By definition, the events \( c \in \mathcal{B}_H \) are mutually independent.

To prove the statement, we first consider a more general set: For \( q \in [0, 1] \), define \( C(q) \) as a random subset of \([d - 1]\), where each \( c \in [d - 1] \) is added with probability \((1 - q)\), independently. Then, as \(|C_q|\) is binomially distributed,

\[
\mathbb{E}[\log_d(1 + |C(q)|)] = \sum_{j=1}^{d-1} \log_d(1 + j) \binom{d-1}{j} (1 - q)^j q^{d-1-j}.
\]

Observe that \( \mathcal{B}_H = C(q_H) \) in distribution. Thus

\[
\mathbb{E}[\log_d(1 + |\mathcal{B}_H|)] = \mathbb{E}[\log_d(1 + |C(q_H)|)].
\]

Furthermore,

\[
\lim_{H \to \infty} \mathbb{E}[\log_d(1 + |C(q_H)|)] = \mathbb{E}[\log_d(1 + |C(q_\infty)|)],
\]

as this expectation is a continuous function in the argument of \( C \).
\( \mathbf{E} \left[ \log_d (1 + |\mathcal{C}(q_\infty)|) \right] \leq \mathbf{E} \left[ \log_d (1 + |\mathcal{C}(q^*)|) \right] \) is intuitively clear: \( x \mapsto \log_d (1 + x) \) is monotonically non-decreasing and \( (1 - q_\infty) \leq (1 - q^*) \). For a formal proof, see Lemma A.8.

Combining all the observations gives

\[
\lim_{H \to \infty} \mathbf{E} \left[ \log_d (1 + |\mathcal{B}_H|) \right] \leq \sum_{j=1}^{d-1} \log_d (1 + j) \binom{d - 1}{j} (1 - q^*)^j q^* d - 1 - j,
\]

where we defined \( q^* := \tilde{R}_{(d,k)}(p) \). The statement follows by defining \( \epsilon_{(d,k)}^{(H)}(p) \) appropriately.

The next step is to deal with dependencies between the node deletions. Here, there are two kinds of such dependencies: The dependencies inside one tree, as in the Boolean case. There we showed in Lemma 3.17 that these dependencies are only good for us, and we can use the same approach here.

The new dependencies are the dependencies between multiple trees. Critical clause trees of \( x \) for different values can share var-labels arbitrarily. This dependency is new, but we will again show that such dependencies are only beneficial.

We will prove the following extension to Lemma 5.10:

**Lemma 5.11.** For each \( p \in [0,1] \), there exists a sequence \( \epsilon_{(d,k)}^{(1)}(p) \), \( \epsilon_{(d,k)}^{(2)}(p), ... \in \mathbb{R}^+ \) of numbers, having \( \epsilon_{(d,k)}^{(H)}(p) \to 0 \) for \( H \to \infty \) such that the following holds:

Let \( \{T^{(c)}\}_{c \in [d-1]} \) be a set of \( (d-1) \) (finite) critical clause trees where for all \( c \in [d-1] \), the root of \( T^{(c)} \) has degree at most \( k - 1 \) and the other nodes have degree at most \( (k - 1)(d - 1) \). Let \( \sigma_c : V(T^{(c)}) \setminus \{\text{root}\} \to \{1, \ldots, r\} \) be labelings of the non-root nodes of the trees, such that on each path from a root to a leaf in \( T^{(c)} \), \( \sigma_c \) is injective. Let \( Z_1, Z_2, ..., Z_r \in \{0,1\} \) be mutually independent binary random variables, each of which takes value 1 with probability \( p \). Consider the experiment of drawing \( Z_1, ..., Z_r \) according to their distribution and then deleting all nodes \( u \) (along with their subtrees) with \( Z_{\sigma_c(u)} = 1 \) from all trees \( T^{(c)} \). Denote the resulting trees by \( T^{(c)}' \), and define \( J_H \) as the set of values \( c \) such
that $T^{r(c)}$ has a height greater than $H$. We have

$$\mathbb{E} \left[ \log_d (1 + |J_H|) \right] \leq \sum_{j=1}^{d-1} \left( \log_d (1 + j) \binom{d-1}{j} \tilde{R}_{(d,k)}(p)^{d-1-j} (1 - \tilde{R}_{(d,k)}(p))^j \right) + \epsilon_{(d,k)}^{(H)}(p).$$

Proof. We will proceed in three steps to prove the lemma:

- first, we argue that if the initial trees are finite instead of being infinite, the bound obtained on $\mathbb{E} \left[ \log_d (1 + |J_H|) \right]$ still holds,

- then, we prove that the bound still holds if we introduce dependencies within the labeling of different trees, while keeping the labeling of the different trees independent,

- finally, we prove that it still holds if we introduce dependencies between the labeling of the different trees.

For the first point, let us consider a family of $(d - 1)$ finite (not necessarily full) trees $\{T^{(c)}\}$ of root degree $(k - 1)$ and of non-root degree $(k - 1)(d - 1)$. Consider the following experiment: each non-root node from every tree of $\{T^{(c)}\}$ is deleted (along with its subtree) independently from all other nodes with probability $p$; denote the resulting tree by $T^{r(c)}$. We can couple this experiment to a random experiment conducted on the family $\{T^{(c)}\} \rightarrow \{T^{(c)}\} \rightarrow \{T^{(c)}\}$ of $d - 1$ infinite full trees where all nodes are also deleted independently (as defined in Lemma 5.10): the family $\{T^{(c)}\}$ is embeddable into $\{T^{(c)}\}$ with every root from a tree from $\{T^{(c)}\}$ coinciding with a root from a tree from $\{T^{(c)}\}$. If we delete every node from the trees from $\{T^{(c)}\}$ with probability $p$, the corresponding deletions are taking place in $\{T^{(c)}\}$. Since the trees $\{T^{(c)}\}$ are subtrees of the trees $\{T^{(c)}\}$, the height of each tree $T^{m(c)}$ is bounded by the height of $T_{\infty}^{(c)}$ after the deletions. Denote the set of values $c$ such that $T^{m(c)}$ has height more than $H$ by $L_H$; and we have $\mathbb{E} \left[ \log_d (1 + |L_H|) \right] \leq \mathbb{E} \left[ \log_d (1 + |B_H|) \right]$ (where $B_H$ is as defined in Lemma 5.10).

For the second point, assume that deletions are done according to the labels, but independently for each tree (i.e. there are binary random variables $Z_{i}^{(c)}$, for $i = 1, \ldots, r$ and $c = 1, \ldots, d - 1$, and the corresponding random variables are used for each tree). Denote the result of these deletions by $T^{m(c)}$. Completely analogous to
Lemma 3.17 (the only differences are the different degrees) we have that \( \Pr(h(T''(c)) \leq H) \geq \Pr(h(T'''(c)) \leq H) \). Denote the set of values \( c \) such that \( \{T''(c)\} \) has height more than \( H \) by \( K_H \). By Lemma A.8, \( E[\log_d(1 + |K_H|)] \leq E[\log_d(1 + |L_H|)] \), as each value \( c \) is in \( K_H \) (as in \( L_H \)) independently of the other values.

In the last step, we reintroduce the dependencies between the trees. Until now, we proceeded essentially proceeded as in the Boolean case. We will show \( E[\log_d(1 + |J_H|)] \leq E[\log_d(1 + |K_H|)] \), which together with the previous observations and Lemma 5.10 concludes the proof.

We will need the following lemma from the appendix:

**Lemma A.9.** Let \( X, T, T', U, V \) be finite vectors of real random variables, all independent, taking only finitely many values. Suppose \( T \) and \( T' \) are over \( \mathbb{R}^n \) and have the same distribution. Let \( f, g : \mathbb{R}^m \to \mathbb{R} \) (for \( m \geq n \)) be monotonically non-decreasing functions, and let \( h : \mathbb{R} \to \mathbb{R} \) be a concave function. Then

\[
E[h(f(X, T, U) + g(X, T, V))] \leq E[h(f(X, T, U) + g(X, T', V))].
\]

In this lemma, we can think of the left-hand side and the right-hand side as two different, but related, random experiments. The functions \( f \) and \( g \) are fed with random input. The vector \( X \) represents shared random input (between both functions), \( U \) and \( V \) are individual random input (for each function), and \( T \) and \( T' \) are random input that is shared in the first experiment between \( f \) and \( g \), but made independent in the second experiment.

Define, for \( i \in [r] \) and \( c \in [d - 1] \), \( Y_i := 1 - Z_i \) and \( Y_i^{(c)} := 1 - Z_i^{(c)} \). Whereas the \( Z \)-variables are 1 if a node is deleted, the \( Y \)-variables are 1 is a node is *not* deleted. For \( c \in [d - 1] \), define the variables

\[
\chi_J^{(c)} = \begin{cases} 1 & \text{if } c \in J_H \\ 0 & \text{otherwise} \end{cases}
\]

and

\[
\chi_K^{(c)} = \begin{cases} 1 & \text{if } c \in K_H \\ 0 & \text{otherwise} \end{cases}
\]

Observe that, for all \( c \in [d - 1] \),

\[
\chi_J^{(c)} = \bigvee_{\text{P path in } T^{(c)} \text{ from root to node at depth } H + 1} \bigwedge_{u \in P} Y_{\sigma_c(u)}
\]
and that
\[ \chi^{(c)}_K = \bigvee_{P\text{ path in } T^{(c)}} \bigwedge_{u \in P} Y^{(c)}_{\sigma(u)}, \]
which makes \( \chi^{(c)}_J \) and \( \chi^{(c)}_K \) monotonically non-decreasing functions in \( Y_i \) and \( Y^{(c)}_i \), respectively. Moreover,
\[
E[\log_d(1 + |J_H|)] = E \left[ \log_d \left( 1 + \sum_{c=1}^{d-1} \chi^{(c)}_J \right) \right]
\]
and
\[
E[\log_d(1 + |K_H|)] = E \left[ \log_d \left( 1 + \sum_{c=1}^{d-1} \chi^{(c)}_K \right) \right].
\]
From there, we can repeatedly use Lemma A.9 (from right to left) to identify \( Z_i^{(c)} \) with \( Z_i \) for \( c = 1, 2, \ldots, d - 1 \) (for all \( i \in [r] \) at the same time), using concavity of \( h: x \mapsto \log_d(1 + x) \) (for non-negative \( x \)). This corresponds to replacing \( \chi^{(c)}_K \) one by one by the \( \chi^{(c)}_J \), decreasing the expectation in the process.

The lemma we have just proven gives us a bound on the expected number of trees of height more than \( H \), conditioned on the fact that the root has the place \( p \). Let \( H := \lceil \log_{(k-1)(d-1)} D \rceil - 1 \), and observe that, similarly to the Boolean case, if a critical clause tree has more than \( \frac{D}{d-1} \) reachable nodes, it needs to have a reachable node at depth more than \( H \). Thus the previous lemma and (5.2) gives us
\[
E[\log_d |A(x, F, V, \alpha, \pi, D)| \mid \pi(x) = p] \leq \sum_{j=1}^{d-1} \left( \log_d(1 + j) \binom{d-1}{j} \tilde{R}_{(d,k)}(p)^{d-1-j}(1 - \tilde{R}_{(d,k)}(p))^j \right) + \epsilon^{(H)}_{(d,k)}(p).
\]
(5.3)

Now we want to remove the conditioning on \( p \), i.e. we want to integrate the right-hand side over \( p \) from 0 to 1, to prove Theorem 5.2. We will show

**Lemma 5.13.** For all \( x \in V \),
\[
\lim_{D \to \infty} E_\pi [\log_d(|A(x, F, V, \alpha, \pi, D)|)] \leq S_{(d,k)},
\]
where \( S_{(d,k)} \) is defined in Definition 5.1.
This lemma implies Theorem 5.2 for \( D = \log n \): by (5.1) we relate the set \( A \) to the success probability of PPSZ, and then we use a standard repetition argument (Lemma A.2). Note that a single run of PPSZ still takes only subexponential time if \( d \) and \( k \) are constant.

**Proof.** Start with (5.3). Firstly, we claim that the additional \( \varepsilon_H^{(d,k)}(p) \) integrates to \( o(1) \). This follows from exactly the same machinery as in the Boolean case:

\[
E[\log_d |A(x, F, V, \alpha, \pi, D)| \mid \pi(x) = p] \text{ is monotonically non-increasing in } p \text{ (moving } x \text{ to the end of PPSZ can only make the set } A \text{ smaller). Also } \sum_{j=1}^{d-1} \left( \log_d(1 + j) \binom{d-1}{j} \tilde{R}_{(d,k)}(p)^{d-1-j}(1 - \tilde{R}_{(d,k)}(p))^j \right) \text{ is monotonically non-increasing by how it is obtained in Lemma 5.10.}
\]

Thus, by applying Lemma A.6, we will get

\[
\lim_{D \to \infty} E[\log_d |A(x, F, V, \alpha, \pi, D)|] \leq \int_0^1 \sum_{j=1}^{d-1} \left( \log_d(1 + j) \binom{d-1}{j} \tilde{R}_{(d,k)}(p)^{d-1-j}(1 - \tilde{R}_{(d,k)}(p))^j \right) dp.
\]

Now we will derive the explicit formula for the integral as given in Theorem 5.2. This formula retains the integral and is very complicated, but it can be computed directly by numerical integration.

First we have to deal with \( \tilde{R}_{(d,k)}(p) \). Remember that

\[
\tilde{R}_{(d,k)}(p) := (p + (1 - p)R_{(d,k)}(p))^{(k-1)},
\]

and \( R_{(d,k)} = R_{(d-1)(k-1)+1} \) from Chapter 3. Thus by Lemma 3.14, \( R_{(d,k)}(p) \) is the inverse of \( S_{(d-1)(k-1)+1}(t) \) for \( p \in [0, \frac{(d-1)(k-1)-1}{(d-1)(k-1)}] \), and \( R_{(d,k)}(p) = 1 \) otherwise, where we defined \( S_k(t) = \frac{t^{k-1} - t}{1 - t} \) for \( t \in [0, 1) \) and let \( S_k(1) = \frac{k-2}{k-1} \).

Furthermore, by definition of \( R_{(d,k)} \) (Definition 5.8), we have the identity

\[
R_{(d,k)}(p) := (p + (1 - p)R_{(d,k)}(p))^{(k-1)(d-1)},
\]

and so \( \tilde{R}_{(d,k)}(p) = R_{(d,k)}^{\frac{1}{(d-1)}}(p) \). Thus

\[
\tilde{R}_{(d,k)}^{-1}(s) = R_{(d,k)}^{-1}(s^{d-1}) = \frac{s^{\frac{1}{(d-1)}} - s^{d-1}}{1 - s^{d-1}}.
\]
We now define \( h_j : [0, 1] \to [0, 1] \) for \( j = 1, \ldots, d - 1 \), where \( h_j(t) = t^{d-1-j}(1-t)^j \), and so our integral can be written as
\[
\int_0^1 \tilde{R}(d,k)(p)^{d-1-j}(1 - \tilde{R}(d,k))j dp = \int_0^1 h_j(\tilde{R}(d,k)(p))dp
\]
\[
= \int_0^{(d-1)(k-1)-1} h_j(\tilde{R}(d,k)(p))dp.
\]
Now we use the following classical substitution rule:

**Lemma 5.14.** Let \( f : I \to \mathbb{R} \) be a continuous function and \( \phi : [a,b] \to \mathbb{R} \) a continuous differentiable function where \( \phi([a,b]) \subset I \). Then we have
\[
\int_{\phi(a)}^{\phi(b)} f(x) dx = \int_a^b f(\phi(t))\phi'(t)dt
\]

Setting \( f = h \circ \tilde{R}(d,k) \) and \( \phi = \tilde{R}^{-1}(d,k) \) allows us to conclude that
\[
\int_0^{(d-1)(k-1)-1} h_j(\tilde{R}(d,k)(p))dp = \int_0^1 h_j(s)(\tilde{R}(d,k))'(s)ds,
\]
which yields
\[
\int_0^1 \tilde{R}(d,k)(p)^{d-1-j}(1 - \tilde{R}(d,k)(p))j dp
\]
\[
= \int_0^1 s^{d-1-j}(1-s)^j
\]
\[
(1 - s^{d-1})(\frac{1}{k-1}s^{\frac{k-1}{k-1}}- - (d-1)s^{d-2})+(s^{\frac{k-1}{k-1}} - s^{d-1})(d-1)s^{(d-2)}
\]
\[
(1 - s^{d-1})^2
\]
ds,
and using linearity of integration concludes the proof.

5.2 General \((d, k)\)-ClSP

The analysis of the previous section does not go through in the general case because there is no guarantee that we can build critical clause trees. However, if a given variable \( x \) has the same value in all the satisfying assignments, the argument for that particular variable \( x \) goes through (as in the Boolean case), as all assignments required to be unsatisfiable in the tree construction assign a different value to \( x \). We define:
Definition 5.15. Let $F$ be any satisfiable $(d,k)$-ClSP over $V$ and $x \in V$. $x$ is said to be (completely) frozen in $F$ if all satisfying assignments of $F$ send $x$ to the same value. Otherwise, $x$ is said to be non-frozen.

We partition the variables $V$ into three categories:

$$V = V_{fo}(F,V) \cup V_{fr}(F,V) \cup V_{nf}(F,V)$$

where

- $V_{fo}(F,V)$ is the set of variables that are currently (completely) forced, i.e. that is there exists $c$ such that for every $c' \neq c, c \in [d]$, $F \models_D (x \neq c')$.

- $V_{fr}(F,V)$ is the set of variables that are (completely) frozen, but not (completely) forced, i.e. which have the same value in all satisfying assignments, but there exist at most $(d-2)$ values $c$ such that $F \models_D (x \neq c)$.

- $V_{nf}(F,V)$ is the set of variables that are not frozen, i.e. the ones that can be assigned at least two values while keeping the formula satisfiable.

Note that forced variables are always frozen. For frozen variables, we have the following lemma:

Lemma 5.16. There exists a sequence $\epsilon^{(1)}_{(d,k)}, \epsilon^{(2)}_{(d,k)}, \ldots \in \mathbb{R}_0^+$ of numbers having $\epsilon^{(D)}_{(d,k)} \to 0$ for $D \to \infty$ such that the following holds: For a $(d,k)$-ClSP formula $F$ over $V$, a frozen variable $x \in V$, $\alpha$ a satisfying assignment, we have

$$\mathbb{E}_\pi [\log d (|A(x,F,V,\alpha,\pi,D)|)] \leq S_{(d,k)} + \epsilon^{(D)}_{(d,k)},$$

where $S_{(d,k)}$ is defined in Definition 5.1, and $\pi$ is a random permutation on $V$.

Proof. This lemma follows directly from Lemma 5.13, and the observation that the same bound can also be proven if there are multiple satisfying assignments, but $x$ is frozen. $\square$

In what follows, we will consider a fixed, large $D$, and we will denote $S^{(D)}_{(d,k)} = S_{(d,k)} + \epsilon^{(D)}_{(d,k)}$. We defined, for Theorem 5.3, $G_{(d,k)} = \max\{S_{(d,k)}, 1 - \frac{\log d e}{2}\}$ and we define now similarly $G^{(D)}_{(d,k)} = \max\{S_{(d,k)} +
5.2. General \((d, k)\)-ClSP

\(c^{(D)}_{(d,k)}, 1 - \frac{\log_d e}{2}\). As we have seen at the end of Chapter 4, we relied on the fact that PPSZ is not “too good” for 3-SAT. It turns out that PPSZ is “too good” for \((d, k)\)-ClSP for small \(k\) and large \(d\). The point of the definition of \(G_{(d,k)}\) is to artificially weaken the bound of Lemma 5.16 to make our proof work (unfortunately, this results in a gap between Unique and general \((d,k)\)-ClSP).

For the remainder of this chapter, let \(F\) be a fixed satisfiable \((d,k)\)-ClSP formula over \(n\) variables \(V\). To do the calculation, we will associate a cost to each state of the algorithm as follows. For notational convenience we write (as in the Boolean case) for a partial assignment \(\alpha_0\) on \(V\)

- \(V(\alpha_0)\) as a shorthand for \(V \setminus \text{dom}(\alpha_0)\),
- \(n(\alpha_0) := |V(\alpha_0)|\),
- \(V_{fo}(\alpha_0) := V_{fo}(F^{[\alpha_0]}, V(\alpha_0))\),
- \(V_{fr}(\alpha_0) := V_{fr}(F^{[\alpha_0]}, V(\alpha_0))\),
- \(V_{nf}(\alpha_0) := V_{nf}(F^{[\alpha_0]}, V(\alpha_0))\).

Similarly to Chapter 4, the cost function cost\((\alpha_0)\) is defined such that cost\((\alpha_0) \leq G^{(D)}_{(d,k)} \cdot n(\alpha_0)\) and to goal is to show that probability of success of PPSZ when starting from \(\alpha_0\) is at least \(d^{-\text{cost}(\alpha_0)}\).

**Definition 5.17.** Let \(\alpha_0\) be a partial assignment and \(\alpha\) be a total assignment on \(V\). Let \(x \in V\) be any variable. We define the cost of \(x\) when completing \(\alpha_0\) to \(\alpha\), in writing cost\((\alpha_0, \alpha, x)\) as follows:

- If \(x \neq V(\alpha_0)\), then cost\((\alpha_0, \alpha, x) = 0\).
- If \(\alpha_0\) and \(\alpha\) are incompatible, i.e. \(\exists y \in \text{vbl}(\alpha_0), \alpha_0(y) \neq \alpha(y)\), then cost\((\alpha_0, \alpha, x) = 0\).
- If \(\alpha\) does not satisfy \(F\), then cost\((\alpha_0, \alpha, x) = 0\).
- Else,

\[
\text{cost}(\alpha_0, \alpha, x) =
\begin{cases}
0 & \text{if } x \in V_{fo}(\alpha_0) \\
E_{\pi} [\log_d (|A(x, F^{[\alpha_0]}, V(\alpha_0), \alpha, \pi, D)|)] & \text{if } x \in V_{fr}(\alpha_0) \\
G^{(D)}_{(d,k)} & \text{if } x \in V_{nf}(\alpha_0)
\end{cases}
\]
where $\pi$ is chosen u.a.r. from all permutations on $V(\alpha_0)$. We define the cost of completing $\alpha_0$ to $\alpha$, in writing $\text{cost}(\alpha_0, \alpha)$ by

$$\text{cost}(\alpha_0, \alpha) = \sum_{x \in V(\alpha_0)} \text{cost}(\alpha_0, \alpha, x).$$

Finally, to define the cost of completing $\alpha_0$ to any satisfying assignment, we need the notion of likelihood.

**Definition 5.18.** Let $F[\alpha_0]$ be satisfiable. Let $S(\alpha_0)$ be the set of value assignments $l = (x \mapsto c)$ such that $x \in V(\alpha_0)$ and $F[\alpha_0 \cup \{l\}]$ is satisfiable. For $x \in V(\alpha_0)$, let $S(x, \alpha_0)$ be the set of values $c$ where $F[\alpha_0 \cup \{x \mapsto c\}]$ is satisfiable.

Note that $x$ if frozen in $F[\alpha_0]$ if and only if $S(x, \alpha_0) = 1$.

**Definition 5.19.** Let $\alpha_0$ be a partial assignment on $V$. Suppose $F[\alpha_0]$ is satisfiable.

We define the random process $\text{AssignSL}(F, \alpha_0)$ that produces a (satisfying) assignment on $\text{vbl}(F)$ as follows. Start with the assignment $\alpha_0$, and repeat the following step until $\alpha_0$ is total on $V$: Choose a value assignment $l \in S(\alpha_0)$ uniformly at random and add $l$ to $\alpha_0$. At the end, output $\alpha_0$.

Let $\alpha$ be a total assignment on $\text{vbl}(F)$. Then the likelihood of completing $\alpha_0$ to $\alpha$, in writing $\text{lkhd}(\alpha_0, \alpha)$ is defined as the probability that $\text{AssignSL}(F, \alpha_0)$ returns $\alpha$.

If $F[\alpha_0]$ is not satisfiable, we define $\text{lkhd}(\alpha_0, \alpha) := 0$ for all $\alpha$.

Observe that if $\alpha_0$ and $\alpha$ are incompatible, then $\text{lkhd}(\alpha_0, \alpha) = 0$.

The above definition is different from the Boolean case, in that the next variable is not chosen uniformly. For the Boolean case, both definitions are equivalent, as we can move around forced variables freely (as long as they remain forced), and non-forced Boolean variables have always exactly two viable value assignments. However, for $(d, k)$-ClSP this is no longer the case, and it turns out that the above definition is the correct one, even though it differs from how PPSZ chooses the next variable.

With the likelihood, we finally define

$$\text{cost}(\alpha_0) = \sum_{\alpha \in \text{sat}_V(F)} \text{lkhd}(\alpha_0, \alpha) \cdot \text{cost}(\alpha_0, \alpha).$$
5.2.1 Basic Facts about Cost and Likelihood

We now gather some basic facts about the cost and the likelihood.

**Observation 5.20.** For any $\alpha_0, \alpha, x$, we have $\text{cost}(\alpha_0, \alpha, x) \leq G_{(d,k)}(D)$. Furthermore $\text{cost}(\alpha_0) \leq G_{(d,k)}(D)n(\alpha_0)$.

Next we extend Lemma 4.8 to $(d,k)$-ClSP:

**Definition 5.21.** Let $\alpha_0$ be a partial assignment on $V$, and $x \in V(\alpha_0)$. Let $A(x, \alpha_0)$ be the set of allowed values of $x$ in PPSZ after setting $\alpha_0$, i.e.

$$A(x, \alpha_0) := [d] \setminus \{c \in [d] \mid F^{[\alpha_0]} \models_D (x \neq c)\}.$$  

Note that $A(x, \alpha_0) = 1$ if and only if $x$ is forced in $F^{[\alpha_0]}$.

**Lemma 5.22.** Let $\alpha_0$ be a partial assignment, and $\alpha$ be an assignment on $V$. For any fixed variable $x \in V(\alpha_0)$, if we set $x$ according to $\alpha$,

(i) the likelihood of $\alpha$ can only increase, i.e.

$$\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \geq \text{lkhd}(\alpha_0, \alpha)$$

with equality if $x$ is frozen in $F^{[\alpha_0]}$, and

(ii) the cost of a fixed variable $y \in V$ w.r.t. $\alpha$ can only decrease, i.e.

$$\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y) \leq \text{cost}(\alpha_0, \alpha, y).$$

On the other hand, when choosing $x \in V(\alpha_0)$ uniformly at random (assume $V(\alpha_0)$ is nonempty) and setting it according to $\alpha$,

(iii) the likelihood of $\alpha$ increases on average as

$$\mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)] = \frac{|S(\alpha_0)|}{n(\alpha_0)} \text{lkhd}(\alpha_0, \alpha),$$

whereas

(iv) the cost of a fixed variable $y \in V(\alpha_0)$ decreases on expectation as

$$\mathbb{E}[\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)] \leq \text{cost}(\alpha_0, \alpha, y) - \frac{s}{n(\alpha_0)},$$

where

$$s = \begin{cases} 
\log_d |A(y, \alpha_0)| & \text{if } y \in V_{fr}(\alpha_0) \\
G^{(D)}_{(d,k)} & \text{if } y \in V_{nf}(\alpha_0) \\
0 & \text{if } y \in V_{fo}(\alpha_0). 
\end{cases}$$
Proof. In this proof, and in the remainder of this section, we will often use the notation \( \alpha_0[x \mapsto c] \) instead of \( \alpha_0 \cup \{ x \mapsto c \} \), which is the same if \( x \notin \text{dom}(\alpha_0) \).

(i) We prove the claim formally by induction over the number of variables left unassigned by \( \alpha_0 \) (a similar proof as in the Boolean case might also work but is more complicated). The claim holds trivially if \( \alpha_0 \) is total on \( \mathcal{V} \). Otherwise, we have

\[
\text{lkhd}(\alpha_0, \alpha) = E_{(x \mapsto c) \leftarrow \text{u.a.r.}, \mathcal{S}(\alpha_0)} [\text{lkhd}(\alpha_0[x \mapsto c], \alpha)]
\]

\[
= \sum_{(x' \mapsto c') \in \mathcal{S}(\alpha_0)} \frac{1}{|\mathcal{S}(\alpha_0)|} \text{lkhd}(\alpha_0[x' \mapsto c'], \alpha)
\]

\[
= \sum_{x' \in \mathcal{V}(\alpha_0)} \frac{1}{|\mathcal{S}(\alpha_0)|} \text{lkhd}(\alpha_0[x' \mapsto \alpha(x')], \alpha)
\]

\[
= \frac{1}{|\mathcal{S}(\alpha_0)|} \left( \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha) + \sum_{x' \in \mathcal{V}(\alpha_0) \setminus \{x\}} \text{lkhd}(\alpha_0[x' \mapsto \alpha(x')], \alpha) \right).
\]

We apply the induction hypothesis and we get

\[
\text{lkhd}(\alpha_0, \alpha) \leq \frac{1}{|\mathcal{S}(\alpha_0)|} \left( \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha) + \right.
\]

\[
\sum_{x' \in \mathcal{V}(\alpha_0) \setminus \{x\}} \left( \text{lkhd}(\alpha_0[x' \mapsto \alpha(x')], \alpha) \right) \right)
\]

\[
= \frac{1}{|\mathcal{S}(\alpha_0)|} \left( \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha) + \right.
\]

\[
|\mathcal{S}(\alpha_0[x \mapsto \alpha(x)])| \cdot \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha) \right).
\]

Now observe that, since \( \mathcal{S}(\alpha_0[x \mapsto c]) \not\subseteq \mathcal{S}(\alpha_0) \), we have \( |\mathcal{S}(\alpha_0[x \mapsto c])| \leq |\mathcal{S}(\alpha_0)| - 1 \), and this allows us to conclude that

\[
\text{lkhd}(\alpha_0, \alpha) \leq \text{lkhd}(\alpha_0[x \mapsto \alpha(x)]).
\]
5.2. General $(d,k)$-ClSP

If $x$ is frozen, then both inequalities of this proof are actually equalities (the former by induction, the latter since fixing a frozen variable preserves all satisfying assignments, and thus all elements in $S$ except for the value assignment on $x$), which proves equality in that case.

(ii) We consider the three cases: the variable $y$ is non-frozen, frozen or forced. Note that if $x = y$, the statement holds trivially.

If $y \in V_{nf}(\alpha_0)$, then cost$(\alpha_0, \alpha, y) = G^{(D)}(d,k)$. Since the cost of a variable is always at most $G^{(D)}(d,k)$, the statement holds.

If $y \in V_{fr}(\alpha_0)$ or $y \in V_{fo}(\alpha_0)$, then cost$(\alpha_0, \alpha, y)$ is the expected logarithm of the number of non-forbidden values for $y$ in the remainder of PPSZ (given that variables are chosen according to $\alpha$). If we now fix another variable $x$ to $\alpha(x)$, then this expectation cannot increase, because adding a value assignment cannot allow a value that was already forbidden. So cost$(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y) \leq$ cost$(\alpha_0, \alpha, y)$.

(iii) We have

\[
\text{lkhd}(\alpha_0, \alpha) = \sum_{(x,c) \in S(\alpha_0)} \frac{1}{|S(\alpha_0)|} \text{lkhd}(\alpha_0[x \mapsto c], \alpha)
\]

\[
= \sum_{x \in V(\alpha_0)} \frac{1}{|S(\alpha_0)|} \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha)
\]

\[
= \frac{n(\alpha_0)}{|S(\alpha_0)|} \mathbf{E}_{x \leftarrow \text{u.a.r.} V(\alpha_0)} \left[ \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha) \right],
\]

which proves the statement. Due to the different definition of likelihood, this proof is actually simpler than in the Boolean case.

(iv) For forced and non-frozen variables this is easy to see and analogous to the Boolean case.

For a fixed, frozen, non-forced variable $y$, we have that

\[
\text{cost}(\alpha_0, \alpha, y) = \mathbf{E}_\pi \left[ \log_d(|A(x, F^{[\alpha_0]}, V, \alpha, \pi, D)|) \right].
\]

Let $\pi$ be a random permutation on $V(\alpha_0)$ and let $z$ be the variable that comes first in $\pi$. By the law of total expectation, we have
that

\[
E_{\pi} \left[ \log_d (|A(y, F^{[\alpha_0]}, V, \alpha, \pi, D)|) \right] = \\
E_z \left[ E_{\pi} \left[ \log_d |A(y, F^{[\alpha_0]}, V, \alpha, \pi, D)|) \right] \mid z \text{ first in } \pi \right].
\]

If \( z = y \), then the expression in the expectation is \( \log_d |A(x, \alpha_0)| \).
After that step, the cost of \( y \) is 0, so the overall cost in that case decreases by at least \( \log_d |A(x, \alpha_0)| \). This happens with probability \( \frac{1}{n(\alpha_0)} \), which yields the desired result.

\[ \Box \]

5.2.2 Proving the Bound

Now we will prove Theorem 5.3. We will give a lower bound to the probability that a specific assignment is returned, as in Lemma 4.14. It is also possible to give a bound to the probability that some satisfying assignment is returned directly, as in Lemma 4.12; however, we omit that proof here as it is less general. We will prove the following lemma:

Lemma 5.23. Let \( \alpha_0 \) s.t. \( F^{[\alpha_0]} \) be satisfiable, and \( \alpha \) be a satisfying assignment of \( F \) on \( V \). Then

\[
Pr \left( PPSZ(F^{[\alpha_0]}, V(\alpha_0), D) \cup \alpha_0 = \alpha \right) \geq \text{lkhd}(\alpha_0, \alpha) \cdot d^{-\text{cost}(\alpha_0, \alpha)}.
\]

This lemma with \( D = \log n \) implies Theorem 5.3 by observing that the cost is bounded (Observation 5.20), summing up over all satisfying assignments \( \alpha \), and finally using Lemma A.2 to obtain a randomized algorithm by independent repetition.

Proof. This proof is along the same lines as the proof of Lemma 4.14. In what follows, we consider a fixed satisfying assignment \( \alpha \). We prove the statement by induction over the size of \( \alpha_0 \): we suppose that the claim holds for all \( \alpha_0 \) that fix a larger number of variables. If \( \alpha_0 \) is total on \( V \), then the statement holds trivially. If \( \alpha_0 \) and \( \alpha \) are not compatible then \( \text{lkhd}(\alpha_0, \alpha) = 0 \). Let

\[
p(\alpha_0, \alpha) := Pr \left( PPSZ(F^{[\alpha_0]}, V(\alpha_0), D) \cup \alpha_0 = \alpha \right),
\]

the probability that the output of \( PPSZ(F^{[\alpha_0]}, V(\alpha_0), D) \), together with \( \alpha_0 \), is \( \alpha \). Let \( x \) and \( c \) be random variables: \( x \in V(\alpha_0) \) uniformly at
random, and \( c \) is then picked uniformly at random in \( \mathcal{A}(x, \alpha_0) \), the set of allowed values for \( x \). Observe that \( x \) and \( c \) are chosen exactly as PPSZ would choose the next variable and its value. Thus we have:

\[
p(\alpha_0, \alpha) = \mathbb{E}[p(\alpha_0[x \mapsto c], \alpha)]
\]

\[
= \frac{1}{n(\alpha_0)} \sum_{x' \in V(\alpha_0)} \frac{1}{|\mathcal{A}(x', \alpha_0)|} \sum_{c' \in \mathcal{A}(x', \alpha_0)} p(\alpha_0[x' \mapsto c'], \alpha)
\]

\[
= \frac{1}{n(\alpha_0)} \sum_{x' \in V(\alpha_0)} \frac{1}{|\mathcal{A}(x', \alpha_0)|} p(\alpha_0[x' \mapsto \alpha(x')], \alpha).
\] (5.4)

We could apply the induction hypothesis at this step and continue by applying Jensen’s inequality. However, our induction hypothesis introduces the likelihood in the expectation. The likelihood is not necessarily very concentrated around its expectation, and Jensen’s inequality does not seem to yield any usable bound. To circumvent this problem, we introduce a new distribution over the variables of \( V(\alpha_0) \), weighted according to \( \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha) \): this will allow us to cancel out the likelihood with the one from the induction hypothesis, and the computation will yield the desired bound. We have that

\[
\text{lkhd}(\alpha_0, \alpha) = \mathbb{E}_{(X, C) \in \text{u.a.r. } S(\alpha_0)} \text{lkhd}(\alpha_0[X \mapsto C], \alpha)
\]

\[
= \sum_{X' \in V(\alpha_0)} \frac{1}{|S(\alpha_0)|} \text{lkhd}(\alpha_0[X' \mapsto \alpha(X')], \alpha).
\]

We write as a shorthand \( l(X') := \frac{\text{lkhd}(\alpha_0[X \mapsto \alpha(X')], \alpha)}{\text{lkhd}(\alpha_0, \alpha)} \). With that, the probability of any variable \( \xi' \in V(\alpha_0) \) for the distribution \( \xi \) is given by

\[
p_{\xi}(\xi') = \frac{l(\xi')}{|S(\alpha_0)|}.
\]

Continuing from (5.4), we get

\[
p(\alpha_0, \alpha) = \frac{1}{n(\alpha_0)} \sum_{x' \in V(\alpha_0)} \frac{1}{|\mathcal{A}(x', \alpha_0)|} p(\alpha[x' \mapsto \alpha(x')], \alpha)
\]

\[
= \frac{1}{n(\alpha_0)} \sum_{\xi' \in V(\alpha_0)} \frac{l(\xi')}{|S(\alpha_0)|} \frac{|S(\alpha_0)|}{l(\xi')} \frac{1}{|\mathcal{A}(\xi', \alpha_0)|} p(\alpha[\xi' \mapsto \alpha(\xi')], \alpha)
\]

\[
= \frac{|S(\alpha_0)|}{n(\alpha_0)} \mathbb{E}_{\xi} \left[ \frac{1}{l(\xi')} \frac{1}{|\mathcal{A}(\xi, \alpha_0)|} p(\alpha[\xi \mapsto \alpha(\xi)], \alpha) \right].
\]
Now we apply the induction hypothesis (the new likelihood cancels out) and then Jensen’s inequality with the convex function $x \mapsto d - x$ and get
\[
p(\alpha_0, \alpha) \geq \frac{|S(\alpha_0)|}{n(\alpha_0)} \text{lkh}(\alpha_0, \alpha) \mathbb{E}_\xi \left[ \frac{d^{-\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha)}}{|A(x, \alpha_0)|} \right] \geq \text{lkh}(\alpha_0, \alpha) \cdot d^\log \frac{|S(\alpha_0)|}{n(\alpha_0)} - \mathbb{E}_\xi [\log_d |A(\xi, \alpha_0)|] - \mathbb{E}_\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha)].
\]

To finish the proof, we need to show
\[
\log_d \frac{|S(\alpha_0)|}{n(\alpha_0)} - \mathbb{E}_\xi [\log_d |A(\xi, \alpha_0)|] - \mathbb{E}_\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha)] \geq -\text{cost}(\alpha_0, \alpha). \tag{5.5}
\]

We will now bound each summand on the right-hand side separately.

**Claim 1.**
\[
\log_d \frac{|S(\alpha_0)|}{n(\alpha_0)} \geq \log_d (e) \frac{\sum_{y \in V_{nf}(\alpha_0)}(|S(y, \alpha_0)| - 1)}{|S(\alpha_0)|}.
\]

**Proof.** We have
\[
\log_d \frac{|S(\alpha_0)|}{n(\alpha_0)} = \log_d \left( 1 + \frac{\sum_{y \in V_{nf}(\alpha_0)}(|S(y, \alpha_0)| - 1)}{n(\alpha_0)} \right)
\]
and by using $\log(1+x) \geq \log(e) \frac{x}{1+x}$ (Lemma A.1), the claim follows.

**Claim 2.**
\[
\mathbb{E}_\xi [\log_d |A(\xi, \alpha_0)|] \leq \frac{\sum_{y \in V_{fr}(\alpha_0) \cup V_{nf}(\alpha_0)} \log_d |A(y, \alpha_0)|}{|S(\alpha_0)|} + \frac{\sum_{y \in V_{nf}(\alpha_0)}(|S(y, \alpha_0)| - 1)}{|S(\alpha_0)|}.
\]

**Proof.** By definition of the expectation, we have that for $x \leftarrow_{\text{u.a.r.}} V(\alpha_0)$,
\[
\mathbb{E}_\xi [\log_d |A(\xi, \alpha_0)|] = \sum_{\xi' \in V(\alpha_0)} \frac{\text{lkh}(\alpha_0[\xi' \mapsto \alpha(\xi')], \alpha)}{|S(\alpha_0)| \cdot \text{lkh}(\alpha_0, \alpha)} \log_d |A(\xi, \alpha_0)| = \frac{n(\alpha_0)}{|S(\alpha_0)| \cdot \text{lkh}(\alpha_0)} \mathbb{E}_x [\log_d |A(x, \alpha_0)| \cdot \text{lkh}(\alpha_0[x \mapsto \alpha(x)], \alpha)].
\]
We use the correlation inequality from Lemma 4.9 with $A = \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha)$, $a = \text{lkhd}(\alpha_0, \alpha)$, $\bar{a} = \frac{|S(\alpha_0)|}{n(\alpha_0)} \cdot \text{lkhd}(\alpha_0, \alpha)$, $B = \log_d |A(x, \alpha_0)|$, $b = 1$, $\bar{b} = \sum_{y \in V_{nf}(\alpha_0) \cup V_{fr}(\alpha_0)} \log_d |A(y, \alpha_0)|$. The conditions of the lemma are satisfied by Lemma 5.22(i) and the fact that $|A(x, \alpha_0)| \leq d$. The lemma states that $E[A \cdot B] \leq ab + b\bar{a} - ab$, so we have

$$E[\xi \cdot \log_d |A(\xi, \alpha_0)|] \leq \sum_{y \in V_{nf}(\alpha_0) \cup V_{fr}(\alpha_0)} \frac{\log_d |A(y, \alpha_0)|}{|S(\alpha_0)|} + 1 - \frac{n(\alpha_0)}{|S(\alpha_0)|},$$

and the claim follows. \qed

Claim 3.

$$E[\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha)] \leq \text{cost}(\alpha_0, \alpha) - \frac{G^{(D)}_{(d,k)}}{|S(\alpha_0)|} \sum_{y \in V_{nf}(\alpha_0)} |S(y, \alpha_0)| - \frac{\sum_{y \in V_{fr}(\alpha_0)} \log_d |A(y, \alpha_0)|}{|S(\alpha_0)|}.$$

Proof. We split our expression variable by variable:

$$E[\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha)] = \sum_{y \in V_{fr}(\alpha_0)} E[\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha, y)] + \sum_{y \in V_{nf}(\alpha_0)} E[\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha, y)]. \quad (5.6)$$

For a variable $y \in V_{nf}(\alpha_0)$, if $x = y$, the cost decreases by $G^{(D)}_{(d,k)}$ at least, hence

$$E[\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha, y)] \leq \text{cost}(\alpha_0, \alpha, y) - G^{(D)}_{(d,k)} \cdot \frac{\text{lkhd}(\alpha_0[y \mapsto \alpha(y)], \alpha)}{|S(\alpha_0)|} \cdot \text{lkhd}(\alpha_0, \alpha).$$

For a variable $y \in V_{fr}(\alpha_0)$, we first change the expectation to $x \in$
\( V(\alpha_0) \) u.a.r.

\[
\mathbb{E}_\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha, y)] = \sum_{x' \in V(\alpha_0)} \frac{\text{lkhd}(\alpha_0[x' \mapsto \alpha(x')], \alpha)}{|S(\alpha_0)|} \cdot \text{cost}(\alpha_0[x' \mapsto \alpha(x')], \alpha, y)
\]

\[
= \frac{n(\alpha_0)}{|S(\alpha_0)|} \cdot \text{lkhd}(\alpha_0, \alpha).
\]

We then invoke the correlation inequality Lemma 4.9 with

\( A = \text{lkhd}(\alpha_0[x \mapsto \alpha(x)], \alpha), \ a = \frac{|S(\alpha_0)|}{n(\alpha_0)} \cdot \text{lkhd}(\alpha_0, \alpha), \ B = \text{cost}(\alpha_0[x \mapsto \alpha(x)], \alpha, y), \ b = \text{cost}(\alpha_0, \alpha, y) - \frac{1}{n(\alpha_0)} \log_d |A(y, \alpha_0)|. \)

The conditions of the lemma are satisfied by Lemma 5.22, and so we have \( \mathbb{E} [A \cdot B] \leq ab + ba - ab, \)
and

\[
\mathbb{E}_\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha, y)] \leq \text{cost}(\alpha_0, \alpha, y) - \frac{\log_d |A(y, \alpha_0)|}{|S(\alpha_0)|}.
\]

Plugging in these expressions in expression (5.6) yields that

\[
\mathbb{E}_\xi [\text{cost}(\alpha_0[\xi \mapsto \alpha(\xi)], \alpha, y)] \leq \text{cost}(\alpha_0, \alpha, y) - \sum_{y \in V_{fr}(\alpha_0)} \frac{\log_d |A(y, \alpha_0)|}{|S(\alpha_0)|}
\]

\[
- \sum_{y \in V_{nf}(\alpha_0)} \frac{G^{(D)}_{(d_k)} \cdot \text{lkhd}(\alpha_0[y \mapsto \alpha(y)], \alpha)}{\text{lkhd}(\alpha_0, \alpha) \cdot |S(\alpha_0)|}. \tag{5.7}
\]

To evaluate the likelihood term, from Lemma 5.22(iii) we have that for \( z \in V(\alpha_0) \) u.a.r.

\[
n(\alpha_0) \mathbb{E}_z [\text{lkhd}(\alpha_0[z \mapsto \alpha(z)], \alpha)] = \sum_{z' \in V(\alpha_0)} |S(z', \alpha_0)| \cdot \text{lkhd}(\alpha_0, \alpha).
\]

On the other hand, for frozen variables \( z' \in V_{fr}(\alpha_0) \cup V_{fr}(\alpha_0), \) \( \text{lkhd}(\alpha_0, \alpha) = \text{lkhd}(\alpha_0[z' \mapsto \alpha(z')], \alpha) \) by Lemma 5.22(i). Thus,

\[
n(\alpha_0) \mathbb{E}_z [\text{lkhd}(\alpha_0[z \mapsto \alpha(z)], \alpha)] = \sum_{z' \in V_{fr}(\alpha_0) \cup V_{fr}(\alpha_0)} \text{lkhd}(\alpha_0, \alpha) + \sum_{z' \in V_{nf}(\alpha_0)} \text{lkhd}(\alpha_0[z' \mapsto \alpha(z')], \alpha).
\]
Combining the last two equalities gives
\[
\sum_{z' \in V_{nf}(\alpha_0)} \text{lkhd}(\alpha_0[z' \mapsto \alpha(z')], \alpha) = \sum_{z' \in V_{nf}(\alpha_0)} |S(z', \alpha_0)| \cdot \text{lkhd}(\alpha_0, \alpha),
\]
and inserting this into (5.7) proves the claim. \(\Box\)

Combining the three claims and canceling \(-\text{cost}(\alpha_0, \alpha)\), we have that the following inequality implies (5.5):
\[
\log_d(e) \frac{\sum_{y \in V_{nf}(\alpha_0)} (|S(y, \alpha_0)| - 1)}{|S(\alpha_0)|} - \frac{\sum_{y \in V_{fr}(\alpha_0) \cup V_{nf}(\alpha_0)} \log_d |A(y, \alpha_0)|}{|S(\alpha_0)|} - \frac{\sum_{y \in V_{nf}(\alpha_0)} (|S(y, \alpha_0)| - 1)}{|S(\alpha_0)|} + G_{(d,k)}^{(D)} \sum_{y \in V_{nf}(\alpha_0)} |S(y, \alpha_0)| \geq 0.
\]

For the frozen variables, the corresponding summands cancel out, so equivalently
\[
\log_d(e) \frac{\sum_{y \in V_{nf}(\alpha_0)} (|S(y, \alpha_0)| - 1)}{|S(\alpha_0)|} - \frac{\sum_{y \in V_{nf}(\alpha_0)} \log_d |A(y, \alpha_0)|}{|S(\alpha_0)|} - \frac{\sum_{y \in V_{nf}(\alpha_0)} (|S(y, \alpha_0)| - 1)}{|S(\alpha_0)|} + G_{(d,k)}^{(D)} \sum_{y \in V_{nf}(\alpha_0)} |S(y, \alpha_0)| \geq 0.
\]

Regrouping the terms of the left-hand side gives equivalently
\[
\frac{1}{|S(\alpha_0)|} \sum_{y \in V_{nf}(\alpha_0)} \left( \log_d(e)(|S(y, \alpha_0)| - 1) - \log_d |A(y, \alpha_0)| - |S(y, \alpha_0)| + 1 + G_{(d,k)}^{(D)} |S(y, \alpha_0)| \right) \geq 0.
\]

Now, as \(\log_d |A(y, \alpha_0)| \leq 1\), it suffices to show for each summand
\[
\log_d(e)(|S(y, \alpha_0)| - 1) - (1 - G_{(d,k)}^{(D)}) |S(y, \alpha_0)| \geq 0,
\]
equivalently
\[
\frac{\log_d(e)}{1 - G_{(d,k)}^{(D)}} \geq \frac{|S(y, \alpha_0)|}{|S(y, \alpha_0)| - 1}.
\]
By definition \( G^{(D)}_{(d,k)} \geq 1 - \frac{\log_d e}{2} \) and so \( \frac{\log_d(e)}{1-G^{(D)}_{(d,k)}} \geq 2 \). On the other hand \( \frac{|S(x,\alpha_0)|}{|S(x,\alpha_0)|-1} \leq 2 \), as for non-frozen variables \( |S(x,\alpha_0)| \geq 2 \), and the inequality follows.

Note that the estimate \( |S(x,\alpha_0)| \geq 2 \) is very crude for \( d \geq 3 \). However, to improve the result one would need to show more if the inequality is tight (\( |S(x,\alpha_0)| = 2 \)), i.e. for variables that are almost frozen. Indeed, for such variables one can build all but one critical clause tree. However, we were not able to define an appropriate cost for partially frozen variables that takes transitions from being non-frozen and to being frozen into account. The main problem is that starting to observe a critical clause tree for one value while preserving some progress of a critical clause for another value tree gives rise to correlations that we were not able to deal with. Our correlation inequalities assume that we have the same “progress” (i.e. deletion probability \( p \)) in all critical clause trees of the same variable.

\[ \square \]

### 5.3 Conclusion

We have shown how to apply the PPSZ algorithm to \((d,k)\)-ClSPs. In the unique case we established correlation inequalities showing that PPSZ behaves as expected. This improves the fastest known running time for Unique \((d,k)\)-ClSP algorithm for all tested values except if \( k = 2 \) and \( d = 3, 4 \). However, the actual running time is given by a complicated integral which is very hard to analyze, any we can only compute particular values. For the values we computed, these bounds transfer to the general case, except if \( k = 2 \) or \( k = 3 \) and \( d \geq 6 \). For \( k = 3 \) and \( d \leq 9 \) we still improve on the PPZ algorithm, the previously fastest algorithm. For \( k \geq 4 \), the bounds of the unique case appear to hold in general, but we do not have a formal proof for this.

In our analysis, we only distinguished the cases of frozen variables (where only one value is viable) and variables with at least two viable values. An analysis with a finer distinction should give an improved result for the general case. The hard part to analyze is the transition between different types of nonfrozenness.
Chapter 6

Improvement over PPSZ

In this chapter we will show that the PPSZ algorithm can be improved for 3-SAT. More precisely, we give a randomized algorithm for 3-SAT with exponentially better running time bounds than what could be shown for PPSZ. The improvement is very small, but it shows that PPSZ can be improved and motivates the search for more significant improvements.

To understand the improvement, first suppose that we have \((\leq 3)\)-CNF-formula with a *unique* satisfying assignment \(\alpha\). We call a clause *critical* for a variable \(x\) (w.r.t. a satisfying assignment \(\alpha\)) if exactly one literal is satisfied by \(\alpha\), and that literal is over \(x\). We have seen in Section 3.2.1 that uniqueness of \(\alpha\) implies that every variable has at least one critical clause. A critical clause for \(x\) can be seen as a reason that the value of \(x\) has to be as prescribed by the unique satisfying assignment. In Section 3.2.1 we argued that this gives a lower bound to the probability that a variable is forced. To analyze PPSZ itself a much more elaborate concept, the critical clause tree, was used.

To improve PPSZ, we observe that at the very beginning of PPSZ, almost no variable is forced (while at the end of PPSZ, almost all variables are forced). Our improvement is based on a case distinction, trying to either improve the beginning of PPSZ, or to give another way of deciding satisfiability better than PPSZ. Firstly, we show that if some variable has at least two critical clauses, there is a better bound on the probability that variable is forced compared to PPSZ (and this is most noticeable at the very beginning). It follows that if any linear fraction of variables has at least two critical clauses, PPSZ itself will be expo-
nentially faster on that formula compared to the worst case bounds. On
the other hand, if there is less than some linear fraction of variables with
at least two critical clauses, we will eliminate these by brute force, and
obtain a formula where every variable has exactly one critical clause.
This gives an exponential overhead which can be made arbitrarily small
depending on what we consider “linear”. Thus this overhead can be
compensated by the following exponential improvement we show on the
reduced formulas.

Given such a formula where every variable has exactly one critical
clause, we next consider how many other (non-critical) clauses there
are. If there are few (for a non-trivial definition of few), we use an algo-
rithm by Wahlström [36] that is faster than PPSZ for formulas with few
clauses allover. If there are many non-critical clauses we use the follow-
ing observation: A non-critical clause has two or more satisfied literals
(w.r.t. the unique satisfying assignment). Especially, after removing a
literal of a non-critical 3-clause, the resulting 2-clause is still satisfied,
whereas for a critical 3-clause, when removing a literal uniformly at
random, the unique satisfied literal is removed with probability \( \frac{1}{3} \). We
will exploit this to obtain a linearly sized set of 2-clauses where only a
small fraction is not satisfied, and show how to use this to improve the
very beginning of PPSZ.

The next question is how to extend this to general 3-SAT. It seems
very hard and cumbersome to combine the previous case distinction with
the machinery of Chapter 4. We will instead add another case distinc-
tion, showing that any exponential improvement over PPSZ for Unique
k-SAT can be transferred to a (diminished) exponential improvement
for general k-SAT. Hence combined with the mentioned improvement to
Unique 3-SAT, this gives us the claimed improvement to general 3-SAT.

The approach is as follows: Let \( F \) be an \((\leq k)\)-CNF over \( n \) variables
\( V \). Suppose there exists an assignment \( \alpha' \) on a small set of variables \( W \)
such that \( F[\alpha'] \) has a unique satisfying assignment. To find a satisfying
assignment of \( F \), we can simply try all \((\binom{n}{|W|})2^{|W|}\) such assignments \( \alpha' \),
and use the algorithm for Unique k-SAT on each \( F[\alpha'] \) (this is the same
brute-force approach as before). If \( W \) is a small enough linear fraction of
\( V \), then that additional factor is less than the assumed fixed exponential
improvement for Unique k-SAT over PPSZ.

Otherwise, for some \( \Delta \), for all assignments \( \alpha' \) on \( \Delta n \) variables, \( F[\alpha'] \)
has no unique satisfying assignment. Especially, this means that for
\((\Delta/2)n\) PPSZ-steps, we are guaranteed to have at least \((\Delta/2)n\) non-
frozen variables, as long as we preserve satisfiability, since fixing all
non-frozen variables according to some satisfying assignment makes this satisfying assignment unique. We show that in this case the analysis of Chapter 4 is not tight and can be improved, and thus PPSZ has already an exponentially better probability to find a satisfying assignment of $F$.

Note that because of the many layers of case distinction we employ, the usage of the brute-force method to get rid of small sets of variables, and some cases only improving the very beginning of PPSZ, the total improvement is extremely small. Still, it is the first algorithm for 3-SAT (and Unique 3-SAT) giving exponential improvements over PPSZ for Unique 3-SAT.

**Our Contribution.** We show how to improve PPSZ to give exponentially faster bounds for Unique 3-SAT and for 3-SAT. To show the latter, we show that if there is an algorithm exponentially faster than PPSZ for Unique $k$-SAT, then there is an algorithm exponentially faster than PPSZ for $k$-SAT. (It might be that the original PPSZ algorithm could be made much faster than what it is currently known; exponentially faster is to be understood with respect to the currently known guaranteed worst case performance).

Let $S_3 = 2\ln 2 - 1$ as defined in Definition 3.5, such that PPSZ for 3-SAT runs in time $2^{(S_3 + o(1))n}$ (and $S_k$ be the corresponding constant for $k$-SAT). We prove

**Theorem 6.1.** There exists a randomized algorithm for Unique 3-SAT running in time $2^{(S_3 - \epsilon_3 + o(1))n}$ where $\epsilon_3 = 10^{-25}$.

**Theorem 6.2.** For all $k \geq 3$, if there exists $\epsilon_k > 0$ and a randomized algorithm for Unique $k$-SAT running in time $2^{(S_k - \epsilon_k + o(1))n}$, then there exists $\epsilon_k' > 0$ and a randomized algorithm for $k$-SAT running in time $2^{(S_k - \epsilon_k' + o(1))n}$.

We have the following immediate corollary for 3-SAT (using the exact values from the proof of Theorem 6.2):

**Corollary 6.3.** There exists a randomized algorithm for 3-SAT running in time $2^{(S_3 - \epsilon_3' + o(1))n}$ for $\epsilon_3' = 10^{-56}$.

In the following section we show the theorem for Unique 3-SAT; in Section 6.2 we show the extension to general 3-SAT.
6.1 Unique 3-SAT

Theorem 6.1 is based on [8]. In this section, we will make use of definitions and statements of Chapter 3. We restate the PPSZ algorithm here (Algorithm 5) in a slightly different but equivalent way: We only consider \( D = \log n \), and we use a placement instead of a permutation.

\begin{algorithm}[h]
\caption{PPSZ(CNF formula \( F \))}
\begin{algorithmic}
\State \( V \gets \text{vbl}(F) \)
\State Choose \( \beta \) u.a.r. from all assignments on \( V \)
\State Choose \( \pi \) as a (uniformly) random placement on \( V \)
\State Let \( (x_1, x_2, \ldots, x_{|V|}) \) be the set \( V \) ordered according to increasing \( \pi \)
\State Let \( \gamma \) be a partial assignment on \( V \), initially empty
\For {\( i \gets 1 \) to \( |V| \)}
\If {\( F[^{\gamma}] \models \log n \ x_i \)}
\State \( \gamma(x_i) \leftarrow 1 \ \{ x_i \) is forced to 1\}
\ElsIf {\( F[^{\gamma}] \models \log n \overline{x}_i \)}
\State \( \gamma(x_i) \leftarrow 0 \ \{ x_i \) is forced to 0\}
\Else
\State \( \gamma(x_i) \leftarrow \beta(x_i) \ \{ x_i \) is guessed\}
\EndIf
\EndFor
\State \text{return} \( \gamma \)
\end{algorithmic}
\end{algorithm}

First we will show some corollaries to the statements of Chapter 3.

**Corollary 6.4.** Let \( F \) be an \((\leq 3)\)-CNF with a unique satisfying assignment \( \alpha \) and let \( r \in [0, 1] \). Let \( n := |\text{vbl}(F)| \). There exist a function \( b(r, n) = \min \left\{ 1, \frac{r^2}{(1-r)^2} \right\} - o(1) \) where \( o(1) \) goes to 0 for fixed \( r \) and \( n \to \infty \), such that in PPSZ(\( F \)), conditioned on \( \beta = \alpha \) and \( \pi(x) = r \), \( x \) is forced with probability at least \( b(r, n) \). For every \( n \), \( b(r, n) \) is monotonically non-decreasing in \( r \).

**Proof.** By (3.4) and Lemma 3.10, it follows that \( x \) is forced with probability at least \( R_3(r) - o_n(1) \) (using \( D = \log n \), and that a tree with bounded height has also bounded size). By Lemma 3.14, \( R_3(r) = \min \left\{ 1, \frac{r^2}{(1-r)^2} \right\} \).

As the probability that a variable \( x \) is forced does not decrease with increasing place \( \pi(x) \), \( b(r, n) \) can be chosen to be non-decreasing in \( r \).
We have \( S_3 = \int_0^1 \left( 1 - \min\{1, \frac{r^2}{1-r^2}\} \right) \, dr \) (defined in Definition 3.5, characterized in Lemma 3.14 and 3.19), corresponding to the probability that a variable is guessed. Note that 0.3862 < \( S_3 < 0.3863 \). We now define \( S_3^{[p]} \) where the integral starts from \( p \) instead of 0, which corresponds to the probability that a variable has place at least \( p \) and is guessed.

**Definition 6.5.** Let \( S_3^{[p]} := \int_p^1 \left( 1 - \min\{1, \frac{r^2}{1-r^2}\} \right) \, dr \).

**Observation 6.6.** For \( p \leq \frac{1}{2} \), \( S_3^{[p]} = S_3 - p + \int_0^p \frac{r^2}{1-r^2} \, dr \).

**Corollary 6.7.** Let \( F \) be an \((\leq 3)\)-CNF with a unique satisfying assignment \( \alpha \). Let \( V := \text{vbl}(F) \) and \( n := |\text{vbl}(F)| \). In PPSZ(\( F \)), conditioned on \( \beta = \alpha \), the expected number of guessed variables is at most \( S_3 n + o(n) \).

Furthermore, suppose we pick every variable of \( F \) with probability \( p \), independently, and let \( V_p \) be the resulting set. Then in PPSZ(F\(^{[\alpha | V_p]} \)), conditioned on \( \beta = \alpha\)\( | V \setminus V_p \), the expected number of guessed variables is at most \( S_3^{[p]} n + o(n) \).

**Proof.** The first statement follows directly from Theorem 3.6.

For the second statement, observe that there is a natural correspondence between the placement \( \pi_p \) (on \( V \setminus V_p \)) of PPSZ\( (F^{[\alpha | V_p]} \)) and the placement \( \pi \) (on \( V \)) of PPSZ\( (F) \) restricted to the variables with place at least \( p \) in \( \pi \). In particular, \( \pi\{x \in V | \pi(x) \geq p \} \) has the same distribution as \( \pi_p \): every variable \( x \) has \( \pi(x) \geq p \) with probability \( 1 - p \), and conditioned on that, is uniform in \([p, 1]\) in \( \pi \). Likewise, every variable is in \( V \setminus V_p \) with probability \( 1 - p \), and conditioned on that, uniform in \([0, 1]\) in \( \pi_p \). Note that since \( F \) and hence \( F^{[\alpha | V_p]} \) has a unique satisfying assignment, \( \text{vbl}(F^{[\alpha | V_p]}) = V \setminus V_p \) (this does not hold in general, and in Chapter 4 and Section 6.2 we need to consider the variable set the formula is over separately).

Coupling the placements of PPSZ\( (F^{[\alpha | V_p]} \)) and PPSZ\( (F) \) as specified above, \( x \in V \) is guessed in PPSZ\( (F^{[\alpha | V_p]} \)) conditioned on \( \beta = \alpha | V \setminus V_p \), if and only if \( x \) is guessed in PPSZ\( (F) \) and has place \( \pi(x) \geq p \) conditioned on \( \beta = \alpha \). Thus, by Corollary 6.4, and by Lemma A.6, the statement follows using the law of total probability.

**Corollary 6.8.** Let \( F \) be an \((\leq 3)\)-CNF with a unique satisfying assignment \( \alpha \). Let \( n := |\text{vbl}(F)| \). Suppose we pick every variable of \( F \) with
probability \( p \), independently, and let \( V_p \) be the resulting set. Then

\[
\mathbb{E}_{V_p} \left[ \log \Pr_{\beta,\pi} \left( \text{PPSZ}(F^{[\alpha|_{V_p}]} = \alpha|_{(V \setminus V_p)}) \right) \right] \geq -S_3^{[p]} n - o(n).
\]

**Proof.** This follows from Corollary 6.7 and (3.1), which relates the expectation of the number guessed variables to the probability that PPSZ returns a fixed satisfying assignment. \( \square \)

### 6.1.1 Reducing to One Critical Clause per Variable

Next we show that variables with at least two critical clauses (or smaller critical clauses, we need this later) are guessed with smaller probability than in the worst case. We are considering only 3-SAT here, but the statements in this section can easily be generalized to \( k \)-SAT. Based on this, we show that an exponential improvement for the case where every variable has exactly one critical clause gives an exponential improvement for Unique 3-SAT.

**Definition 6.9.** Let \( F \) be a CNF formula satisfied by \( \alpha \). We call a clause \( C \) critical for \( x \) (w.r.t. \( \alpha \)) if \( \alpha \) satisfies exactly one literal of \( C \), and this literal is over \( x \).

**Definition 6.10.** A 1C-Unique (\( \leq 3 \))-CNF is a uniquely satisfiable (\( \leq 3 \))-CNF where every variable has at most one critical clause. Call the corresponding promise problem (of deciding whether an (\( \leq 3 \))-CNF-formula is unsatisfiable or a 1C-Unique (\( \leq 3 \))-CNF) 1C-Unique 3-SAT.

**Lemma 6.11.** Let \( F \) be an (\( \leq 3 \))-CNF uniquely satisfied by \( \alpha \). Consider PPSZ\((F)\), given that \( \beta = \alpha \). A variable \( x \) with at least two critical clauses (w.r.t. \( \alpha \)) is guessed with probability at most \( S_3 - 0.0014 + o(1) \). Furthermore, a variable \( x \) with a critical (\( \leq 2 \))-clause is guessed with probability at most \( S_3 - 0.035 + o(1) \).

**Proof.** Let \( C \) be some critical clause for \( x \). If all other variables of \( C \) come before \( x \) in PPSZ, then \( x \) is forced (assuming that all previous variables have been chosen according to \( \alpha \); this is ensured by \( \beta = \alpha \), see Observation 3.4). Given \( \pi(x) = r \), the probability for this to happen is \( r^{|C| - 1} \), as every variable \( y \) has \( \pi(x) < r \) with probability \( r \), independently.

Let \( C_1 \) and \( C_2 \) be two critical 3-clauses of \( x \) (if one clause is a (\( \leq 2 \))-clause, we will prove an even better bound). Suppose \( \pi(x) = r \). If \( C_1 \)
and $C_2$ share no variable besides $x$, then the probability that $x$ is forced is at least $2r^2 - r^4$ by the inclusion-exclusion principle. If $C_1$ and $C_2$ share one variable besides $x$, then the probability that $x$ is forced is at least $2r^2 - r^3$ (which is smaller than $2r^2 - r^4$). $C_1$ and $C_2$ cannot share two variables besides $x$: in that case $C_1 = C_2$, as being a critical clause for $x$ w.r.t. $\alpha$ predetermines the polarity of the literals. Intuitively, if $r$ is small, then $2r^2 - r^3$ is almost twice as large as $\frac{r^2}{(1-r)^2}$; in this case the additional clause helps us and the overall forcing probability increases. For a critical ($\leq 2$)-clause the argument is analogous. Here, the probability that $x$ is forced given place $r$ is at least $r$.

From Corollary 6.4, we know that the probability that $x$ is forced is (also) at least $\min \left\{ \frac{r^2}{(1-r)^2}, 1 \right\} - o(1)$. Now we average over $r$ using integrals. The overall probability that $x$ is forced for two critical 3-clauses is at least

$$\int_0^1 \max \left\{ 2r^2 - r^3, \min \left\{ \frac{r^2}{(1-r)^2}, 1 \right\} - o(1) \right\} \, dr \geq 0.6152 - o(1),$$

and for a critical ($\leq 2$)-clause at least

$$\int_0^1 \max \left\{ r, \min \left\{ \frac{r^2}{(1-r)^2}, 1 \right\} - o(1) \right\} \, dr \geq 0.649 - o(1).$$

The $o(1)$ integrates to $o(1)$ by Lemma A.6, as the maximum of two monotone functions is monotone.

This proves the lemma, as $S_3 > 0.3862$, and so $1 - 0.6152 = 0.3848 < S_3 - 0.0014$ and $1 - 0.649 = 0.351 < S_3 - 0.035$. \hfill \square

By (3.1), this gives the following corollary about the success probability:

**Corollary 6.12.** Let $F$ be an ($\leq 3$)-CNF formula uniquely satisfied by $\alpha$. Let $\Delta \in [0, 1]$.

If at least $\Delta n$ variables of $F$ have (at least) two critical clauses, PPSZ finds $\alpha$ with probability at least $2^{-(S-0.0014\Delta + o(1))n}$.

If at least $\Delta n$ variables of $F$ have a critical ($\leq 2$)-clause clause, PPSZ finds $\alpha$ with probability at least $2^{-(S-0.035\Delta + o(1))n}$.

If for some small $\Delta_1$ we have less than $\Delta_1 n$ variables with one critical clause, we find and guess them by brute force, to obtain a 1C-Unique ($\leq 3$)-CNF. If we choose $\Delta_1$ small enough, any exponential improvement
for 1C-Unique 3-SAT gives a (diminished) exponential improvement to Unique 3-SAT.

To bound the number of subsets of size $\Delta_1 n$, we define the binary entropy and use a well-known upper bound to the binomial coefficient (proved in the appendix):

**Lemma A.10.** For $p \in [0, 1]$, define $H(p) := -p \log p - (1-p) \log (1-p)$ (with $0 \log 0 := 0$).

If $pn$ is an integer, then $\binom{n}{pn} \leq 2^{H(p)n}$.

As our main goal just is to give some exponential improvement, we will usually not include actual numbers in the proofs. We use symbolic names instead ($\epsilon_1, \epsilon_3, \Delta_1, \Delta_2, \epsilon_2, p^*$), and argue only about the existence of numbers with the desired properties. The actual numbers leading to the claimed improvement are given by the following table, and are straightforward to check.

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon_1$</td>
<td>$10^{-20}$</td>
<td>improvement in 1C-Unique 3-SAT</td>
</tr>
<tr>
<td>$\epsilon_3$</td>
<td>$10^{-25}$</td>
<td>improvement in Unique 3-SAT</td>
</tr>
<tr>
<td>$\Delta_1$</td>
<td>$10^{-22}$</td>
<td>threshold fraction of vars. with more than 1 crit. clause</td>
</tr>
<tr>
<td>$\Delta_2$</td>
<td>$5 \cdot 10^{-5}$</td>
<td>$\Delta_2 n$ is the amount of variables for $\Delta_2$-sparse and $\Delta_2$-dense</td>
</tr>
<tr>
<td>$\epsilon_2$</td>
<td>$10^{-3}$</td>
<td>exponential savings on repetitions if $F$ is $\Delta_2$-sparse</td>
</tr>
<tr>
<td>$p^*$</td>
<td>$5 \cdot 10^{-7}$</td>
<td>prob. that a var. is assigned using indep. 2-clauses instead of PPSZ</td>
</tr>
</tbody>
</table>

**Lemma 6.13.** If there is a randomized algorithm $\text{ONECC}(F)$ solving 1C-Unique 3-SAT in time $2^{(S-\epsilon_1+o(1))n}$ for some $\epsilon_1 > 0$, then there is a randomized algorithm ($\text{PPSZIMPROVED}$, listed in Algorithm 6) solving Unique 3-SAT in time $2^{(S-\epsilon_3+o(1))n}$ for some $\epsilon_3 > 0$.

**Proof.** Let $F$ be an $(\leq 3)$-CNF uniquely satisfied by $\alpha$. Let $c(F)$ be the number of variables of $F$ with at least two critical clauses. If $c(F) \geq \Delta_1 n$, PPSZ itself is already faster by Corollary 6.12. If $c(F) = 0$, we can use $\text{ONECC}(F)$.

However, what if $0 < c(F) < \Delta_1 n$? In that case, we get rid of these variables by brute force: For all $|\Delta_1 n|$-subsets $W$ of variables and for all $2^{|\Delta_1 n|}$ possible assignments $\alpha'$ on $W$, we try $\text{ONECC}(F^{[\alpha']})$. For (at least) one such $\alpha'$, we have $F^{[\alpha']}$ satisfiable and $c(F^{[\alpha']}) = 0$;
6.1. Unique 3-SAT

namely if $W$ includes all variables with multiple critical clauses and $\alpha'$ is compatible with $\alpha$. This is because fixing variables according to $\alpha$ does not produce new critical clauses w.r.t. $\alpha$.

There are $\binom{n}{\lfloor \Delta_1 n \rfloor}$ subsets of size $\lfloor \Delta_1 n \rfloor$ of the variables of $F$, each with $2^{\lfloor \Delta_1 n \rfloor}$ possible assignments. As $\binom{n}{\lfloor \Delta_1 n \rfloor} \leq 2^{H(\Delta_1) n}$ (Lemma A.10), we invoke ONECC($F^{[\alpha']}$) at most $2^{(\Delta_1 + H(\Delta_1)) n}$ times. Setting $\Delta_1$ small enough such that $\Delta_1 + H(\Delta_1) < \epsilon_1$ retains an exponential improvement for Unique 3-SAT (note that we do not use that the formula of ONECC has less than $n$ variables).

\begin{algorithm}
\caption{PPSZImproved(CNF formula $F$)}
\end{algorithm}


6.1.2 Using One Critical Clause per Variable

In this section we give an exponential improvement for 1C-Unique 3-SAT. Here we need that the clauses have length at most 3. We prove the following theorem:

**Theorem 6.14.** Given a 1C-Unique ($\leq 3$)-CNF $F$ with $n := |\text{vbl}(F)|$, ONECC($F$) runs in expected time $2^{(S-\epsilon_1 + o(1)) n}$ and finds the satisfying assignment with probability $2^{-o(n)}$.

**Corollary 6.15.** There exists a randomized algorithm for 1C-Unique 3-SAT running in time $2^{(S-\epsilon_1 + o(1)) n}$.

Together with Lemma 6.13 this immediately implies Theorem 6.1.

**Proof.** Like in the proof of Lemma A.2, we first repeat ONECC($F$) independently $2^{o(n)}$ times to obtain an algorithm ONECC'($F$) that finds the satisfying assignment with probability at least $\frac{3}{4}$, running in expected time $2^{(S-\epsilon_1 + o(1)) n}$ (for a different $o(1)$). By Markov’s inequality, 12 times the expected running time is exceeded with probability at most $\frac{1}{12}$. Hence if we abort ONECC' after 12 times its expected time, with probability at least $\frac{3}{4} - \frac{1}{12} = \frac{2}{3}$ the satisfying assignment is found. \qed
Chapter 6. Improvement over PPSZ

The core of the improvement is a case distinction into sparse and dense formulas, defined as follows:

**Definition 6.16.** We say a clause $C$ contains variable $x$ if $x \in \text{vbl}(C)$. For a CNF formula $F$ and a variable $x$, the degree of $x$ in $F$, $\deg(F, x)$ is defined to be the number of clauses in $F$ that contain the variable $x$. The 3-clause degree of $x$ in $F$, $\deg_3(F, x)$ is defined to be the number of 3-clauses in $F$ that contain the variable $x$.

For a set of variables $W$, denote by $F \setminus W$ the part of $F$ independent of $W$, defined as the set of clauses of $F$ that do not contain variables of $W$ (formally $F \setminus W = \{ C \in F \mid \text{vbl}(C) \cap W = \emptyset \}$).

For $\Delta \in [0, 1]$, we say that $F$ is $\Delta$-sparse if there exists a set $W$ of at most $\Delta n$ variables such that $F \setminus W$ has maximum 3-clause degree 4. We say that $F$ is $\Delta$-dense otherwise.

We will show that for $\Delta_2$ small enough, we get an improvement for $\Delta_2$-sparse 1C-Unique ($\leq 3$)-CNF formulas. On the other hand, for any $\Delta_2$ we will get an improvement for $\Delta_2$-dense 1C-Unique ($\leq 3$)-CNF formulas. In the sparse case we can fix by brute force a small set of variables to obtain a formula with few 3-clauses. We need to deal with the ($\leq 2$)-clauses and then use an algorithm from Wahlström [36] for CNF formulas with few clauses.

**Algorithm 7** ONECC((\leq 3)-CNF $F$)

\begin{verbatim}
try DENSE($F$)
try SPARSE($F$)
\end{verbatim}

**Dense Case.** First we show the improvement for any $\Delta_2$-dense 1C-Unique ($\leq 3$)-CNF. $\Delta_2$-density means that for any set of $\Delta_2 n$ variables, even after ignoring all clauses containing a variable in the set, a variable with 3-clause degree of at least 5 remains. The crucial idea is that for a variable $x$ with 3-clause degree of at least 5, picking one occurrence of $x$ u.a.r. and removing it gives a 2-clause satisfied (by the unique satisfying assignment) with probability at least $\frac{4}{5}$. This is because the only way a non-satisfied 2-clause can arise is if the 3-clause the literal over $x$ was deleted from was critical for $x$. However, we assumed that there is at most one critical clause for $x$, so for all other occurrences of $x$, the resulting 2-clause remains satisfied.
6.1. Unique 3-SAT

**Algorithm 8 GetInd2Clauses(\((\leq 3)\)\)-CNF)**

\{for the analysis, \(F\) is considered to be \(\Delta_2\)-dense; the procedure might fail otherwise\}

\(n \leftarrow |\text{vbl}(F)|\)

\(F_3 \leftarrow \{C \in F \mid |C| = 3\}, F_2 \leftarrow \{}\)

**for** \([\Delta_2 n] \text{ times}\) **do**

let \(x\) be a variable with \(\deg_3(F_3, x) \geq 5\) (return failure if no such variable exists)

Choose \(C\) u.a.r. from all clauses \(C \in F_3\) with \(x \in \text{vbl}(C)\).

\(l \leftarrow\) the literal of \(C\) over \(x\); \(C_2 \leftarrow C \setminus l\)

\(F_2 \leftarrow F_2 \cup C_2\)

\{remove all clauses of \(F_3\) sharing variables with \(C_2\)\}

\(F_3 \leftarrow \{C_3 \in F_3 \mid \text{vbl}(C_3) \cap \text{vbl}(C_2) = \emptyset\}\)

**end for**

return \(F_2\)

---

**Algorithm 9 Dense(\((\leq 3)\)\)-CNF \(F\))**

\(F_2 \leftarrow \text{GetInd2Clauses}(F)\)

**for** \(2^{(S - \epsilon_1)n}\) **times** **do**

\(V_p^* \leftarrow\) pick each \(x \in \text{vbl}(F)\) with probability \(p^*\)

\(\alpha' \leftarrow \{}\)

**for** \(C_2 \in F_2\) **do**

if \(\text{vbl}(C_2) \subseteq V_p\) **then**

Let \(\{u, v\} = C_2\)

\((\alpha'(u), \alpha'(v)) \leftarrow \begin{cases} (0, 0), & \text{with probability } \frac{3}{15} \\ (0, 1), & \text{with probability } \frac{4}{15} \\ (1, 0), & \text{with probability } \frac{4}{15} \\ (1, 1), & \text{with probability } \frac{4}{15} \end{cases}\)

**end if**

**end for**

**for all** \(x \in V_p, \) if \(\alpha'(x)\) is not defined yet let \(\alpha'(x) \leftarrow \text{u.a.r.} \{0, 1\}\)

PPSZ\((F[\alpha'])\); if a satisfying assignment \(\alpha\) has been found, return \(\alpha \cup \alpha'\)

**end for**
Ignoring all 3-clauses that share variables with the produced 2-clause and repeating the above process gives 2-clause whose variables are disjoint. Also each produced 2-clause is satisfied with probability $\frac{4}{5}$, irrespective of whether the previously produced 2-clauses are satisfied or not. $\Delta_2$-density implies that we can repeat this process $\lceil \frac{1}{2} \Delta_2 n \rceil$ times (as listed in GetInd2Clauses($F$), Algorithm 8) without running out of variables with degree at least 5 in the 3-clauses.

Observation 6.17. For a $\Delta_2$-dense 1C-Unique ($\leq 3$)-CNF $F$, GetInd2Clauses($F$) returns a set of $\lfloor \frac{1}{2} \Delta_2 n \rfloor$ independent 2-clauses. For all $1 \leq i \leq \lfloor \frac{1}{2} \Delta_2 n \rfloor$, the $i$-th returned 2-clause is satisfied (by the unique satisfying assignment of $F$) with probability at least $\frac{4}{5}$, independent of whether the other 2-clauses are satisfied.

As a random 2-clause is satisfied with probability $\frac{3}{4}$ by a specific assignment, this set of 2-clauses gives us nontrivial information about the unique satisfying assignment. Now we show how to use these 2-clauses to improve PPSZ:

Lemma 6.18. For every $\Delta_2 > 0$, there exists $\epsilon_1 > 0$ and an algorithm (Dense($F$), Algorithm 9), that when given a $\Delta_2$-dense 1C-Unique ($\leq 3$)-CNF $F$, runs in time $2^{(S - \epsilon_1 + o(1)) n}$ and returns the satisfying assignment $\alpha$ with probability $2^{-o(n)}$.

Proof. First we give some intuition. For variables that occur late in PPSZ, the probability of being forced is large (being almost 1 in the second half). However, for variables that come at the beginning, the probability is very small; a variable $x$ at place $p$ is forced (in the worst case) with probability $\Theta(p^2)$ for $p \to 0$, hence we expect $\Theta(p^3 n)$ forced variables among the first $pn$ variables in total.

However, a 2-clause that is satisfied by $\alpha$ with probability $\frac{4}{5}$ can be used to guess both variables in a better way than uniform, giving constant savings in the random bits required. For $\Theta(n)$ such 2-clauses, we expect $\Theta(p^2 n)$ of them to have both variables among the first $pn$ variables. For each such 2-clause we have some nontrivial information; intuitively we save around 0.01 bits. In total we save $\Theta(p^2 n)$ bits among the first $pn$ variables, which is better than PPSZ for small enough $p$.

Formally, let $V_{p^*}$ be a random set of variables, where each variable of $V$ is added to $V_{p^*}$ with probability $p^*$, independently. On $V_{p^*}$, we replace PPSZ by our improved guessing; on the remaining variables $V \setminus V_{p^*}$ we run PPSZ as usual. Let $E_{\text{guess}}$ be the event that the improved guessing on $V_{p^*}$ (to be defined later) finds $\alpha|_{V_{p^*}}$. Let $E_{\text{PPSZ}}$ be the event that
PPSZ($F^{[\alpha|V_{p^*}]})$ finds $\alpha|V\setminus V_{p^*}$. Observe that for a fixed $V_{p^*}$, $E_{\text{guess}}$ and $E_{\text{PPSZ}}$ are independent. Hence we can write the overall probability to find $\alpha$ (call it $p_s$) as an expectation over $V_{p^*}$:

$$p_s = E_{V_{p^*}}[\Pr(E_{\text{guess}} \cap E_{\text{PPSZ}}|V_{p^*})]$$

$$= E_{V_{p^*}}[\Pr(E_{\text{guess}}|V_{p^*}) \Pr(E_{\text{PPSZ}}|V_{p^*})]$$

$$\geq 2^{E_{V_{p^*}}[\log \Pr(E_{\text{guess}}|V_{p^*})+\log \Pr(E_{\text{PPSZ}}|V_{p^*})]}$$

$$= 2^{E_{V_{p^*}}[\log \Pr(E_{\text{guess}}|V_{p^*})]+E_{V_{p^*}}[\log \Pr(E_{\text{PPSZ}}|V_{p^*})]},$$

where in the last two steps we used Jensen’s inequality and linearity of expectation.

By Corollary 6.8, $E_{V_{p^*}}[\log \Pr(E_{\text{PPSZ}}|V_{p^*})] = (-S_3^{[p]} - o(1))n$. We now define the guessing and analyze $E_{V_{p^*}}[\log \Pr(E_{\text{guess}}|V_{p^*})]$ (see Algorithm 9 as a reference):

By Observation 6.17 we obtain a set of $\lceil \frac{1}{2} \Delta_2 n \rceil$ independent 2-clauses $F_2$, each satisfied (by $\alpha$) independently with probability $\frac{4}{5}$. In the following we assume that $F_2$ has at least a $\frac{4}{5}$-fraction of satisfied 2-clauses as this happens with constant probability (for a proof, see e.g. [5]) and we only need to show subexponential success probability.

Using the set of 2-clauses $F_2$, we choose an assignment $\alpha'$ on $V_{p^*}$ as follows: For every clause $C_2$ in $F_2$ completely over $V_{p^*}$, choose an assignment on both of its variables: with probability $\frac{1}{5}$ such that $C_2$ is violated, and, with probability $\frac{4}{15}$ each, one of the three assignments that satisfy $C_2$. Afterwards, guess any remaining variable of $V_{p^*}$ u.a.r. from $\{0, 1\}$.

Given $V_{p^*}$, let $m_0$ be the number of clauses of $F_2$ completely over $V_{p^*}$, not satisfied by $\alpha$. Let $m_1$ be the number of clauses of $F_2$ completely over $V_{p^*}$ satisfied by $\alpha$. Then

$$\Pr(E_{\text{guess}}|V_{p^*}) = \left(\frac{1}{2}\right)^{|V_{p^*}|} - 2m_0 - 2m_1 \left(\frac{1}{5}\right)^{m_0} \left(\frac{4}{15}\right)^{m_1}.$$
set both variables according to \( \alpha \) with probability \( \frac{1}{5} \). For any clause satisfied by \( \alpha \), we set both variables according to \( \alpha \) with probability \( \frac{4}{15} \), as we have to pick the right one of the three assignments that satisfy \( C_2 \). As \( E[|V_p^*|] = p^*n \), \( E[m_0] \leq \frac{1}{5}p^*2\lceil \frac{1}{2}\Delta_2 n \rceil \), \( E[m_1] \geq \frac{4}{5}p^*2\lceil \frac{1}{2}\Delta_2 n \rceil \), \( E[m_0 + m_1] = p^*2\lceil \frac{1}{2}\Delta_2 n \rceil \), we have

\[
E[\log \Pr(E_{\text{guess}}|V_p^*)] = -E[V_p^* - 2m_0 - 2m_1] + \log \left( \frac{1}{5} \right) E[m_0] + \log \left( \frac{4}{15} \right) E[m_1] \\
\geq -p^*n \\
+ p^*2 \left\lceil \frac{1}{2}\Delta_2 n \right\rceil \left( 2 + \log \left( \frac{1}{5} \right) \frac{1}{5} + \log \left( \frac{4}{15} \right) \frac{4}{5} \right).
\]

The inequality follows from the observations and \( \log \left( \frac{4}{15} \right) \geq \log \left( \frac{1}{5} \right) \). One can calculate \( 2 + \log \left( \frac{1}{5} \right) \frac{1}{5} + \log \left( \frac{4}{15} \right) \frac{4}{5} \geq 0.01 \). This corresponds to the fact that a four-valued random variable where one value occurs with probability at most \( \frac{1}{5} \) has entropy at most 1.99.

Hence by our calculations and Observation 6.6 (to evaluate \( S_{3}^{(p)} \)), we have

\[
\frac{1}{n} \log p_s \geq -S_3 + p^* - \int_0^{p^*} \frac{r^2}{(1-r)^2} dr - o(1) - p^* + \frac{1}{2}\Delta_2 p^*2 \cdot 0.01 \\
= -S_3 - \int_0^{p^*} \frac{r^2}{(1-r)^2} dr + \frac{1}{2}\Delta_2 p^*2 \cdot 0.01 - o(1).
\]

This gives an improvement over PPSZ of \(- \int_0^{p^*} \frac{r^2}{(1-r)^2} dr + \frac{1}{2}\Delta_2 p^*2 \cdot 0.01 \). The first term corresponds to the savings PPSZ would give us on \( V_p^* \), and the second term corresponds to the savings we have in our modified guessing. Observe that for small \( p^* \), the integral evaluates to \( \Theta(p^*3) \), but the second term is \( \Theta(p^*2) \). Hence choosing \( p^* \) small enough gives an improvement. \( \square \)

**Sparse Case.** Now we show that if \( \Delta_2 > 0 \) is small enough we get an improvement for a \( \Delta_2 \)-sparse 1C-Unique (\( \leq 3 \))-CNF. For this, we need the following theorem by Wahlström:

**Theorem 6.19 ([36]).** Let \( F \) be a CNF formula with average degree at most 4.2 where we count degree 1 as 2 instead. Then satisfiability of \( F \) can be decided in time \( O(2^{0.371n}) \leq 2^{(S-0.015+o(1))n} \). Denote this algorithm by \( \text{Wahlstroem}(F) \).
Algorithm 10 Sparse((≤ 3)-CNF $F$)

$n \leftarrow |\text{vbl}(F)|$

repeat the following $2^{(S-\varepsilon_2)n}$ times:

for all $W \subseteq \text{vbl}(F)$ of size $\lfloor \Delta_2 n \rfloor$ and all assignments $\alpha'$ on $W$ do

$F' \leftarrow F[\alpha']$

while no satisfying assignment found do

try PPSZ($F[\alpha']$)

$F'_2 \leftarrow \{C \in F' \mid |C| \leq 2\}$

if $|F'_2| \leq \frac{1}{10}|\text{vbl}(F')|$ then

with probability $2^{-(S-0.015)|\text{vbl}(F')|}$, run Wahlstroem($F'$)

return failure if no satisfying assignment found so far

end if

{set all literals in a uniform (≤ 2)-clause to 1}

$C' \leftarrow \text{u.a.r. } F'_2$

for $l \in C'$ do

$F' \leftarrow F'[l \rightarrow 1]$

end for

end while

end for

Lemma 6.20. For $\Delta_2$ small enough, there exists $\varepsilon_2 > 0$ such there exists an algorithm (Sparse($F$), Algorithm 10), that given a $\Delta_2$-sparse 1C-Unique (≤ 3)-CNF $F$, runs in expected time $2^{(S-\varepsilon_2+o(1))n}$ and finds the satisfying assignment $\alpha$ of $F$ with probability $2^{-o(n)}$.

Proof. Similar to Section 6.1.1, we first check by brute force all subsets $W$ of $\lfloor \Delta_2 n \rfloor$ variables and all possible assignments $\alpha'$ of $W$; by definition of $\Delta_2$-sparse for some $W$, the part of $F$ independent of $W$ (i.e. $F \setminus W$) has maximum 3-clause degree 4. If furthermore $\alpha'$ is compatible with $\alpha$, $F' := F[\alpha']$ is a 1C-Unique (≤ 3)-CNF with maximum 3-clause degree 4: We observed that critical clauses cannot appear in the process of assigning variables according to $\alpha$; furthermore any clause of $F$ not independent of $W$ must either disappear in $F'$ or become a (≤ 2)-clause. As earlier, there are at most $2^{(\Delta_2+H(\Delta_2))n}$ cases of choosing $W$ and $\alpha'$. We now analyze what happens for the correct choice of $F'$:

We would like to use Wahlstroem on $F'$; however, $F'$ might contain an arbitrary amount of (≤ 2)-clauses. The plan is to use the fact that either there are many critical (≤ 2)-clauses, in which case PPSZ is better than in the worst case, or few critical (≤ 2)-clauses, in which case
all other \((\leq 2)\)-clauses are non-critical and have only satisfied literals.

The algorithm works as follows: We have a 1C-Unique \((\leq 3)\)-CNF on \(F'\) on \(n' := |\text{vbl}(F')|\) variables; the maximum degree in the 3-clauses is at most 4. First we try PPSZ: if there are \(\frac{1}{30} n'\) critical \((\leq 2)\)-clauses, this gives a satisfying assignment with probability \(2^{-(S-0.035\frac{1}{30}+o(1))n'}\) by Corollary 6.12 (every variable has at most one critical clause, so every critical \((\leq 2)\)-clause must be critical for a different variable). Otherwise, if there are less than \(\frac{1}{10} n'\) \((\leq 2)\)-clauses in total, the criterion of Theorem 6.19 applies: We invoke \textsc{Wahlstroem}(\(F'\)) with probability \(2^{-S-0.015\cdot n'}\); this runs in expected time \(2^{-o(n)}\) and finds a satisfying assignment with probability \(2^{-(S-0.015)n'}\).

If both approaches fail, we know that \(F'\) has less than \(\frac{1}{30} n'\) critical \((\leq 2)\)-clauses clauses, but also more than \(\frac{1}{10} n'\) \((\leq 2)\)-clauses overall. Hence at most one third of the \((\leq 2)\)-clauses is critical. However, a non-critical \((\leq 2)\)-clause must be a 2-clause with both literals satisfied. Hence choosing a \((\leq 2)\)-clause of \(F'\) uniformly at random and setting all its literals to 1 sets two variables correctly with probability at least \(2^3 > 2^{0.371} > 2^{-(S-0.015)}\). That is, we reduce the number of variables by 2 with a better probability than PPSZ overall. We then repeat the above process with the reduced formula. This shows that for the correct \(F'\), we have expected running time \(2^{o(n)}\) and success probability \(2^{(-S+\epsilon_2-o(1))n}\) for some \(\epsilon_2 > 0\). It is important that \(\epsilon_2\) does not depend on \(\Delta_2\). Repeating this process \(2^{(-S+\epsilon_2-o(1))n}\) times gives success probability \(2^{o(n)}\).

Together with the brute-force choice of \(W\) and \(\alpha'\), we have expected running time of \(2^{(S-\epsilon_2+\Delta_2+H(\Delta_2)+o(1))n}\). By choosing \(\Delta_2\) small enough (depending on \(\epsilon_2\)) we are better than PPSZ.

\[\square\]

### 6.2 From Unique \(k\)-SAT to General \(k\)-SAT

The goal of this section is to show Theorem 6.2, which we restate here:

**Theorem 6.2.** For all \(k \geq 3\), if there exists \(\epsilon_k > 0\) and a randomized algorithm for Unique \(k\)-SAT running in time \(2^{(S_k-\epsilon_k+o(1))n}\), then there exists \(\epsilon'_k > 0\) and a randomized algorithm for \(k\)-SAT running in time \(2^{(S_k-\epsilon'_k+o(1))n}\).

In this section we use the notation of Chapter 4, and the version of PPSZ defined in Chapter 3 in Algorithm 1 and 2. Let \(F\) be a satisfiable
(≤ k)-CNF over n variables V. We distinguish two cases, whether F is close to having a unique satisfying assignment or not:

**Definition 6.21.** Let Δ ∈ [0, 1]. We say that F is Δ-unique if there exists a set of at most Δn variables W, and an assignment α′ on W such that F[α′] has a unique satisfying assignment.

Otherwise, we say that F is Δ-general.

Let A be the provided algorithm for Unique k-SAT. It is not hard to see that (1) by stopping A appropriately it can be made to run in time \(2^{(S_k - \epsilon_k + o(1))n}\) for all input formulas (even these with many satisfying assignments), and (2) we can transform A into an algorithm that actually gives the unique satisfying assignment with probability at least \(\frac{2}{3}\) (e.g. by checking \(F[x \mapsto 0]\) and \(F[x \mapsto 1]\) independently \(O(\log n)\) times for every \(x \in V\)). The algorithms we considered so far have both these properties already.

If F is Δ-unique, we can use A by using the same brute-force approach as in the previous section: Choose all possible assignments \(\alpha'\) on all possible subsets of \([\Delta n]\) variables W; in total there are at most \(2^{(\Delta + H(\Delta))n}\) choices by Lemma A.10 (where \(H\) is the binary entropy function). For each such \(\alpha'\), apply the algorithm A for Unique k-SAT to \(F[\alpha']\). By Δ-uniqueness, we know that for some \(\alpha'\), \(F[\alpha']\) is uniquely satisfiable, and in this case A must find the unique satisfying assignment of \(F[\alpha']\) with probability at least \(\frac{2}{3}\), and in this case our algorithm also finds a satisfying assignment of F. If \(\Delta + H(\Delta) < \epsilon_k\), we retain an improvement over PPSZ in this case.

**Observation 6.22.** If F is Δ-unique, a satisfying assignment of F can be found in time \(2^{(S_k - \epsilon_k + \Delta + H(\Delta) + o(1))n}\).

The more involved case is if F is Δ-general. We show that for any \(\Delta > 0\), PPSZ finds a satisfying assignment of F with probability exponentially better than the worst-case bounds of PPSZ. We do this by modifying the proof of Lemma 4.12 to show the following lemma:

**Lemma 6.23.** Let F be a satisfiable (≤ k)-CNF over n variables V, Δ-general for some Δ. For \(\epsilon' := \frac{\Delta}{10 + 5\Delta}\), the following holds:

Let \(\alpha_0\) s.t. \(F[^{\alpha_0}]\) is satisfiable, with \(\text{dom}(\alpha_0) \leq \frac{1}{2}\Delta n\). The overall probability of PPSZ(\(F[^{\alpha_0}], V(\alpha_0), D\)) to output some satisfying assignment of \(F[^{\alpha_0}]\) is at least \(2^{-\text{cost}(\alpha_0) - \epsilon'\left(\frac{1}{2}\Delta n - \text{dom}(\alpha_0)\right)}\).

By setting \(\alpha_0 = \{\}\) and using independent repetition, we get the following corollary:
Corollary 6.24. If $F$ is $\Delta$-general, a satisfying assignment of $F$ can be found in time $2^{(S_k - \epsilon + o(1))n}$, for $\epsilon = \frac{\Delta^2}{20 + 10\Delta}$.

Observation 6.22 and Corollary 6.24 together immediately imply Theorem 6.2. For $\Delta = 10^{-27}$, we have $\Delta + H(\Delta) \leq 9.3 \cdot 10^{-26}$, so if $F$ is $\Delta$-unique, and $\epsilon_3 = 10^{-25}$ (the improvement for Unique 3-SAT from Theorem 6.1) we retain an improvement of $10^{-25} - 9.3 \cdot 10^{-26} = 7 \cdot 10^{-27}$. If $F$ is $\Delta$-general, the above argument gives us an improvement of $\frac{\Delta^2}{20 + 10\Delta} \geq 10^{-56}$.

Proof. The idea of the proof is that the last inequality of the proof of Lemma 4.12 is not tight if a certain amount of non-frozen variables is guaranteed. We proceed analogously to the proof of Lemma 4.12, but using a slightly smaller cost

$$\text{cost}'(\alpha_0) := \max \left\{ 0, \text{cost} - \epsilon' \left( \frac{1}{2} \Delta n - \text{dom}(\alpha_0) \right) \right\}$$

instead of $\text{cost}(\alpha_0)$; for $\text{dom}(\alpha_0) > \frac{1}{2} \Delta n$, we let $\text{cost}'(\alpha_0) := \text{cost}(\alpha_0)$.

We proceed again by induction on $n - \text{dom}(\alpha_0)$, i.e. on the number of variables not assigned by $\alpha_0$. In the proof of Lemma 4.12, we are interested only in how the cost changes when setting one more variable in $\alpha_0$. The only difference here is that for the first $\frac{1}{2} \Delta n$ variables, the decrease of $\text{cost}'$ is smaller by $\epsilon'$ compared to $\text{cost}$ (unlike $\text{cost}$, $\text{cost}'$ might even increase when fixing more variables).

We follow the proof of Lemma 4.12 with the modified cost to the point where we apply the induction hypothesis and Lemma 4.11 (up to this point, the cost is not used in the proof). We have for $\text{dom}(\alpha_0) \leq \frac{1}{2} \Delta n$

$$\mathbb{E} \left[ -\log(p(\alpha_0 \cup \{X \mapsto B}\}) \right] \leq$$

$$\leq \mathbb{E} \left[ \text{cost}'(\alpha_0 \cup \{X \mapsto B}\}) \right]$$

$$\leq \text{cost}'(\alpha_0) - \frac{|V_{fr}(\alpha_0)|}{|S|} \cdot |S| - \frac{2S_k^{(D)} \cdot |V_{nf}(\alpha_0)|}{|S|} + \epsilon',$$

and proceeding as in the proof of Lemma 4.12 we have to show that

$$\log \left( \frac{|S|}{2n(\alpha_0)} \right) + \frac{|V_{fo}(\alpha_0)|}{|S|} + \frac{|V_{fr}(\alpha_0)|}{|S|} + 2S_k^{(D)} \cdot |V_{nf}(\alpha_0)| \geq \epsilon',$$

where in Lemma 4.12 we just had to show that the left-hand side is at least 0.
Proceeding further as in the proof to Lemma 4.12, we obtain that it is sufficient to show that

$$-|S| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)| + (\log(e) + 2S_k^{(D)}) \cdot |V_{nf}(\alpha_0)| \geq \epsilon' |S|,$$

and equivalently

$$-|S| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)| + \left(\log(e) + 2S_k^{(D)} - \epsilon' \frac{|S|}{|V_{nf}(\alpha_0)|}\right) \cdot |V_{nf}(\alpha_0)| \geq 0.$$

As $F$ is $\Delta$-general, and $\text{dom}(\alpha_0) \leq \frac{1}{2}\Delta n$, we have that $|V_{nf}(\alpha_0)| \geq \Delta^2 n$, and thus

$$\frac{|S|}{|V_{nf}(\alpha_0)|} = \frac{n(\alpha_0) + |V_{nf}(\alpha_0)|}{|V_{nf}(\alpha_0)|} \leq \frac{n + |V_{nf}(\alpha_0)|}{|V_{nf}(\alpha_0)|} \leq \frac{2}{\Delta} + 1,$$

and it is sufficient to prove

$$-|S| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)| + \left(\log(e) + 2S_k^{(D)} - \epsilon' \left(\frac{2}{\Delta} + 1\right)\right) \cdot |V_{nf}(\alpha_0)| \geq 0.$$

As long as $\log(e) + 2S_k^{(D)} - \epsilon' \left(\frac{2}{\Delta} + 1\right) \geq 2$, things work out nicely. However, we know $S_k^{(D)} \geq S_k \geq S_3 > 0.38$, and $\log(e) + 2S_k^{(D)} > 1.44 + 2 \cdot 0.38 = 2.2$, so as long as $\epsilon' \left(\frac{2}{\Delta} + 1\right) \leq 0.2$, the inequality works, hence we can indeed use $\epsilon' := \frac{\Delta}{10 + 5\Delta}$. \qed

6.3 Open Problems

The main open problem is whether we can improve Unique $k$-SAT for $k > 3$. PPSZ becomes slower as $k$ increases, which makes an improvement easier. However, the guessing in SPARSE relied on the fact that non-critical ($\leq 2$)-clauses have all literals satisfied, which is not true for larger clauses. Another problem is that a random $k-1$-clause is satisfied by a fixed assignment with probability $2^{-(k-1)}$, so to give anything non-trivial, GETIND2CLAUSES would require a variable with degree at least $2^{(k-1)} + 1$, i.e. exponentially large in $k$. Thus, for the sparse case, Wahlström’s algorithm would be required to perform better than PPSZ
on $k$-CNF formulas with average degree exponential in $k$. However, it follows from Lemma 5 of [36] that Wahlström’s algorithm is only better than PPSZ for average degree linear in $k$.

Another question is the following: Suppose Wahlström’s algorithm is improved so that it runs in time $O(c^n)$ on 3-CNF formulas with average degree $D$. The sparsification lemma [15] shows that for $c \to 1$ and $D \to \infty$, we obtain an algorithm for 3-SAT running in time $O(b^n)$ for $b \to 1$. It would be nice if our approach would give a similar sparsification result. As it is now, this is not the case; for example our improvement is bounded by the performance of PPSZ on a uniquely satisfiable formula where all variables have exactly two critical clauses.

A third open problem is whether PPSZ becomes faster if the input formula $F$ is guaranteed to have many satisfying assignments. We have shown that this holds if we can guarantee some lower bound on the number of non-frozen variables; especially if $F$ is $\Delta$-general. Unfortunately having many satisfying assignments does not give such a lower bound, as there still might be some very isolated satisfying assignments (and these will be very likely to be chosen by PPSZ).
Appendix A

General Statements and Deferred Proofs

Here we some classical mathematical statements and proofs deferred from the main part.

A.1 General Statements

A.1.1 An Inequality about Logarithms

Lemma A.1. For $x \geq 0$,

$$\log(1 + x) \geq \log(e) \frac{x}{1 + x}.$$  

Proof. As $\log(x)$ is an antiderivative of $\log(e)1/x$ and $\log(1) = 0$, we have

$$\log(1 + x) = \int_1^{1+x} \log(e) \frac{1}{t} dt \geq \int_1^{1+x} \log(e) \frac{1}{1 + x} dt = \log(e) \frac{x}{1 + x}.$$

A.1.2 Independent Repetition of Algorithms with One-Sided Error

Lemma A.2. Let $A(F)$ be a randomized algorithm for some decision problem or promise problem $L$ (e.g. $k$-SAT or Unique $k$-SAT), where $F$ denotes an instance, with the following properties:
1. \( \mathcal{A}(F) \) runs in time \( O(t(F)) \) for some function \( t(F) \).

2. On no-instances \( F \), \( \mathcal{A}(F) \) always returns “no”.

3. On yes-instances \( F \), \( \mathcal{A}(F) \) returns “yes” with probability at least \( p(F) \), where \( p(F) \) is a function computable in time \( O(t(F)p(F)) \).

Then repeating \( \mathcal{A}(F) \) independently \( \lceil \frac{2}{p(F)} \rceil \) times and returning “yes” if one run of \( \mathcal{A}(F) \) returned “yes”, and returning “no” otherwise, gives a randomized algorithm \( \mathcal{A}'(F) \) for \( L \) running in time \( O(t(F)p(F)) \) and correct with probability at least \( \frac{2}{3} \). On no-instances, \( \mathcal{A}' \) is always correct.

**Proof.** The running time bound is easy to see; we first compute \( \lceil \frac{2}{p(F)} \rceil \) (we assume that elementary numerical operation can be done in constant time) and then run \( \mathcal{A}(F) \) that many times.

For the success probability, observe that \( \mathcal{A}' \) is always correct on no-instances. For “yes”-instances, \( \mathcal{A}' \) is not correct only if all runs of \( \mathcal{A} \) returned “no”. This happens with probability at most \( (1 - p(F))^{\lceil \frac{2}{p(F)} \rceil} \leq e^{-2} \leq \frac{1}{3} \) (using the inequality \( 1 + x \leq e^x \)). Hence \( \mathcal{A}' \) is correct with probability at least \( \frac{2}{3} \).

### A.1.3 Jensen’s inequality

**Theorem A.3.** Let \( I \) be a real interval. If \( f : I \rightarrow \mathbb{R} \) is a convex function and \( X \) is a random variable that attains values in \( I \) only, then

\[
E[f(X)] \geq f(E[X]),
\]

provided that both expectations exist.

**Proof ([37]).** Let \( \mu = E[X] \). Let \( \lambda \) be such that \( f(x) \geq f(\mu) + \lambda(x - \mu) \) for all \( x \in I \). \( \lambda \) exists because \( f \) is convex. Due to linearity of expectation,

\[
E[f(X)] \geq E[f(\mu) + \lambda(X - \mu)]
\]
\[
= f(\mu) + \lambda(E[X] - \mu)
\]
\[
= f(\mu)
\]
\[
= f(E[X]),
\]

and we are done. \( \square \)
A.1.4 Monotone Convergence

Theorem A.4 (Monotone Convergence Theorem). Let $B_1, B_2, \ldots$ be an infinite sequence of events such that $B_i \supseteq B_{i+1}$ for all $i \geq 1$. Then we have

$$\lim_{n \to \infty} \Pr (B_n) = \Pr \left( \bigcap_{i=1}^{\infty} B_i \right).$$

Proof. We write

$$\bigcap_{i=1}^{\infty} B_i = B_1 \setminus \bigcup_{i=1}^{\infty} (B_i \setminus B_{i+1}).$$

Since the union is disjoint and moreover contained in $B_1$, this implies

$$\Pr \left( \bigcap_{i=1}^{\infty} B_i \right) = \Pr (B_1) - \sum_{i=1}^{\infty} \Pr (B_i \setminus B_{i+1}).$$

By definition, the infinite sum on the right hand side is the limit of sums truncated after the first $n$ summands and since $\Pr (B_1)$ does not depend on this truncation, we can write

$$\Pr \left( \bigcap_{i=1}^{\infty} B_i \right) = \lim_{n \to \infty} \left( \Pr (B_1) - \sum_{i=1}^{n} \Pr (B_i \setminus B_{i+1}) \right).$$

But as we easily check, the expression inside the limit is $\Pr (B_{n+1})$, establishing the claim. \hfill \square

A.1.5 Numerical Integration and Riemann Sums

Lemma A.5. Let $\varphi : [0,1] \to [0,1]$ be a monotonically non-decreasing function. Then for any $N \in \mathbb{N}$,

$$\frac{1}{N} \sum_{i=0}^{N-1} \varphi \left( \frac{i}{N} \right) \leq \int_{0}^{1} \varphi (x) dx \leq \frac{1}{N} \sum_{i=0}^{N-1} \varphi \left( \frac{i}{N} \right) + \frac{1}{N}.$$

Proof. Since $\varphi$ is monotonically non-decreasing, we have for all $0 \leq x_0 \leq x \leq x_1 \leq 1$ that $\varphi (x_0) \leq \varphi (x) \leq \varphi (x_1)$. In particular, if $x_0 = i/N$ and $x_1 = (i + 1)/N$ for some $i \in \{0..N-1\}$, then

$$\frac{1}{N} \varphi (x_0) \leq \int_{x_0}^{x_1} \varphi (x) dx \leq \frac{1}{N} \varphi (x_1) = \frac{1}{N} \varphi (x_0) + \frac{1}{N} [\varphi (x_1) - \varphi (x_0)].$$
By summing over all $i$, we obtain
\[
\frac{1}{N} \sum_{i=0}^{N-1} \varphi \left( \frac{i}{N} \right) \leq \int_0^1 \varphi(x)dx \leq \frac{1}{N} \sum_{i=0}^{N-1} \varphi \left( \frac{i}{N} \right) + \frac{1}{N} \sum_{i=0}^{N-1} \left[ \varphi \left( \frac{i+1}{N} \right) - \varphi \left( \frac{i}{N} \right) \right].
\]

Noting that the last term is a telescopic sum yielding $\varphi(1) - \varphi(0) \leq 1$ completes the proof.

\[\square\]

**Lemma A.6.** Let $\varphi : [0, 1] \to [0, 1]$ and $\varphi_i : [0, 1] \to [0, 1]$ for $i \in \mathbb{N}$ be monotonically non-decreasing functions.

Suppose $\lim_{i \to \infty} \varphi_i(r) \geq \varphi(r)$ for all $r \in [0, 1]$. Then
\[
\lim_{i \to \infty} \int_0^1 \varphi_i(r)dr \geq \int_0^1 \varphi(r)dr.
\]

On the other hand, let $\varphi : [0, 1] \to [0, 1]$ and $\varphi_i : [0, 1] \to [0, 1]$ for $i \in \mathbb{N}$ be monotonically non-increasing functions.

Suppose $\lim_{i \to \infty} \varphi_i(r) \leq \varphi(r)$ for all $r \in [0, 1]$. Then
\[
\lim_{i \to \infty} \int_0^1 \varphi_i(r)dr \leq \int_0^1 \varphi(r)dr.
\]

**Proof.** First assume all functions are monotonically non-decreasing. By applying Lemma A.5 twice we have for any $N \in \mathbb{N}$
\[
\lim_{i \to \infty} \int_0^1 \varphi_i(r)dr \geq \lim_{i \to \infty} \frac{1}{N} \sum_{i=0}^{N-1} \varphi \left( \frac{i}{N} \right)
\geq \frac{1}{N} \sum_{i=0}^{N-1} \varphi \left( \frac{i}{N} \right)
\geq \int_0^1 \varphi(r)dr - \frac{1}{N}.
\]

As this holds for all $N \in \mathbb{N}$, indeed $\lim_{i \to \infty} \int_0^1 \varphi_i(r)dr \geq \int_0^1 \varphi(r)dr$.

The second part of the statement can be obtained by applying the first part to $\varphi' := 1 - \varphi$ and $\varphi_i' := 1 - \varphi_i$. \[\square\]
A.1.6 FKG Inequality

Theorem A.7. (FKG Inequality) Let \( A = \{A_1, A_2, \ldots, A_r\} \) be a collection of mutually independent binary random variables. Let \( E_1, E_2 \) be events which are determined by \( A \) and monotonically increasing in \( A \). Then

\[
\Pr (E_1 \land E_2) \geq \Pr (E_1) \cdot \Pr (E_2).
\]

Furthermore, for \( l \) such events \( E_1, E_2, \ldots, E_l \),

\[
\Pr \left( \bigwedge_{i=1}^l E_i \right) \geq \prod_{i=1}^l \Pr (E_i).
\]

Proof. First observe that the second statement follows from the first by simple induction, and the observation that the intersection of increasing events are also increasing.

For the first statement, we proceed by induction on \( r \). For a simple base case, let \( r = 1 \). If \( E_1 \) or \( E_2 \) is empty, the statement is trivial. There there are only two non-empty monotonically increasing events determined by \( A_1 \): either an event that occurs for both values of \( A_1 \) or an event that occurs only if \( A_1 = 1 \). Now if \( E_1 = E_2 \) the statement is trivial, so the only thing we have to check is if \( E_1 \) is the former of these cases and \( E_2 \) is the latter. If \( p \) is the probability that \( A_1 = 1 \), then the probability that both events occur is \( p \), the probability that \( E_1 \) occurs is 1 and the probability that \( E_2 \) occurs is \( p \), establishing the claim.

For the induction step, let us assume that the statement holds for smaller values of \( r \) and let \( p \) be the probability that \( A_1 = 1 \). We can rewrite the right hand side of our claim using the law of total probability as

\[
\Pr (E_1) \cdot \Pr (E_2) =
\]

\[
(p \Pr (E_1 \mid A_1 = 1)+ (1-p) \Pr (E_1 \mid A_1 = 0)) \cdot (p \Pr (E_2 \mid A_1 = 1)+ (1-p) \Pr (E_2 \mid A_1 = 0)) =
\]

\[
P^2 e_{11} e_{21} + p(1-p)(e_{11} e_{20} + e_{10} e_{21}) + (1-p)^2 e_{10} e_{20}.
\]

Now note that the mutual independence of \( A \) together with the monotonicity of both \( E_1 \) and \( E_2 \) in \( A_1 \) implies that both events can only be
more likely in the conditional space determined by \( \{ A_1 = 1 \} \) than in the case \( \{ A_1 = 0 \} \), thus we get \( e_{11} \geq e_{10} \) and \( e_{21} \geq e_{20} \). We can use this to estimate the mixed term in our expansion: consider \( e_{11} \geq e_{10} \) to be weights and the mixed term to be a weighted sum of \( e_{21} \geq e_{20} \). Currently, the larger weight accompanies the smaller summand. Thus if we exchange the weighs so that the larger summand gets the larger weight, the weighted sum can only increase. This yields

\[
\Pr(E_1) \cdot \Pr(E_2) \leq p^2 e_{11} e_{21} + p(1-p)(e_{10} e_{20} + e_{11} e_{21}) + (1-p)^2 e_{10} e_{20}
\]

which after some term recollection becomes

\[
\Pr(E_1) \cdot \Pr(E_2) \leq pe_{11} e_{21} + (1-p)e_{10} e_{20}. \quad (A.1)
\]

It is now time to invoke the induction hypothesis. We have assumed that the statement holds for smaller \( r \). The events \( E_1 \) and \( E_2 \) are, in the conditional space of \( \{ A_1 = 1 \} \), determined by \( A_2, A_3, \ldots, A_r \) and monotonically increasing in these variables. Therefore, in the conditional space of \( \{ A_1 = 1 \} \), the FKG inequality holds for \( E_1 \) and \( E_2 \) by the induction hypothesis, yielding that

\[
e_{11} e_{21} \leq \Pr(E_1 \land E_2 \mid A_1 = 1).
\]

\[
(A.2)
\]

Analogously

\[
e_{10} e_{20} \leq \Pr(E_1 \land E_2 \mid A_1 = 0).
\]

\[
(A.3)
\]

Plugging in (A.2) and (A.3) into (A.1) and applying once more the law of total probability, we can complete the induction step. \( \Box \)

### A.1.7 Expectations of Increasing Functions

**Lemma A.8.** Let \( A \) be a finite set and let \( f : 2^A \to \mathbb{R} \) be a function that satisfies \( f(B') \leq f(B) \), \( \forall B' \subseteq B \subseteq A \).

Let \( q' : A \to [0,1] \) and \( q : A \to [0,1] \) be two functions that satisfy \( q'(a) \leq q(a), \forall a \in A \).

At first, let \( C_{q'} \) and \( C_q \) be the empty set. Now consider the two following random experiments:

1. \( \forall a \in A \), choose \( a \in C_{q'} \) with probability \( q'(a) \), independently.
2. \( \forall a \in A \), choose \( a \in C_q \) with probability \( q(a) \), independently.

Then we have \( \mathbf{E}[f(C_{q'})] \leq \mathbf{E}[f(C_q)] \).
Proof. We define a set \( C_{q''} \) such that \( C_{q''} = C_{q'} \cup B \) where, for all \( a \in A \), we choose \( a \in B \) with probability \( p(a) = \frac{q(a) - q'(a)}{1 - q'(a)} \) (0 if \( q'(a) = 1 \)), independently of the other events.

Then the probability that a given \( x \) is in \( C_{q''} \) is

\[
\Pr (a \in C_{q''}) = \Pr (a \in C_{q'} \vee a \in B) = \Pr (a \in C_{q'}) + \Pr (a \in B) - \Pr (a \in C_{q'} \land a \in B) = q'(a) + p(a) - p(a)q'(a) = p(a)(1 - q'(a)) + q'(a) = q(a).
\]

Hence, \( C_{q''} \) and \( C_q \) have the same probability distribution, so

\[
E[f(C_{q''})] = E[f(C_q)].
\]

But then, \( C_{q'} \subseteq C_{q''} \), so

\[
E[f(C_{q'})] \leq E[f(C_{q''})] = E[f(C_q)],
\]

which concludes the proof.

Lemma A.9 ([32]). Let \( X, T, T', U, V \) be finite vectors of real random variables, all independent, taking only finitely many values. Suppose \( T \) and \( T' \) are over \( \mathbb{R}^n \) and have the same distribution. Let \( f : \mathbb{R}^m \rightarrow \mathbb{R} \), \( g : \mathbb{R}^{m'} \rightarrow \mathbb{R} \) be monotonically non-decreasing functions, and let \( h : \mathbb{R} \rightarrow \mathbb{R} \) be a concave function. Then

\[
E[h(f(X, T, U) + g(X, T, V))] \leq E[h(f(X, T, U) + g(X, T', V))].
\]

Proof. Observe first that if the lemma holds for any particular choice of \( X, U \) and \( V \) and not only when taking expectation over these three vectors, the lemma follows. We prove this by substituting arbitrary concrete real vectors for \( X, U \) and \( V \), and consequently we obtain functions \( f' \) and \( g' \) that depend only on \( T \) and \( T' \), and are still monotone. Hence, it suffices to prove that

\[
E[h(f'(T) + g'(T))] \leq E[h(f'(T) + g'(T'))] \quad (A.4)
\]

where \( T, T' \) are over \( \mathbb{R}^n \) and \( f', g' : \mathbb{R}^n \rightarrow \mathbb{R} \) are monotonically non-decreasing functions. Second, we claim that it suffices to examine the case \( n = 1 \). If we can prove the lemma for \( n = 1 \), we can repeatedly replace every component \( T_i \) of \( T \) with \( T'_i \) of \( T'_i \), eventually obtaining the above inequality. Hence, let us assume \( n = 1 \), and \( T, T' \) independent identically distributed real random variables. By symmetry, it holds that

\[
E[h(f'(T) + g'(T))] = E[h(f'(T') + g'(T'))],
\]

and

\[
E[h(f'(T) + g'(T'))] = E[h(f'(T') + g'(T))].
\]

Multiplying equation
(A.4) by 2 and using these identities, we obtain:

\[
\begin{align*}
\mathbb{E} [h(f'(T) + g'(T))] & \leq \mathbb{E} [h(f'(T) + g'(T'))] \\
\iff 2\mathbb{E} [h(f'(T) + g'(T))] & \leq 2\mathbb{E} [h(f'(T) + g'(T'))] \\
\iff \mathbb{E} [h(f'(T) + g'(T))] + \mathbb{E} [h(f'(T') + g'(T'))] & \leq \mathbb{E} [h(f'(T) + g'(T'))] + \mathbb{E} [h(f'(T') + g'(T))] \\
\iff \mathbb{E} [h(f'(T) + g'(T)) + h(f'(T') + g'(T'))] & \leq \mathbb{E} [h(f'(T) + g'(T')) + h(f'(T') + g'(T))].
\end{align*}
\]

We now prove this last inequality, and for this we show that it holds not only in expectation, but for every particular choice \( t, t' \in \mathbb{R} \) of \( T, T' \). Fix such a choice. Without loss of generality, we can assume that \( t \leq t' \) and therefore \( f'(t) \leq f'(t') \) and \( g'(t) \leq g'(t') \), using monotonicity. Second, we can assume without loss of generality that \( f'(t) + g'(t') \leq f'(t') + g'(t) \). Writing \( a := f'(t) + g'(t), b := f'(t) + g'(t'), c := f'(t') + g'(t) \) and \( d := f'(t') + g'(t') \), we see that \( a \leq b \leq c \leq d \). The claimed inequality states that \( h(a) + h(d) \leq h(b) + h(c) \). Divided by two, the statement becomes equivalent to saying that the midpoint of the longer line segment in Figure A.1 is below the one of the shorter. This follows immediately from \( h \) being a concave function (note that \( \frac{a+d}{2} = \frac{b+c}{2} \), so both midpoints have the same \( x \)-coordinate). \( \square \)

![Figure A.1: Illustration of the final equality in the proof of Lemma A.9.](image)
A.1.8 Bounding Binomial Coefficients

Lemma A.10. For $p \in [0, 1]$, define $H(p) := -p \log p - (1-p) \log (1-p)$ (with $0 \log 0 := 0$).

If $pn$ is an integer, then $\binom{n}{pn} \leq 2^{H(p)n}$.

Proof (adapted from [37]). By the binomial theorem, we have

$$1 = (p + (1-p))^n \geq \binom{n}{pn} \cdot p^{pn} (1-p)^{(1-p)n}.$$  

Furthermore,

$$\log \left( p^{pn} (1-p)^{(1-p)n} \right) = n \cdot (p \log p + (1-p) \log (1-p)) = n \cdot (-H(p)).$$

Hence $1 \geq \binom{n}{pn} \cdot 2^{-H(p)n}$ and so $2^{H(p)n} \geq \binom{n}{pn}$.  

\hfill \Box

A.2 Deferred Proofs

A.2.1 Proof of Lemma 3.14

Lemma 3.14. For $k = 3$, we have

$$R_3(p) = \begin{cases} \frac{p^2}{(1-p)^2} & \text{if } p < \frac{1}{2}, \\ 1 & \text{otherwise.} \end{cases}$$

In general, define $S_k(t) = \frac{t^{k-1}}{1-t}$ for $t \in [0, 1)$, and $S_k(1) = \frac{k-2}{k-1}$. Then

- $S_k$ is monotonically non-decreasing and continuous on $[0, 1]$.
- For $p \in [0, \frac{k-2}{k-1}]$, $R_k(p)$ is the inverse of $S_k(t)$.
- For $p \in [\frac{k-2}{k-1}, 1]$, $R_k(p) = 1$.
- $R_k$ is monotonically non-decreasing and continuous on $[0, 1]$.

Proof. Remember that $R_k(p)$ is the smallest non-negative solution to $q = (p + (1-p)q)^{k-1}$. For $k = 3$, the equation is a quadratic polynomial with solutions $1$ and $\frac{p^2}{(1-p)^2}$, and the first statement follows.
Appendix A. General Statements and Deferred Proofs

Suppose \( t \in [0, 1) \) and write \( S_k(t) = 1 - \frac{1}{1-t} t^{\frac{1}{k-1}} \). We have

\[
(1 - t^{\frac{1}{k-1}}) \cdot \left( \sum_{i=0}^{k-2} t^{\frac{i}{k-1}} \right) = 1 - t,
\]

and by using this equality in the denominator we see

\[
S_k(t) = 1 - \frac{1}{\sum_{i=0}^{k-2} t^{\frac{i}{k-1}}},
\]

From this it is clear that \( \lim_{t \to 1^+} S_k(t) = S_k(1) \), and that \( S_k(t) \) is continuous and monotonically non-decreasing on \([0, 1]\).

Now we show the other statements. The following proof is adapted from [12] (which is turn is based on the proof from [24]). Since \( q = 1 \) satisfies this equation, we have \( R_k(p) \leq 1 \). Let

\[
f_k(p, q) = (p + (1 - p)q)^{k-1}.
\]

As \( f_k \) is increasing in \( p \), for \( p \in \left[\frac{k-2}{k-1}, 1\right] \) and \( q < 1 \), \( f_k \) is minimized at \( p = \frac{k-2}{k-1} \), and in this case

\[
f_k(p, q) = (p + (1 - p)q)^{k-1} \\
\geq \left( \frac{k-2}{k-1} + \frac{1}{k-1} q \right)^{k-1} \\
= \left( 1 - \frac{1}{k-1} (1-q) \right)^{k-1}
\]

Using \((1+p)^n > 1 + np\) for \( p > 0 \) and \( n \) a positive integer, we have \( f_k(p, q) > q \) and the smallest non-negative root to \( q = f_k(p, q) \) is 1. Thus, for \( p \in \left[\frac{k-2}{k-1}, 1\right] \), \( R_k(p) = 1 \).

For the second statement of the lemma, for \( q \in [0, 1) \), the following are equivalent:

\[
(p + (1 - p)q)^{k-1} = q \\
p + (1 - p)q = q^{\frac{1}{k-1}} \\
p(1 - q) = q^{\frac{1}{k-1}} - q \\
p = \frac{q^{\frac{1}{k-1}} - q}{(1-q)} = S_k(q).
\]
Therefore, if $R_k(p) < 1$, then $S_k(q)$ is the inverse of $R_k(p)$. We now want to prove that for $p \in \left[0, \frac{k-2}{k-1}\right]$, $R_k(p) < 1$, which will conclude the proof of the second statement of the lemma.

First observe that, for $p = 0$, the smallest $q \geq 0$ satisfying the equation

$$q = (p + (1 - p))q^{k-1} = q^{k-1}$$

is $q = 0$; hence $R_k(0) = 0 < 1$, so we only have to prove the statement for $0 < p < \frac{k-2}{k-1}$.

By definition of $R_k(p)$, we can write

$$R_k(p) = (p + (1 - p)R_k(p))^{k-1}.$$ 

Let $\delta = 1 - R_k(p)$; this yields:

$$(p + (1 - p)(1 - \delta))^{k-1} + \delta = 1$$

$$(1 - (1 - p)\delta)^{k-1} + \delta - 1 = 0.$$ 

Let $g(p, \delta) = (1 - (1 - p)\delta)^{k-1} + \delta - 1$. For a fixed $0 \leq p \leq \frac{k-2}{k-1}$, $g$ is a continuous function of $\delta$. Moreover, we have that, still for a fixed $p$, $g'(p, \delta) = (k - 1)(p - 1)(1 - (1 - p)\delta)^{k-2} + 1$, which yields $g'(p, 0) = (k - 1)(p - 1) + 1$. As we only consider $k \geq 3$, $g'(p, 0) < 0$. We also have that $g(p, 0) = 0$, and that $g(p, 1) > 0$ for $p > 0$. So for all $0 < p < \frac{k-2}{k-1}$ there exists a $\delta^* > 0$ such that $g(\delta^*, p) = 0$. So $1 - \delta^* < 1$ is a solution of

$$q = (p + (1 - p)q)^{k-1}$$

which proves that $R_k(p) < 1$, and hence that $S_k(q)$ is the inverse of $R_k(p)$ on the interval $\left[0, \frac{k-2}{k-1}\right)$. We have now completely characterized $R_k$ in terms of $S_k$ and it follows that $R_k$ is monotonically non-decreasing and continuous.

\section*{A.2.2 Proof of Lemma 3.19}

\textbf{Lemma 3.19.}

$$\int_0^1 R_k(x)dx = 1 - S_k,$$

$$\int_0^1 R_3(x)dx = 2 - 2 \ln 2.$$
Proof. Let $S_k(t)$ be defined as in Lemma 3.14. We have $1 - S_k = 1 - \int_0^1 S_k(t) dt$. By integration by parts and the substitution $x = S_k(t)$ we have

$$1 - S_k = 1 - 1 \cdot S_k(1) + 0 \cdot S_k(0) + \int_0^1 S_k'(t) dt.$$ 

$$= \frac{1}{k-1} + \int_0^1 R_k(S_k(t)) S_k'(t) dt$$ 

$$= \frac{1}{k-1} + \int_0^{S_k(1)} R_k(x) dx$$ 

$$= \int_{\frac{k-2}{k-1}}^1 R_k(x) dx + \int_0^{\frac{k-2}{k-1}} R_k(x) dx$$ 

$$= \int_0^1 R_k(x) dx.$$ 

For $k = 3$, we have by 3.14 that $R_3(p) = \frac{p^2}{(1-p)^2}$ for $p < \frac{1}{2}$ and $R_3(p) = 1$ otherwise. Hence

$$\int_0^1 R_3(x) dx = \frac{1}{2} + \int_0^{\frac{1}{2}} \frac{p^2}{(1-p)^2}$$ 

$$= \frac{1}{2} + \int_{\frac{1}{2}}^1 \frac{(1-p)^2}{p^2}$$ 

$$= \frac{1}{2} + \int_{\frac{1}{2}}^1 \left( \frac{1}{p^2} - \frac{2}{p} + 1 \right)$$ 

$$= \frac{1}{2} - 1 + 2 + 2 \ln \frac{1}{2} + \frac{1}{2}$$ 

$$= 2 - 2 \ln 2.$$
Bibliography


Curriculum Vitae

Timon Hertli

born October 28, 1985
citizen of Kleinandelfingen ZH, Switzerland

1998–2004 Highschool
Kantonsschule Alpenquai Luzern, Luzern, Switzerland
Degree: Matura

2004–2010 Studies of Computer Science
ETH Zürich, Zürich, Switzerland
Degree: Master of Science ETH in Computer Science

2010–2014 PhD Student
Institute of Theoretical Computer Science
ETH Zürich, Zürich, Switzerland