ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Towards the Derandomization of the PPSZ algorithm for the Multiple Satisfying Assignments Case

Master Thesis

Isabelle Hurbain

April 1st, 2014

Advisors: Prof. Dr. Emo Welzl, Timon Hertli
Department of Computer Science, ETH Zürich

Abstract

The PPSZ algorithm [11] is currently the fastest algorithm for k-SAT [5]. It has been derandomized in 2005 by Rolf [13] for the case where the formula is guaranteed to have a single satisfying assignment. In this thesis, we explore a derandomization plan for the general case, and the associated questions. We start by reviewing previous work on derandomization of the PPZ [12] and PPSZ algorithms [13]. We then prove a general result on the PPSZ algorithm that may make subsequent analyses easier.

We next present our work on the derandomization itself: our final goal would be to reduce as much as possible the number of random bits used in the PPSZ algorithm. We consider the random bits induced by choosing a random permutation: we want to find a small set of permutations sach that PPSZ still returns a satisfying assignment with a good enough probability. This would allow us to enumerate this set in a derandomization. We have not been able to find such a small set of permutations yet, but we have gained some insight on the reason why this problem resisted our attempts so far, namely that it is difficult to control the correlations between variables when not considering the whole set of permutations. Finally, we consider the question of what happens if we replace the random bits that are used for non-frozen variables by some construction, for instance if we consider them set by an external oracle. We have proven that the success probability of PPZ was not adversely affected by setting the non-frozen variables to 0. Whether it is also the case for PPSZ is still not known; whether it is possible to construct an oracle in reasonable time is not known either. We conclude this thesis by enumerating a number of issues and questions that are still open.

Acknowledgements

First and foremost, many thanks to Timon Hertli for supervising this work; his insights, advice and enthusiasm made the development of this thesis a truly rewarding and enjoyable experience. I would also like to thank Robin Moser for the fruitful discussions about some ideas in this thesis. The SAT course run in 2012 by Emo Welzl and Robin Moser was truly inspiring: many thanks to them as well for this. Finally, thank you Tobias and Matthias for proofreading the draft version of this thesis.

Contents

Part I

Preliminaries

Chapter 1

Introduction and notation

1.1 Motivation

The boolean satisfiability problem, in short SAT, is the decision problem of determining if there exists an assignment to the variables of a boolean formula such that the whole formula evaluates to 'true'. It is the first known NP-complete problem, as proven independently by Cook [3] and Levin [8]. Even when restricting the boolean formulas to $k-$CNF formulas – conjunctive normal form, i.e. formulas that are composed of the conjunction ("AND") of disjunction ("OR") of k literals – the problem, then called k-SAT, stays NP-complete. Provable bounds for the runtime of algorithms solving k-SAT have been an active research domain for decades. The record for these bounds is currently held by the PPSZ algorithm [11], named after its authors Paturi, Pudlák, Saks and Zane; the bound itself has been proven to hold in full generality by Hertli [5]; the corresponding bounds for the runtime are $\mathcal{O}(1.308^n)$ for 3-SAT and $\mathcal{O}(1.469^n)$ for 4-SAT. PPSZ is a randomized algorithm: it uses random bits to make some choices during its execution. The performance of such an algorithm is actually expressed in terms of success probability. It is proven that PPSZ, given a satisfiable 3-SAT formula as an input, will return a satisfying assignment with probability at least 1.308^{-n} (where n is the number of variables of the formula), where the probability is over the choice of random bits that are made by the algorithm. This probability is quite low by itself, but if we now proceed to $\lambda \cdot 1.308^n$ independent repetitions of PPSZ, the probability that PPSZ does not find a satisfying assignment is $e^{-\lambda}$, and can thus be made arbitrarily small. When talking about the runtime of PPSZ, we actually give an upper bound on the number of repetitions that is guaranteed to yield an exponentially small probability that a satisfying assignment will not be found[1].

[1] A single run of PPSZ takes polynomial time in n; this factor is absorbed by the precision with which we give the success probability and running time of the algorithm.

A year after the introduction of the PPSZ algorithm, but some time before Hertli's analysis proving its general bound, Schöning proposed another randomized algorithm [15] that held, at that time, the runtime record for 3-SAT formulas with multiple satisfying assignments with a runtime of $\mathcal{O}(1.334^n)$. PPSZ and Schöning's algorithms use a different approach. The idea of Schöning is to start from a random assignment from the hypercube, and to search around that random assignment for a satisfying assignment, by guiding the search according to the clauses in the formula. On the other hand, the idea of PPSZ is to repeatedly cut the hypercube in two, keeping the "correct" half if it can be identified or a random one if it cannot, until it contains only a satisfying assignment or can be identified to not contain any. Schöning's algorithm has been derandomized by Moser and Scheder [10] and holds the current record of the fastest deterministic algorithm for k-SAT, $k \geq 4$; for the case of 3-SAT, the bound of that derandomization has been improved to $\mathcal{O}(1.3303^n \cdot \text{poly}(n))$ by Makino, Tamaki and Yamamoto [9] by derandomizing a modified version of Schöning's algorithm.

One of the fundamental open questions when it comes to randomized algorithms is to know whether randomness is at all necessary. In other words, given a randomized algorithm, is there a deterministic algorithm whose runtime is arbitrarily close to the expected runtime of the randomized algorithm? Without considering this open question in its full generality, the derandomization of individual algorithms still represents an adequate challenge and often allows to gain insight on the original algorithm. A full derandomization of PPSZ would also be, at the time of this writing, the fastest deterministic algorithm for k-SAT.

1.2 Notation

We use the notational framework and definitions established in [17].

1.2.1 Notions specific to the satisfiability of boolean formulas

A boolean formula in propositional is an expression built from boolean variables, the operators AND ($\wedge$), OR($\vee$) and NOT (denoted by an upper bar $\bar{\cdot}$ on the negated expressions). For a given formula F, we denote by V the set of all boolean variables that appear in F.

An *assignment α on V* is a mapping $\alpha : V \to \{0,1\}$. An assignment α on V satisfies a formula F if, when substituting the variables with their value in the assignment, F evaluates to 1. Such an assignment is called a *satisfying assignment*. The set of all satisfying assignments of F is denoted $\text{sat}_V(F)$ or $\text{sat}(F)$ if V is clear from the context.

The problem of *satisfiability of boolean formulas* or SAT is the problem of deciding whether, given a boolean formula in propositional logic, there exists an assignment that satisfies it. In the realm of this thesis, we consider exclusively formulas in conjunctive normal form (we use the term *CNF formula*); to define this notion, we first need the notion of *literal* and the notion of *CNF clause*.

A *literal* is a variable or the negation of a variable; for a variable x we define the *positive literal* x and the *negative literal* $\bar{x}$, where $\bar{x} = 0$ iff $x = 1$.

A *CNF clause* C over V is a disjunction of literals whose variables come from V: for instance, $(x_1 \vee \bar{x}_2 \vee x_3)$ is a CNF clause over V if $x_1, x_2, x_3 \in V$.The set of variables that occur in C is denoted by $\mathrm{vbl}(C)$.

A *CNF formula* over V is a conjunction of CNF clauses over V: for instance, $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \bar{x}_4)$ is a CNF formula over V if $x_1, x_2, x_3, x_4 \in V$. If all the clauses of a CNF formula contain exactly (respectively at most) k literals, it is called a k-CNF formula (respectively a $(\leq k)$-CNF formula). The decision problem for $(\leq k)$-CNF formulas is called k-SAT.

Every boolean formula in propositional logic F admits an equivalent 3-SAT formula F' such that F is satisfiable if and only if F' is satisfiable; moreover F' can be built in polynomial time of the size of F.

For a partial or a total assignment α_0, we define by *domain of* α_0 (denoted by $\mathrm{dom}(\alpha_0)$) as the set of variables x such that α_0 contains a value assignment $l = (x \mapsto b)$, where $b = 0$ or $b = 1$.

For a formula F and a partial assignment α_0 on $\mathrm{vbl}(F)$, we denote by $F^{[\alpha_0]}$ the restriction of F to α_0, that is the formula that is obtained by substituting the variables in the domain of α_0 with their assigned values (for a variable x that is assigned the value 0, we replace in the formula all instances of x by 0 and all instances of $\bar{x}$ by 1). For a CNF clause, this means that a clause containing a satisfied literal of the variable x is removed from the formula (since it is satisfied), and that the literals of the variable x that are unsatisfied are removed from their corresponding clause: they cannot contribute to the satisfaction of these clauses. A clause that does not contain any literal (empty clause) is always unsatisfied; we denote an empty clause by $\square$.

We can consider some elements of these definitions in the language of sets, and we will use these notation in the following of this thesis.

The set of all literals is $V \cup \bar{V}$, where V contains all the positive literals and $\bar{V}$ all the negative literals of V. A *clause* C over V is a subset of pairwise strictly distinct literals of $V \cup \bar{V}$ (where strictly distinct means that if the literal x is in the clause, then $\bar{x}$ is not, and reciprocally). C is a k-clause if $|C| = k$ and a $(\leq k)$-clause if $|C| \leq k$. A CNF formula F is a set of clauses. This allows us to use the set terminology when talking about formulas and clauses; in

particular, we will be able to write that $x \in C$ and that $C \in F$ for x a literal, C a clause and F a formula.

We can interpret a partial assignment α_0 on $V(F)$ as a set of value assignments $l = (x \mapsto b)$, where $b = 0$ or $b = 1$. We denote $\alpha_0(x) = b$. We use the shorthand $\alpha_0[l] = \alpha_0 \cup \{l\} = \alpha_0 \cup \{x \mapsto b\}$. If the assignment $(x \mapsto b)$ is in the partial assignment α_0, we write that $(x, b) \in \alpha_0$.

1.2.2 General notions

On top of the notions that are specific to the satisfiability problem, we also need a few general notions that are out of the scope of satisfiability per se.

Given a set V (of variables), we define a *permutation* π of V as a total ordering of the elements of V. We define Π_V as the set of all permutations of V. Alternatively, we will define π as a *placement* of the variables in V as $\pi : V \to [0, 1]$. If the values $\pi(x)$ are all distinct, we can sort the values $\pi(x)$ to obtain a permutation of V. When picking the values $\pi(x)$ independently and uniformly at random from $[0, 1]$ for each $x \in V$, then with probability 1, π is injective and we obtain a uniformly distributed permutation of V.

A k-ary tree is a rooted tree such that every node has at most k children. A full k-ary tree is a tree such that every internal node has exactly k children and such that every path from the root to a leaf has the same length.

We will use the abbreviation "u.a.r." for "uniformly at random". We will also use the shorthand $x \in_\Phi X$ to indicate that x is chosen in the set X according to the distribution Φ (and $x \in_{\text{u.a.r.}} X$ to indicate that x is chosen uniformly at random in X).

Chapter 2

The PPZ and PPSZ algorithms

The PPZ algorithm, named after its authors Paturi, Pudlák and Zane ([12]) can be described as follows:

> Pick a random permutation of the variables of the formula, and process the variables in that order. Start with an empty assignment. For every variable x, if the formula contains the clause $\{x\}$, add $\{x \mapsto 1\}$ to the assignment; if the formula contains the clause $\{\bar{x}\}$, add $\{x \mapsto 0\}$ to the assignment. If the formula contains neither of these clauses, pick a random value $b \in \{0, 1\}$ and add $\{x \mapsto b\}$ to the assignment. Modify the formula to reflect the fact that a variable has been set. If the resulting assignment is satisfying, then we are done. Otherwise, try again.

The PPSZ algorithm, named after its authors Paturi, Pudlák, Saks and Zane ([11]), is a refinement of the PPZ algorithm, and can be described as follows:

> Pick a random permutation of the variables of the formula, and process the variables in that order. Start with an empty assignment. For every variable x, check if its value b can be determined by examining a small number of clauses. If yes, add $\{x \mapsto b\}$ to the assignment. If not, pick a random value for b in $\{0, 1\}$, add $\{x \mapsto b\}$ to the assignment. Modify the formula to reflect the fact that a variable has been set. If the resulting assignment is satisfying, then we are done. Otherwise, try again.

In this chapter, we will give a more formal description of these two algorithms, and we will give the bounds that has been established for the expected runtime of these algorithms. We will also give a few tools and give a few results that are used in the analysis in these algorithms, and that we will also use in this thesis.

2.1 The PPZ algorithm

We define the PPZ algorithm formally as follows:

```
nd for    function ppz(F, V):
    Input: F CNF formula over V
    Output: Satisfying assignment for F or ∅
    π ←u.a.r. Π_V;
    α ← ∅;
    for i ← 1 to |V|:
        x ← π(i);
        if {x} ∈ F:
            b ← 1;
        else if {x̄} ∈ F:
            b ← 0;
        else:
            b ←u.a.r. {0,1};
        α ← α ∪ {x ↦ b};
        F ← F^[x↦b];
        if □ ∈ F:
            return ∅;
    e
    return α;
```

This algorithm has been introduced by Paturi, Pudlák and Zane [12], and they established its success probability:

Theorem 2.1 ([12]) *The probability that ppz() finds a satisfying assignment in a satisfiable $(\leq k)$-CNF formula over n variables is at least $2^{-n+n/k}$.*

The proof of this theorem relies heavily on the notion of "j-isolation", which itself relies on the notion of critical variables.

Definition 2.2 (Critical variables and critical clauses [17]) *Given an assignment $\alpha \in sat_V(F)$, we call a variable $x \in V$* critical for *α if flipping the value of x in α stops it being satisfying. This requires at least one clause in F that has x or $\bar{x}$ as a unique literal that is set to 1 by α. Such a clause is called a* critical clause for *x.*

Definition 2.3 (Isolation of an assignment [17]) *By $j(\alpha)$, we denote the number of critical variables for α, and we say that α is j-*isolated *if $j(\alpha) \geq j$. n-isolated assignments (thus $j(\alpha) = n$) are called* isolated.

The PPZ algorithm, when using the permutation π, can be seen as a decoding function for the encoding function Φ_π, defined as follows. Let $\alpha \in \{0,1\}^n$ be a satisfying solution of F. Build the encoding bit by bit following the order defined by the permutation $\pi = (x_1, ..., x_n)$. The i^{th} bit

is either omitted if $F^{[x_1 \mapsto \alpha(x_1), x_2 \mapsto \alpha(x_2), \ldots, x_{i-1} \mapsto \alpha(x_{i-1})]}$ contains the clause $\{x_i\}$ or $\{\bar{x}_i\}$, or $\alpha(x_i)$ otherwise. The length of the encoding for a given assignment depends on the permutation π.

Together with the notion of j-isolation, the following lemma is crucial to prove Theorem 2.1:

Lemma 2.4 (Satisfiability Coding Lemma [12]) *If x is a j-isolated satisfying assignment of a k-CNF, then its average (over all permutations π) description length under the encoding Φ_π is at most $n - j/k$.*

For a full proof of this lemma and of Theorem 2.1, refer to the original paper [12].

2.2 The PPSZ algorithm

The PPSZ algorithm [11] is an improvement on the PPZ algorithm: instead of checking whether the clause $\{x\}$ or $\{\bar{x}\}$ exists in the formula, it checks whether such a clause can be derived from a subset of clauses in the formula. A slightly modified version of the original PPSZ algorithm, proposed by Hertli [5], uses the notion of D-implication:

Definition 2.5 *Let F be a satisfiable CNF formula over a set of variables V. We say that a literal l is D-implied by F (in writing $F \vDash_D l$) if there exists a subset G of F with $|G| \leq D$ such that all satisfying assignments of $G = (G, V)$ set l to 1.*

Observe that, in this definition, we are talking about literals and not variables: "setting l to 1" can mean "setting x to 1" or "setting x to 0" depending on whether the considered literal is x or $\bar{x}$.

With this definition, we can state the PPSZ algorithm formally, where α_0 is $\emptyset$ when running the algorithm itself, but will be used later in the analysis.

function ppsz(F, V, α_0, D)**:**
  Input: F CNF formula over V, $D \in \mathbb{N}_0$, α_0 partial assignment whose values we want to fix in advance
  Output: Satisfying assignment for F or $\emptyset$
  $\pi \leftarrow_{\text{u.a.r.}} \Pi_{V \setminus \text{dom}(\alpha_0)}$;
  $\alpha \leftarrow \alpha_0$;
  for $i \leftarrow 1$ **to** $|\pi|$**:**
    $x \leftarrow \pi(i)$;
    if $F^{[\alpha]} \vDash_D \{x \mapsto 1\}$**:**
      $b \leftarrow 1$;
    else if $F^{[\alpha]} \vDash_D \{x \mapsto 0\}$**:**
      $b \leftarrow 0$;
    else:
      $b \leftarrow_{\text{u.a.r.}} \{0,1\}$;
    $\alpha \leftarrow \alpha \cup \{x \mapsto b\}$;
    $F \leftarrow F^{[x \mapsto b]}$;
  end for
  if α *satisfies* F**:**
    return α;
  return failure;

Bounds on the runtime of the PPSZ algorithm were first given by the original authors, Paturi, Pudlák, Saks and Zane [11], but the proven bounds were better for the case where the formula is guaranteed to have a unique satisfying assignment than for the case where the formula could have an arbitrary number of satisfying assignments. Hertli [5] closed the gap, which allows us to state the following theorem:

Theorem 2.6 ([11, 5]) *For any D, the probability that ppsz(F, V, $\emptyset$, D) finds a satisfying assignment in a satisfiable $(\leq k)$-CNF formula F over a set V of n variables is at least $2^{-S_k^{(D)} n}$, where*

$$\lim_{D \to \infty} S_k^{(D)} = S_k = \int_0^1 \frac{t^{1/(k-1)-t}}{1-t}\, dt.$$

The proof of this theorem is considerably more involved than the proof of Theorem 2.1. In this section, we will give a few definitions and results that are involved in the proof; for a full proof, refer to the original papers [11, 5]. An extended, more detailed write-up of these papers can be found in [17] and in [6].

2.2.1 Variable classification for PPSZ

We define here some of the terminology that we will use when talking about variables in the context of PPZ and PPSZ. Let F be a CNF formula over n variables, $\alpha \in \{0,1\}^n$ a satisfying assignment, and α_0 a partial assignment that is compatible with α. To simplify notation, we will write $\mathcal{U}(\alpha_0) = V \backslash \mathrm{dom}(\alpha_0)$ and $n(\alpha_0) = |\mathcal{U}(\alpha_0)|$. Let $\pi = (x_1, ..., x_{n(\alpha_0)})$ be a permutation of $\mathcal{U}(\alpha_0)$, and $D > 0$.

Definition 2.7 *A variable is called* forced *(with respect to F, α_0, α, π, and D) if $F^{[\alpha_0 \cup \{x_1 \mapsto \alpha_(x_1), ..., x_{i-1} \mapsto \alpha(x_{i-1})\}]}$ D-implies x_i or $\bar{x}_i$. Otherwise, the variable is called* guessed. *We denote the set of forced (resp. guessed) variables within $\mathcal{U}(\alpha_0)$ by $Forced(F, \alpha_0, \alpha, \pi, D)$ (resp. $Guessed(F, \alpha_0, \alpha, \pi, D)$).*

Definition 2.8 *A variable x is called* frozen *(with respect to F and α_0) if all satisfying assignments of $F^{[\alpha_0]}$ set x to the same value. It is called* non-frozen *otherwise.*

Observe that, for a given F, α_0, π and D, a variable x can be:

- non-frozen (we write that $x \in \mathrm{V}_{\mathrm{nf}}(\alpha_0)$): there exists two satisfying assignments of $F^{[\alpha_0]}$ α and α' such that $\alpha(x) = 0$ and $\alpha'(x) = 1$,
- frozen, but not forced (we write that $x \in \mathrm{V}_{\mathrm{fr}}(\alpha_0)$): all the satisfying assignments of $F^{[\alpha_0]}$ send x to the same value b, but for every subset of G of F such that $|G| \leq D$, there is a satisfying assignment of G that sets x to $\bar{b}$,
- forced (we write that $x \in \mathrm{V}_{\mathrm{fo}}(\alpha_0)$): $F \vDash_D x$ or $F \vDash_D \bar{x}$ (a forced variable is necessarily frozen).

The frozen, non-forced variables are the only "source" of possible errors for the PPSZ algorithm: in the other cases, assigning a value to the a variable cannot make a partial satisfying assignment unsatisfying. In a formula that admits a single satisfying assignment, all variables are frozen.

2.2.2 Critical clause trees

The analysis of the PPSZ algorithm relies on *critical clause trees*, whose definition we give here.

Definition 2.9 ([17]) *Let F be a satisfiable $\leq k$-SAT formula, let α^* be a satisfying assignment, and let x be a frozen variable of F. We define a* critical clause tree of *x and we denote by T_x a tree which is built as follows. T_x is a rooted tree such that every node has at most $k-1$ children, and where every node $u \in V(T_x)$ is labelled both with a variable $y \in V$, which we denote by var-label(u), and a clause $C \in F$, denoted by clause-label(u). For a fixed x:*

1. *Start with T_x consisting of a single root. This root has variable label x, and no 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 $W := \{\textit{var-label}(v) \mid v \in V(T_x)$ is an ancestor of u in $T_x\}$, where* ancestor *includes u itself and the root.*

   b) *Define the total assignment μ as:*

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

   c) *Let $C \in F$ be a clause not satisfied by μ. Since x is a frozen variable and its value is assigned to $1 - \alpha^*(x)$, such a clause exists. Set clause-label$(u) = C$.*

   d) *For each unsatisfied literal $w \in C$, 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.*

Figure 2.1 shows an example of a critical clause tree being build (before the clause-labels for the current leafs are assigned) for a 3-CNF formula containing the clauses $\{x, \bar{y}, \bar{z}\}$, $\{x, \bar{v}, \bar{w}\}$ and $\{x, z, \bar{a}\}$, and whose unique satisfying assignment is the all-1 assignment.

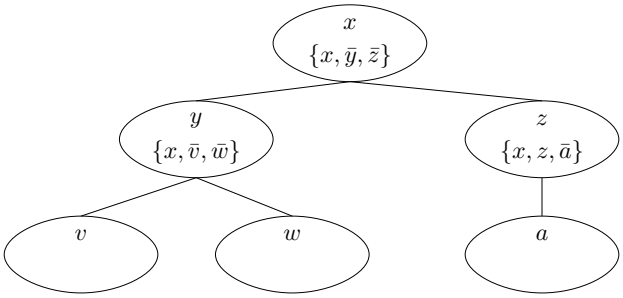

Figure 2.1: A critical clause tree being built

We then call a node $u \in T_x$ *reachable at time γ w.r.t. π* if there exists a path $v_0, v_1, ..., v_m$ such that v_0 is the root of the tree, $v_m = u$ and $\pi(v_i) \geq \gamma$ for all $1 \leq i \leq m$. Let us denote Reachable(T_x, γ, π) the set of all nodes in T_x reachable at time γ w.r.t. π. Reachable nodes and forced variables are related as follows:

Lemma 2.10 ([17]) *If we have $|\textit{Reachable}(T_x, \pi(x), \pi)| \leq D$, then it holds as well that x is forced.*

The proof of this lemma can be found in [17] and in [6].

2.2.3 Cost function

The main idea of the proof of the success probability of PPSZ for the multiple case, as established in [5] and described in [17] and in [6], relies on the definition of a *cost function* and on the relation between that cost function and the success probability.

Definition 2.11 (Cost function [17]) *Let α_0 be a partial and α be a total assignment and let $x \in V$ be any variable. We define the* cost of x when completing α_0 to α*, in writing $cost(\alpha_0, \alpha, x)$ as follows:*

- *If $x \notin \mathcal{U}(\alpha_0)$, then $cost(\alpha_0, \alpha, x) = 0$.*
- *If α_0 and α are incompatible, i.e. $\exists x : \{\alpha_0(x), \alpha(x)\} = \{0, 1\}$, then $cost(\alpha_0, \alpha, x) = 0$.*
- *If α does not satisfy F, then $cost(\alpha_0, \alpha, x) = 0$.*
- *Otherwise:*
  - *If $x \in V_{fo}(\alpha_0)$, then $cost(\alpha_0, \alpha, x) = 0$.*
  - *If $x \in V_{fr}(\alpha_0)$, then $cost(\alpha_0, \alpha, x) = \Pr[x \in Guessed(F, \alpha_0, \alpha, \pi, D)]$.*
  - *If $x \in V_{nf}(\alpha_0)$, then $cost(\alpha_0, \alpha, x) = S_k^{(D)}$.*

We also define the *likelihood* of an assignment:

Definition 2.12 (Likelihood of an assignment [5]) *Let $F^{[\alpha_0]}$ be satisfiable and let $\mathcal{S}_{\alpha_0}$ be the set of value assignments $l = \{x \mapsto b\}$ such that $x \in \mathcal{U}(\alpha_0)$ and $F^{[\alpha_0[l]]}$ is satisfiable. We define the random process AssignSL(F, α_0) that produces an assignment on $vbl(F)$ as follows. Start with the assignment α_0 and repeat the following step until $vbl(F^{[\alpha_0]}) = \emptyset$: Choose a value assignment $l \in \mathcal{S}_{\alpha_0}$ uniformly at random and add l to α_0. At the end, output α_0.*

Let α be a total assignment on $vbl(F)$. Then the likelihood of completing α_0 to α*, in writing $lkhd(\alpha_0, \alpha)$ is defined as the probability that AssignSL(F,α_0) returns α. For completeness, if $F^{[\alpha_0]}$ is not satisfiable, we define $lkhd(\alpha_0, \alpha) = 0$.*

Finally, we define $\text{cost}(\alpha_0, x)$, $\text{cost}(\alpha_0, \alpha)$ and $\text{cost}(\alpha_0)$:

Definition 2.13 *Let α_0 be a partial assignment over V. The* cost of x when completing α_0 to any satisfying assignment*, in writing $cost(\alpha_0, x)$ is defined as*

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

The cost of completing α_0 to α*, in writing $cost(\alpha_0, x)$, is defined as*

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

The total cost of completing α_0 to any satisfying assignment*, in writing* $cost(\alpha_0)$*, is defined as*

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

Let us state a lemma about the evolution of the cost and the likelihood when assigning a variable:

Lemma 2.14 ([17]) *Let* α_0 *and* α *be fixed and compatible. For any fixed variable* $x \in \mathcal{U}(\alpha_0)$*, if we set* x *according to* α*, then*

(i) the likelihood of α *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* $F^{[\alpha_0]}$*.*

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

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

When choosing $x \in \mathcal{U}(\alpha_0)$ *uniformly at random and setting it according to* α*, then*

(iii) the likelihood of α *increases on average as*

$$\mathbb{E}[lkhd(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)] = \left(1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}\right) lkhd(\alpha_0, \alpha)$$

(iv) the cost of a fixed variable $y \in V_{fr}(\alpha_0)$ *decreases on expectation as*

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

where

$$s = \begin{cases} 1 & \text{if } y \in V_{fr}(\alpha_0) \\ S_k^{(D)} & \text{if } y \in V_{nf}(\alpha_0) \\ 0 & \text{if } y \in V_{fo}(\alpha_0) \end{cases}$$

The proof of this lemma is available in [17] and in [6].

Chapter 3

First attempts at derandomizing PPZ and PPSZ

A brutal approach to derandomize the PPZ and PPSZ algorithms would be to consider all permutations and all binary strings corresponding to all possible runs of the algorithm. While this will work, the corresponding runtime will be completely useless, even with regard to the naive approach to test all 2^n possibilities to satisfy the formula. However, reducing the search space for the permutations and the possible binary strings while guaranteeing that a "good" combination will happen yields deterministic algorithms with a non-trivial runtime.

Some attempts at derandomizing PPZ and PPSZ following this approach have been made so far. In the original PPZ paper [12], Paturi, Pudlák and Zane propose a derandomization of the algorithm that does not achieve exactly the bounds of the randomized algorithm. Rolf [13] proposed a method to derandomize PPSZ with the same runtime as the randomized version of the algorithm when a formula is guaranteed to have at most one satisfying assignment[1]. In this chapter, we will first explain how to reduce the search space of the permutations. We will then use this tool to explain the previous attempts at derandomizing PPZ and PPSZ.

3.1 Building a permutation probability space

As in the standard analysis of the PPSZ algorithm [11], we consider the permutation as a *placement* of variables: we pick, for each variable, a real number independently and uniformly at random in the interval $[0, 1]$ and we sort the variables according to these numbers. With probability 1, any two numbers are distinct. In the next sections, we will see that the full

[1] At that time, it was not known whether the bound established for the unique satisfying assignment case also hold in general for PPSZ.

independence hypothesis can be replaced by w-wise independence. This can be achieved with smaller pools than the continuous interval $[0,1]$, and yields a much smaller set of permutations than $n!$. This section gives a method to build such a small set of permutations efficiently.

We use the following theorem, which is discussed in [1]:

Theorem 3.1 ([1],[13]) *For every n, w such that $1 \leq w \leq n$, there exists a probability space $\Omega(n,w)$ of size $\mathcal{O}(n^{w/2})$ and w-wise independent random variables $y_1, ..., y_n$ over $\Omega(n,w)$, each of which takes value 0 or 1 with probability $1/2$. $\Omega(n,w)$ can be constructed in polynomial time.*

The proof of this theorem is given in Appendix A.5 on page 66.

The following procedure is presented in [13]. Given Theorem 3.1, for given n, w and L, we can construct a probability space as follows. For each ℓ in $[L]$, draw n w-wise independent random variables $y_{1,\ell}, ..., y_{n,\ell}$ as stated in Theorem 3.1 using $\Omega(n,w)$. Define, for all x in $[n]$,

$$\pi(x) = \sum_{\ell \in [L]} 2^{-\ell} y_{x,\ell},$$

i.e. y_x is a binary encoding of $\pi(x)$ with length L. Let $A^{(L)}$ be the set of all possible real values that $\pi(.)$ can take. For a fixed x, the random variables $y_{x,.}$ are fully independent since they are drawn from independent probability spaces. Hence, every value of $A^{(L)}$ has the same probability to be chosen for $\pi(x)$. For a fixed ℓ, the random variables $y_{.,\ell}$ are w-wise independent, because they are drawn from $\Omega(n,w)$. This holds for every ℓ independently, so the values of $\pi(.)$ are w-wise independent.

The values of π are taken w-wise independently and uniformly at random over $A^{(L)}$: we call this probability space a *w-wise independent probability space for n reals with precision L*, denoted by $\Omega(n,w,L)$, where L is the number of bits defining a place for a given variable. The following lemma follows from the construction:

Lemma 3.2 *$\Omega(n,w,L)$ can be constructed with size $\mathcal{O}(n^{Lw/2})$ and in polynomial time in its size.*

3.2 Limited derandomization of PPZ

The original PPZ paper [12] gives a limited (in the sense that the runtime does not match the randomized algorithm runtime) derandomization of the PPZ algorithm. The construction of a small space of permutations is not given explicitly in the paper, and is different from $\Omega(n,w,L)$, but $\Omega(n,w,L)$

could be substituted for it. A probability space S' is assumed to be given[2]: such S' is a probability space over which n k-wise independent random variables each take values over $[m]$, where m is a prime power larger than n^3, and such that $|S'| = \mathcal{O}(n^{3k})$. $S \subseteq S'$ is then defined as the event that all the n variables take distinct values. The probability of S can be estimated to be greater than $1 - 1/n$: since $k \geq 2$, the elements are at least pairwise independent; the probability that two given elements have the same value is less than $\frac{1}{n^3}$ (because $m \geq n^3$), so by the union bound, the probability that any two elements is less than $\frac{\binom{n}{2}}{n^3}$, which yields the desired probability. We can, in that case, interpret an element of S as a permutation of the variables of the formula.

In the PPZ algorithm, a sufficient condition for a clause $\{x\}$ or $\{\bar{x}\}$ to exist at a given step is that x is the latest variable processed in a critical clause. The probability that a given variable x is last in a t-clause, with $t \leq k$, given that all variables have a unique place in the clause, is $\frac{1}{t}$. Hence, in S, we get:

$$\begin{aligned}
&\Pr_{\pi \in S}[x \text{ is last in } \pi \text{ in a } t\text{-clause}] \\
&= \Pr_{\pi \in S'}[\pi \in S \wedge x \text{ is last in } \pi \text{ in a } t\text{-clause}] \\
&= 1 - \Pr_{\pi \in S'}[x \notin S \vee x \text{ not last}] \\
&\geq 1 - (1 - \Pr_{\pi \in S'}[x \in S]) - (1 - \Pr_{\pi \in S'}[x \text{ has the (possibly multiple) last place}]) \\
&= 1 - \frac{1}{n} - 1 + \frac{1}{t} = \frac{1}{t} - \frac{1}{n}.
\end{aligned}$$

This means that, by using a permutation from the family S rather than a truly uniform permutation, a j-isolated satisfying assignment can still be encoded using at most $n - j/k + 1$ bits (using the standard PPZ machinery).

Now we observe that either there is a satisfying assignment which has few ones, or any minimal solution (with respect to the partial order defined as $\alpha_1 \prec \alpha_2$ iff, for all variable x in V, $\alpha_1(x) \leq \alpha_2(x)$) has a lot of ones. And a minimal solution must be isolated in all the directions where a variable has the value one. This dichotomy can be exploited in the sense that there are few assignments that have a small number of ones (so they can be enumerated quickly), and that a solution that is highly isolated has short codings. More precisely, it is known that the following inequality holds for $0 < \varepsilon < 1/2$ (the proof of this is stated in Appendix A.2 on page 64):

$$\sum_{i=0}^{\lfloor \varepsilon n \rfloor} \binom{n}{i} \leq 2^{nH(\varepsilon)},$$

[2]It seems plausible that the referred construction relies on polynomial hashing defined by Wegman and Carter[16]. Enumerating all possible polynomials of degree $k-1$ over the field $[m]$ would yield the desired result.

where H is the binary entropy function

$$H(\varepsilon) = -\varepsilon \log \varepsilon - (1-\varepsilon)\log(1-\varepsilon).$$

The following algorithm gathers all these ideas.

```
function dppz(F, V, ε):
    Input: F CNF formula over V, 0 < ε < 1/2
    Output: Satisfying assignment of F if F is satisfiable, ∅ otherwise
    for each assignment α with at most ε|V| ones:
        if F is satisfied by α:
            return α;
    end for
    for each permutation π in S:
        for each string β of n(1 − ε/k) + 1 bits:
            F' ← F;
            α ← ∅;
            j ← 0;
            for i ← 1 to |V|:
                x ← π(i);
                if {x} ∈ F':
                    b ← 1;
                else if {x̄} ∈ F':
                    b ← 0;
                else if j < n(1 − ε/k) + 1:
                    b ← β(j);
                    j ← j + 1;
                else:
                    break;
                α ← α ∪ {x ↦ b};
                F' ← F'[x↦b];
            end for
        end for
        if α satisfies F:
            return α;
    end for
    return ∅;
```

Theorem 3.3 ([12]) *For $0 < \varepsilon < 1/2$ such that $H(\varepsilon) = 1 - \frac{\varepsilon}{k}$, the algorithm dppz returns a satisfying assignment of a satisfiable k-CNF in $\mathcal{O}(2^{(1-\varepsilon/k)n+o(n)})$ time.*

Proof The first loop checks whether there is a satisfying assignment with at most $\varepsilon|V|$ ones. This can be done in $\mathcal{O}(2^{H(\varepsilon)n+o(n)})$ time. If no solution is found, any minimal satisfying assignment of F is at least $(\varepsilon \cdot n)$-isolated. Such an assignment has, consequently, at least εn critical variables and, for

each critical variable, at least one critical clause. Fix one critical clause per critical variable. When picking a permutation uniformly at random in S, the probability that the critical variable occurs last among the variables of the corresponding critical clause is at least $1/k - 1/n$. By linearity of expectation, the expected number of critical clauses where this happens is at least $\varepsilon n/k - \varepsilon/n$, and this bound is achieved for some permutation of S. For that permutation, the encoding of our εn-isolated assignment has length at most $n(1 - \varepsilon/k) + 1$ bits. Since we enumerate all permutations of S and all bitstrings of length $n(1 - \varepsilon/k) + 1$, we are guaranteed to find an εn-isolated assignment if it exists. Enumerating all bitstrings for all permutations requires time at most $\mathcal{O}(n^{3k}2^{n(1-\varepsilon/k)+1})$. The sum of these runtimes is minimized for $0 < \varepsilon < 1/2$ such that $H(\varepsilon) = 1 - \frac{\varepsilon}{k}$, which yields the desired result. □
The bound given here tends to $\mathcal{O}(2^{(1-1/(2k))n+o(n)})$ when k tends to infinity (to be compared with the randomized runtime of $\mathcal{O}(2^{(1-1/k)n+o(n)})$ even for small k).

3.2.1 Derandomization of PPZ in the j-isolated or unique case

If we are guaranteed that there exists a j-isolated assignment for the formula F, then we can skip the step of checking for assignments with few ones, and we can directly run through the strings of length $n(1 - j/k) + 1$ bits and be guaranteed to find a satisfying assignment if it exists. In particular, this reduces the running time to $O(2^{n-j/k+o(n)})$ for j-isolated assignments and to $O(2^{n(1-1/k)+o(n)})$ for unique or n-isolated assignments.

3.3 Derandomization of PPSZ in the unique case

In this section, we study the work of Rolf [13] and adapts it to the framework and notations defined in [17] and used in [6].
We discuss the proof of the following theorem:

Theorem 3.4 ([13]) *For a uniquely satisfiable k-CNF on n variables, integers $D > 0$, $L > 0$, there exists a deterministic algorithm dPPSZ that finds the satisfying assignment in deterministic running time at most*

$$\mathcal{O}\left(2^{-S_k n+\varepsilon_{k,D,L} n}\right),$$

where $S_k = \int_0^1 \frac{t^{1/(k-1)}-t}{1-t}\mathrm{d}t$ and

$$\lim_{L\to\infty}\lim_{D\to\infty}\varepsilon_{k,D,L} = 0.$$

This theorem readily implies the following:

Theorem 3.5 ([13]) *For a uniquely satisfiable 3-CNF resp. 4-CNF on n variables, the satisfying assignment can be found in deterministic running time at most $\mathcal{O}(1.3071^n)$ resp. $\mathcal{O}(1.4699^n)$.*

3.3.1 Properties of the random permutation

In the randomized PPSZ analysis, we consider a permutation as a set of values picked independently and uniformly at random from the interval $[0,1]$ and ordering the variables according to these values. In this section, we consider what happens if, instead, we define a permutation π from the mapping defined when picking places from $\Omega(n,w,L)$: we define it as "u comes before v if $\pi(u) < \pi(v)$", breaking ties arbitrarily.
In the standard PPSZ version, since we pick the place of elements uniformly at random independently from $[0,1]$, the probability, for a fixed variable x, that some variable u comes before x in the permutation is exactly the place of x. This is not true anymore in $\Omega(n,w,L)$: first, it can happen that x and u end up having the same place (since the set $A^{(L)}$ is finite) and, even if the elements are all distinct, the probability that u comes before x itself is not exactly $\pi(x)$ anymore. We have the following situation, for a set of variables B that would correspond to a set of variables of a CNF formula:

Lemma 3.6 ([13]) *Let $x \in [n]$ be a variable and $B \subseteq [n]$ a set of variables with $|B| < w$ and $x \notin B$. Given $\pi \in_{u.a.r.} \Omega(n,w,L)$, then the following are true:*

1. *The probability over $\Omega(n,w,L)$ that the value of x is unique is $(1-2^{-L})^{|B|}$.*
2. *Given that x has a unique value, all $\pi(u)$ with $u \in B$ are independent, and for each $u \in B$, the probability that $\pi(u) < \pi(x)$ (i.e. that u comes before x in the random permutation) is equal to $\pi(x) \cdot 2^L/(2^L-1)$.*

Proof Since $x \notin B$, the probability that x doesn't get attributed the same value as any element of B is $(1-2^{-L})^{|B|}$ because the probability that any given element of B has the same value is $(1-2^{-L})$, and $|B| < w$ so all the values are independent.
Now we condition on the fact that no element of B has the same value as x. All elements can still be seen as being drawn independently at random from $A^{(L)} \setminus \{\pi(x)\}$. Moreover, there is exactly $\pi(x) \cdot 2^L$ elements of $A^{(L)} \setminus \{\pi(x)\}$ that are strictly less than $\pi(x)$, and a total of $2^L - 1$ elements total in that set, to be chosen uniformly at random. Hence, the probability that a given element u comes before x in B is $\pi(x) \cdot 2^L/(2^L-1)$. □

3.3.2 Reachable nodes in critical clause trees

The initial proof from Rolf [13] uses, as in the analysis of the unique PPSZ case [11], resolution and cuts in critical clause trees to prove the bounds of the unique case derandomization. The version of the proof that we give here translates these notions to the notions of D-implication (as in [5]) and reachable nodes in the critical clause trees (as in [17]), but is otherwise the same proof.
We have given in Section 2.2.2 on page 11 the definition of critical clause trees and the definition of reachable nodes. We also recall here the crucial

lemma that allows to relate the reachable nodes to the fact that a variable is forced:

Lemma 2.10 ([17]) *If we have $|Reachable(T_x, \pi(x), \pi)| \leq D$, then it holds as well that x is forced.*

The main idea of the derandomization is, as in the PPZ case, that we do not need full independence over the places of the variables to prove that the set of reachable variables is small. In particular, if the resulting tree has height at most $\log_{k-1} D - 1$, then it is guaranteed that said resulting tree will have less than D reachable nodes. The crucial observation there is that, to check whether a tree, after removing nodes, has height at most $\log_{k-1} D - 1$, it is enough to limit the depth of the original tree to $\log_{k-1} D$, which corresponds to $\frac{D(k-1)-1}{k-2}$ nodes in total at most. Let us denote $w = \frac{D(k-1)-1}{k-2}$. Instead of building full critical clause trees, we limit the depth of these to $\log_{k-1} D$, and we use permutations from $\Omega(n, w, L)$ on these trees. On such a tree, since the permutations are w-wise independent sets of n variables, all the places are picked independently of each other.

Hence, we can consider a critical tree T_x that is limited at depth $\log_{k-1} D$, and we delete every node whose place is less than the place of x. We define $r = \pi(x) \cdot 2^L/(2^L - 1)$ (corresponding to the probability that the place of the var-label of a given node is less than the place of x), and, for any subtree T_0 of T that has root v, we define $Q_{T_0}(r)$ the probability that the tree T_0 has height $h(T_0)$ such that $h(v) + h(T_0) \leq \log_{k-1} D - 1$. The beginning of the proof of the lower bound for $Q_{T_0}(r)$ is similar to the one for the general case [11]; since our "usual" proof for the PPSZ unique case is slightly different in [17] and [6], we give it here.

Lemma 3.7 ([13], adapted) *Given a critical tree T, where every node is deleted if its place is less than the place of x, let T_0 be some subtree of T with more than one node and let $T_1, ..., T_t$ be the subtrees rooted at the children of the root of T_0. Let $u_1, ..., u_t$ be their roots. Then it is true that*

$$Q_{T_0}(r) \geq \prod_{i=1}^{t} (r + (1-r)Q_{T_i}(r))$$

holds, where the empty product is interpreted as 1.

Proof For each tree $T_1, ..., T_t$, the root of T_i is deleted with probability r, or T_i must be of height $h(u_i)$ such that $h(u_i) + h(T_i) \leq \log_{k-1} D - 1$, which happens with probability $Q_{T_i}(r)$. For each of these subtrees, since they are part of critical clause trees, the other var-labels of the tree are different from u_i. Hence, these events are independent and, for each tree T_i, the probability that this subtree is compatible with the desired total height is $r + (1-r)Q_{T_i}(r)$.

We denote by K_i the event "the subtree T_i is compatible with the desired total height". We can then use the FKG inequality (see Appendix A.3 on page 64) in the same fashion as in the standard PPSZ analysis ([17]) to conclude that

$$Q_{T_0}(r) = \Pr\left[\bigcap_{i=1}^{t} K_i\right] \geq \prod_{i=1}^{t} r + (1-r)Q_{T_i}(r),$$

which concludes our proof. □

By multiplying this probability with the probability that the places are, indeed, different from the place of x, this implies the following corollary, since the tree contains at most w nodes:

Corollary 3.8 *[13] Given a critical clause tree T limited to depth $\log_{k-1} D$ with root labeled by x and given that $\pi(x) = p$, the probability that the height of T after deleting the nodes whose place is before x is less than $\log_{k-1} D - 1$ is at least $Q'_T(p)$, with*

$$Q'_T(p) = Q_T(p \cdot 2^L/(2^L - 1)) \cdot (1 - 2^L)^{w-1}.$$

This probability is defined for a fixed p; now we need to consider the unconditional probability over the whole domain $p \in A^{(L)}$. This yields:

$$2^{-L} \sum_{l=0}^{2^L-1} Q'_T(l/2^L) = 2^{-L} \sum_{l=0}^{2^L-1} Q_T\left(\frac{l}{2^L - 1}\right) \cdot (1 - 2^{-L})^{w-1}.$$

If L grows to infinity, this yields:

$$\begin{aligned}
&\lim_{L\to\infty} 2^{-L} \sum_{l=0}^{2^L-1} Q_T\left(\frac{l}{2^L - 1}\right) \cdot (1 - 2^{-L})^{w-1} \\
&= \lim_{L\to\infty} (1 - 2^{-L})^{w-1} \sum_{l=0}^{2^L-1} \frac{1}{2^L} Q_T\left(\frac{l}{2^L - 1}\right) \\
&= \lim_{L\to\infty} (1 - 2^{-L})^{w-1} \sum_{l=0}^{2^L-1} \frac{1}{2^L - 1} Q_T\left(\frac{l}{2^L - 1}\right) \\
&= \int_0^1 Q_T(p)\,\mathrm{d}p,
\end{aligned}$$

by definition of the Riemann integral, since it has been proven in [11] that this integral exists (which allows us to do the above computation) and that it tends to $1 - S_k$ when D grows to infinity. For D going to infinity, the difference between the sum and $1 - S_k$ can be made arbitrary small by choosing L large enough. A slowly growing function L still yields a sub-exponential construction of $\Omega(n, w, L)$. We denote by $\varepsilon_{k,D,L}$ the difference between the sum and $1 - S_k$.

3.3.3 A derandomized algorithm

The previous section allowed us to establish that, when picking permutations from $\Omega(n, w, L)$, with L and D tending to infinity, the probability that a cut happened in a critical clause tree was at least $\int_0^1 Q_T(p)\mathrm{d}p = 1 - (S_k - \varepsilon_{k,D,L})$. If a cut happens, then the variable corresponding to the critical clause tree is forced. By linearity of expectation, the expected number of forced variables when picking permutations uniformly at random from $\Omega(n, w, L)$ is $n(1 - S_k + \varepsilon_{k,D,L})$. Hence, there is a permutation in $\Omega(n, w, L)$ for which this number is achieved. When a variable is forced, no random bit is consumed. Hence, for a "good" permutation and a "good" choice of random bits, we will consume at most $(S_k + \varepsilon_{k,D,L})n$ bits. Considering bitstrings of that length only is consequently sufficient, which allows us to state the following derandomized algorithm:

function `dppsz`(F, V, L, D)**:**
  Input: F uniquely satisfied CNF formula over V, L and $D \in \mathbb{N}_0$
  Output: Satisfying assignment of F
  for each $\pi \in \Omega(|V|, \frac{D(k-1)-1}{k-2}, L)$**:**
    for each $\beta \in \{0,1\}^{\lceil n(S_k+\varepsilon_{k,D,L})\rceil}$**:**
      $F' \leftarrow F$;
      $\alpha \leftarrow \emptyset$;
      $j \leftarrow 0$;
      for $i \leftarrow 1$ **to** $|V|$**:**
        $x \leftarrow \pi(i)$;
        if $F' \vDash (x \mapsto 1)$**:**
          $b \leftarrow 1$;
        else if $F' \vDash (x \mapsto 0)$**:**
          $b \leftarrow 0$;
        else if $j < \lceil n(S_k + \varepsilon_{k,D,L})\rceil$**:**
          $b \leftarrow \beta[j]$;
          $j \leftarrow j + 1$;
        else:
          break;
        $\alpha \leftarrow \alpha \cup \{x \mapsto b\}$;
        $F' \leftarrow F'^{[x \mapsto b]}$
      end for
      if α *satisfies* F**:**
        return α;
    end for
  end for

The correctness of the algorithm follows from the analysis of this section. The runtime can be computed as follows:

- we can construct and enumerate $\Omega(n, \frac{D(k-1)-1}{k-2}, L)$ in time $\mathcal{O}(n^{Lw/2})$,

which stays subexponential for L and D slowly growing functions of n,

- we can enumerate all bit strings of length $\lceil n(S_k + \varepsilon_{k,D,L})\rceil$ in time $\mathcal{O}(2^{\lceil n(S_k+\varepsilon_{k,D,L})\rceil+o(n)})$,
- we can check whether F' D-implies a literal in subexponential time for D a slowly growing function of n.

This concludes the proof of the theorem.

Part II

Elements for the derandomization of PPSZ

Chapter 4

Derandomization plan

Our derandomization plan was proposed by Timon Hertli and Robin Moser in private communications and discussions. They made a number of suggestions and hypotheses that directed the core of this thesis.

4.1 Main ideas for derandomization

There exists a trivial way to derandomize PPSZ: enumerate all permutations of variables and all bitstrings of size n, and check if PPSZ returns a satisfying assignment. By doing so, if a formula is satisfiable, then a satisfying assignment will necessarily be found; and if no satisfying assignment is found, we are guaranteed that the formula is unsatisfiable. This is, however, very inefficient: the runtime for such an algorithm would be $\Omega(n!2^n)$, and it is actually enough to enumerate all possible bitstrings (in $\Theta(2^n)$ time) to guarantee these properties: this would represent a superexponential speed-up compared to our trivial derandomization.

However, this derandomization can be improved in a number of ways. First, as we have seen in Section 3.3, Rolf [13] proved that, if the formula was guaranteed to have a unique satisfying assignment, it was enough to consider a sub-exponential number of permutations and smaller bitstrings to be guaranteed that a "good" permutation and a "good" bitstring will be encountered if such exists.

Following the same lines, we try to see where random bits can be saved, and hope that enumerating the remaining ones that we are not able to save will yield non-trivial runtimes.

The first conjecture is that, as in the unique case, it is enough to consider d-wise independence, for some value of d (ideally constant), in lieu of full independence for the choice of the permutation. If this is true and if d is small enough, then we will be able to enumerate a set of such permutations in (hopefully) subexponential time or (if needed) low exponential time.

This would take care of the $n!$ factor in the runtime of the derandomized algorithm.
The other random bits are coming from the fact that, when the value of a variable is not forced, a random bit is used. However, there are two ways for a variable to not be forced: it can be frozen but not forced, or it can be non-frozen. If a variable is frozen but not forced, there is not much that can be done except trying to guess it, hence the consumption of a random bit. On the other hand, if a variable is non-frozen, then whatever choice is made is a valid choice – it will only steer the algorithm towards an assignment or another. Hence, if we choose, to set any non-frozen variable to a deterministic value, this allows us to save the random bits corresponding to the non-frozen variables, while hopefully not degrading the probability of returning a satisfying assignment too much.
It makes sense, as a first step, assume that we are given an oracle that allows us to set non-frozen variables to 0 when we encounter them. The main question here is how would that setup impact the success probability of PPSZ. If the success probability of PPSZ with oracle can be established, such an oracle must also be built and run in a subexponential or low exponential runtime to be at all useful – which might be possible for a slightly weaker oracle.

4.2 Questions explored in this thesis

For the aforementioned plan to come together, a number of questions need to be answered. We will divide them in the four following chapters, which are organized as follows.
Chapter 5 is an alternative proof of the runtime bound for PPSZ in the multiple satisfying assignments case. Instead of considering the probability of success for all assignments at once, we prove a lower bound for the probability that a *s*pecific satisfying assignment is returned. By disentangling the assignments from each other, we hope to facilitate further progress on the PPSZ derandomization a bit.
In Chapter 6, we try to answer the question "Can PPSZ be run with the same success probability if we consider a small subset of permutations instead of all possible permutations?". As we will see, this question is still open; we have however gained some insight on the reasons why this may be difficult to prove.
In Chapter 7, we present our work on the question of what happens if we introduce an oracle that sets non-frozen variables to 0: we are interested in the probability of success of PPSZ with such an oracle. We prove, as a first step, that the success probability of PPZ stays bounded by the same value with or without an oracle. We then show that, for PPSZ, if the probability of a frozen variable to be guessed stays the same (or becomes larger) when

using the oracle, then we also get the same bound for the multiple satisfying assignment case whether we use the oracle or not. We also raise the question of whether such an oracle would be at all realistic, and we suggest some directions that may be interesting to pursue on that matter.
Finally, in Chapter 8, we will wrap up our findings and explain what still needs to be done to derandomize PPSZ according to our plan. We explain some of our thoughts on the matter of the oracle construction, and we formulate some questions that may guide future research on the topic of the derandomization of PPZ and PPSZ.

Chapter 5

Probability of returning a given assignment

The original proof of the PPSZ bound for the multiple satisfying assignments case [5] relies on the fact that, over the whole set of assignments, the likelihood and the cost of the assignments compensate nicely, and that allows us to conclude that the bound actually holds. The success probability of the algorithm is computed over all possible satisfying assignments. The probability of returning a *given* satisfying assignment was however, so far, an open question.
In this chapter, we prove the following theorem:

Theorem 5.1 *Let α_0 s.t. $F^{[\alpha_0]}$ is satisfiable. Then the probability that PPSZ outputs a given satisfying assignment α when starting in state α_0 is at least $lkhd(\alpha_0, \alpha) \cdot 2^{-cost(\alpha_0,\alpha)}$.*

Proof In what follows, we consider a fixed satisfying assignment α. This proof is by induction; we suppose that the claim holds for all α_0 that fix a larger number of variables. If α_0 is total, then the statement holds trivially.
Let us denote by $p(\alpha_0, \alpha)$ the probability that PPSZ outputs α when starting from state α_0. Let x and b be random variables: $x \in \mathcal{U}(\alpha_0)$ u.a.r., and b is the forced value by D-implication if $x \in \mathrm{V_{fo}}(\alpha_0)$ and u.a.r. otherwise. We have:

$$\begin{aligned}
&p(\alpha_0, \alpha) \\
&= \Pr_{x \in_{\text{u.a.r}} \mathcal{U}(\alpha_0); b}[(x, b) \in \alpha \wedge \mathrm{PPSZ}(F, V, \alpha_0 \cup \{x \mapsto b\}, D) \text{ returns } \alpha] \\
&= \Pr_{x \in_{\text{u.a.r}} \mathcal{U}(\alpha_0); b}[(x, b) \in \alpha] \cdot \mathop{\mathbb{E}}_{x \in_{\text{u.a.r}} \mathcal{U}(\alpha_0); b}[p(\alpha \cup \{x \mapsto b\}, \alpha) \mid (x, b) \in \alpha] \\
&= \frac{n(\alpha_0) + |\mathrm{V_{fo}}(\alpha_0)|}{2n(\alpha_0)} \mathop{\mathbb{E}}_{x \in_{\text{u.a.r}} \mathcal{U}(\alpha_0); b}[p(\alpha \cup \{x \mapsto b\}, \alpha) \mid (x, b) \in \alpha].
\end{aligned}$$

We now relate the expectation over "x, then b" to the expectation over $(x, b) \in \alpha$ (or, equivalently, "choose x u.a.r. in $\mathcal{U}(\alpha_0)$ and set it to $\alpha(x)$").

The difference between these two expectations is that, in the second expectation, the weight of variables in $\mathrm{V}_{\mathrm{fo}}(\alpha_0)$ is twice the weight of the other variables; we define

$$w(x) = \begin{cases} 2 & \text{if } x \in \mathrm{V}_{\mathrm{fo}}(\alpha_0) \\ 1 & \text{otherwise} \end{cases}.$$

To then form the expectation, we need a normalization factor so that the probabilities sum to 1. The normalization factor for these weights is $\frac{n(\alpha_0)}{|n+\mathrm{V}_{\mathrm{fo}}(\alpha_0)|}$ and so we get

$$\begin{aligned} p(\alpha_0, \alpha) &= \frac{n + |\mathrm{V}_{\mathrm{fo}}(\alpha_0)|}{2n(\alpha_0)} \mathop{\mathbb{E}}_{x \in_{\text{u.a.r.}} \mathcal{U}(\alpha_0)} \left[\frac{n(\alpha_0) \cdot w(x)}{n + |\mathrm{V}_{\mathrm{fo}}(\alpha_0)|} \cdot p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \right] \\ &= \frac{1}{2} \mathop{\mathbb{E}}_{x \in \text{u.a.r.} \mathcal{U}(\alpha_0)} [w(x) \cdot p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)]. \end{aligned}$$

In the usual analysis, here we would apply the induction hypothesis and use Jensen's inequality before continuing the computation. This approach does not seem to work as well as in the general case. This failure can be explained by the presence of a likelihood term in the expression when applying directly the induction hypothesis. The likelihood is not very concentrated, and we can observe in some cases "jumps" when fixing a variable. Consider for instance the case of a formula with the following satisfying assignments: 000...000 (the all-zero assignment), and all the assignments of type 100..000, 010...000, ..., 000...001 (all the assignments that have exactly one 1). The likelihood of the all-zero assignment is 2^{-n} (we need to choose 0 for all the n variables); by symmetry, the likelihood of any other assignment is $(1 - 2^{-n})/n$. However, when setting any variable to 1, the likelihood of the corresponding assignment becomes 1. As discussed in [14] (Section 6.3), "As a rule of thumb, Jensen's inequality is pretty tight if [the considered random variable] is very concentrated around its expectation".
To take care of this issue, we introduce a new distribution over the variables of $\mathcal{U}(\alpha_0)$ which brings the likelihood inside the probability distribution. By moving the likelihood into the distribution used for the expectation, we hope that the likelihood will cancel out thanks to the induction hypothesis and that things will work out as desired. We denote by ξ the distribution over the variables of $\mathcal{U}(\alpha_0)$ weighted, for any variable x, according to $\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)$. We have that

$$\begin{aligned} \mathrm{lkhd}(\alpha_0, \alpha) &= \mathop{\mathbb{E}}_{(x,b) \in_{\text{u.a.r.}} \mathcal{S}_{\alpha_0}} \mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \\ &= \sum_{x \in \mathcal{U}(\alpha_0)} \frac{1}{|\mathcal{S}_{\alpha_0}|} \mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha), \end{aligned}$$

which allows us to determine the sum of all the weights, and so the probability of any variable x for the distribution ξ is given by

$$p_{\tilde{\xi}}(x) = \frac{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)}.$$

We now want to introduce this probability distribution in our current expression for $p(\alpha_0, \alpha)$:

$$\begin{aligned}
&p(\alpha_0, \alpha) \\
&= \frac{1}{2} \mathop{\mathbb{E}}_{x \in_{\text{u.a.r.}} \mathcal{U}(\alpha_0)} [w(x) \cdot p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)] \\
&= \frac{1}{2} \sum_{x \in \mathcal{U}(\alpha_0)} \frac{w(x)}{n(\alpha_0)} \cdot p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \\
&= \frac{1}{2} \sum_{x \in \mathcal{U}(\alpha_0)} \frac{w(x)}{n(\alpha_0)} \cdot \frac{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)} \cdot \frac{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)}{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)} \\
&\cdot p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \\
&= \frac{1}{2} \sum_{x \in \mathcal{U}(\alpha_0)} \frac{w(x) \cdot |\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)} \\
&\cdot \frac{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)} \cdot \frac{p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)} \\
&= \frac{1}{2} \frac{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)} \mathop{\mathbb{E}}_{x \in_{\tilde{\xi}} \mathcal{U}(\alpha_0)} \left[\frac{w(x) \cdot p(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)} \right].
\end{aligned}$$

We apply the induction hypothesis and we get

$$\begin{aligned}
&p(\alpha_0, \alpha) \\
&\geq \frac{1}{2} \frac{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)} \\
&\cdot \mathop{\mathbb{E}}_{x \in_{\tilde{\xi}} \mathcal{U}(\alpha_0)} \left[\frac{w(x) \cdot \mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \cdot 2^{-\mathrm{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}}{\mathrm{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)} \right] \\
&= \frac{1}{2} \frac{|\mathcal{S}_{\alpha_0}| \cdot \mathrm{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)} \mathop{\mathbb{E}}_{x \in_{\tilde{\xi}} \mathcal{U}(\alpha_0)} \left[w(x) \cdot 2^{-\mathrm{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)} \right].
\end{aligned}$$

Now that we have gotten rid of the likelihood in the expectation, we use Jensen's inequality (see Appendix A.1 on page 63) with the convex function $x \mapsto 2^{-x}$ and we get:

$$p(\alpha_0, \alpha) \geq \mathrm{lkhd}(\alpha_0, \alpha) 2^{-1 + \log \frac{|\mathcal{S}_{\alpha_0}|}{n(\alpha_0)} + \mathbb{E}_{x \in_{\tilde{\xi}} \mathcal{U}(\alpha_0)} [\log w(x)] + \mathbb{E}_{x \in_{\tilde{\xi}} \mathcal{U}(\alpha_0)} [-\mathrm{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)]}.$$

We will then need the following lemma:

Lemma 5.2 *For a fixed α, we have*

$$\mathop{\mathbb{E}}_{x\in_{\xi}\mathcal{U}(\alpha_0)}[cost(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)]\leq cost(\alpha_0,\alpha)-\frac{|V_{fr}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|}-\frac{2S_k^{(D)}|V_{nf}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|}.$$

Proof We have, by definition of the expectation:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{x\in_{\xi}\mathcal{U}(\alpha_0)}[\mathrm{cost}(\alpha_0\cup\{x\mapsto\alpha(x)\})]\\
&=\sum_{x\in\mathcal{U}(\alpha_0)}\frac{\mathrm{lkhd}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)}{|\mathcal{S}_{\alpha_0}|\cdot\mathrm{lkhd}(\alpha_0,\alpha)}\cdot\mathrm{cost}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)\\
&=\frac{n(\alpha_0)}{|\mathcal{S}_{\alpha_0}|\cdot\mathrm{lkhd}(\alpha_0,\alpha)}\\
&\cdot\mathop{\mathbb{E}}_{x\in_{\mathrm{u.a.r.}}\mathcal{U}(\alpha_0)}[\mathrm{lkhd}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)\cdot\mathrm{cost}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)].
\end{aligned}$$

To work with this expression, we invoke a correlation inequality that is used in [17] and [6] (we defer its proof to Appendix A.6 on page 67):

Lemma 5.3 ([17]) *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 $\mathbb{E}[A]=\bar{a}$ and $\mathbb{E}[B]=\bar{b}$. Then*

$$\mathbb{E}[A\cdot B]\leq a\bar{b}+b\bar{a}-ab.$$

Using Lemma 2.14 on page 14, and by defining $A=\mathrm{lkhd}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)$, $a=\mathrm{lkhd}(\alpha_0,\alpha)$, $\bar{a}=\left(1+\frac{|\mathrm{V_{nf}}(\alpha_0)|}{n(\alpha_0)}\right)\mathrm{lkhd}(\alpha_0,\alpha)$, $B=\mathrm{lkhd}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)$, $b=\mathrm{cost}(\alpha_0,\alpha)$, $\bar{b}\leq\mathrm{cost}(\alpha_0,\alpha)-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{n(\alpha_0)}-\frac{S_k^{(D)}|\mathrm{V_{nf}}(\alpha_0)|}{n(\alpha_0)}$, we get

$$\begin{aligned}
&\mathop{\mathbb{E}}_{x\in_{\mathrm{u.a.r}}\mathcal{U}(\alpha_0)}[\mathrm{lkhd}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)\cdot\mathrm{cost}(\alpha_0\cup\{x\mapsto\alpha(x)\},\alpha)]\\
&\leq\mathrm{lkhd}(\alpha_0,\alpha)\cdot\left(\mathrm{cost}(\alpha_0,\alpha)-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{n(\alpha_0)}-\frac{S_k^{(D)}|\mathrm{V_{nf}}(\alpha_0)|}{n(\alpha_0)}\right)\\
&+\mathrm{cost}(\alpha_0,\alpha)\cdot\left(1+\frac{|\mathrm{V_{nf}}(\alpha_0)|}{n(\alpha_0)}\right)\cdot\mathrm{lkhd}(\alpha_0,\alpha)\\
&-\mathrm{cost}(\alpha_0,\alpha)\cdot\mathrm{lkhd}(\alpha_0,\alpha)\\
&=\mathrm{lkhd}(\alpha_0,\alpha)\cdot\left(\frac{|\mathcal{S}_{\alpha_0}|}{n(\alpha_0)}\mathrm{cost}(\alpha_0,\alpha)-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{n(\alpha_0)}-\frac{S_k^{(D)}|\mathrm{V_{nf}}(\alpha_0)|}{n(\alpha_0)}\right).
\end{aligned}$$

Plugging this in our expression yields the desired result. □

Back to our main computation, let us deal with the $\mathbb{E}[\log w(x)]$ term. Observe that

$$\mathop{\mathbb{E}}_{\xi}[\log w(x)]=\Pr_{\xi}[w(x)=2]=\Pr_{\xi}[x\text{ is forced}].$$

When x is forced, $\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) = \text{lkhd}(\alpha_0, \alpha)$, and so

$$\mathop{\mathbb{E}}_{\xi}[\log w(x)] = \frac{|\mathrm{V}_{\mathrm{fo}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|}.$$

Finally, we deal with the $\log \frac{|\mathcal{S}_{\alpha_0}|}{n(\alpha_0)}$ in the same way than in the standard PPSZ proof, by using the following inequality (see Appendix A.4 on page 66):

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

We have:

$$\begin{aligned}
\log \frac{|\mathcal{S}_{\alpha_0}|}{n(\alpha_0)} &= \log \frac{n(\alpha_0) + |\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{n(\alpha_0)} \\
&= \log\left(1 + \frac{|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{n(\alpha_0)}\right) \\
&\geq \log(e)\frac{|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{n(\alpha_0) + |\mathrm{V}_{\mathrm{nf}}(\alpha_0)|} \\
&= \log(e)\frac{|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|}.
\end{aligned}$$

Putting all these elements together, we get that

$$p(\alpha_0, \alpha) \geq \text{lkhd}(\alpha_0, \alpha) \cdot 2^{-1 + \frac{|\mathrm{V}_{\mathrm{fo}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} + \log(e)\frac{|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} - \text{cost}(\alpha_0,\alpha) + \frac{|\mathrm{V}_{\mathrm{fr}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} + \frac{2S_k^{(D)}|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|}}.$$

For our theorem to hold, we need that

$$-1 + \frac{|\mathrm{V}_{\mathrm{fo}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} + \log(e)\frac{|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} + \frac{2S_k^{(D)}|\mathrm{V}_{\mathrm{nf}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} + \frac{|\mathrm{V}_{\mathrm{fr}}(\alpha_0)|}{|\mathcal{S}_{\alpha_0}|} \geq 0.$$

This is true if $\log(e) + 2S_k^{(D)} \geq 2$; since $S_k^{(D)} \geq S_3^{(D)}$ and $\log(e) + 2S_3^{(D)} > 1.44 + 2 \cdot 0.38 > 2$, we are done. □

This theorem allows us to get an alternative proof for the overall bounds for PPSZ. We have:

$$\begin{aligned}
\Pr[\text{PPSZ succeeds}] &= \sum_{\alpha \in \text{sat}(F)} \Pr[\text{PPSZ returns } \alpha] \\
&\geq \sum_{\alpha \in \text{sat}(F)} \text{lkhd}(\alpha_0, \alpha) 2^{-\text{cost}(\alpha_0,\alpha)}.
\end{aligned}$$

Since the likelihood can be seen as a probability distribution over the assignments, we can see this expression as an expectation over α picked according to this distribution:

$$\Pr[\text{PPSZ succeeds}] \geq \mathop{\mathbb{E}}_{\alpha}\left[2^{-\text{cost}(\alpha_0,\alpha)}\right],$$

and apply Jensen to this:

$$\begin{aligned}\Pr[\text{PPSZ succeeds}] &\geq 2^{-\mathbb{E}[\text{cost}(\alpha_0,\alpha)]} \\ &= 2^{-\sum_{\alpha\in\text{sat}(F)} \text{lkhd}(\alpha_0,\alpha)\text{cost}(\alpha_0,\alpha)} \\ &= 2^{-\text{cost}(\alpha_0)} \geq 2^{-S_k^{(D)}n}.\end{aligned}$$

Chapter 6

PPSZ with a smaller permutation set

As we have previously observed, the derandomization of PPSZ needs to address two sources of randomness: the random permutation used to run the algorithm, and the random bits used to build the assignment when the processed variable is not forced. A natural question, before looking at the global derandomization plan, is to ask whether PPSZ itself can run with a smaller subset of permutations than the complete permutation set, and what conditions would be sufficient (and/or necessary) for such a permutation set for the proof to go through. We investigate these questions in this chapter.

6.1 The set $\Omega(n, w, L)$

The derandomization of PPSZ in the unique case by Rolf [13] uses a construction for a "w-wise independent probability space for n variables with precision L", denoted by $\Omega(n, w, L)$, and described in Section 3.1 on page 15. The use of this set of permutations does not, however, directly translate into a proof for the multiple satisfying assignments case. In this section, we will explain two problems that we have identified: the probability that a variable is forced, and the variations in the likelihood.

6.1.1 Probability of a variable to be guessed

The proof of Rolf shows that, for a variable that is frozen at the beginning of the algorithm, the probability that it is guessed is

$$\Pr_{\pi \in \Omega(n,w,L)}[x \in \text{Guessed}(F, \alpha_0, \alpha, \pi, D)] \leq S_k^{(D)}.$$

What happens when a variable is unfrozen at the beginning of the algorithm but becomes frozen after some steps is, however, unclear. Giving α_0 as a parameter restricts the set of permutations to the ones that start with the variables of α_0; we haven't found any compelling argument to exclude

the possibility that a partial assignment that freezes a variable would also be only compatible with permutations such that the probability that said variable is guessed is greater than $S_k^{(D)}$.
Another possible issue is that, even if the probability of a variable being guessed does not get larger than $S_k^{(D)}$, the proof of the previous chapter (and the original proof of the PPSZ success probability) relies on the fact that the cost never increases. When considering, as in Lemma 2.14 on page 14, $\text{cost}(\alpha_0, \alpha, y)$ and $\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)$, the cost of a single variable could increase when adding a single other variable to the mix, because the probability that a variable is guessed could increase when considering subsets of permutations that are compatible, respectively, with α_0 and with $\alpha_0 \cup \{x \mapsto \alpha(x)\}$. Defining an order for α_0 would help here (because we would not re-add permutations to the set of compatible permutations), but it would not for the previous issue.

6.1.2 Likelihood evolution

The usual proof for all permutations also uses the fact that the likelihood of a satisfying assignment cannot decrease when adding a compatible variable assignment. Specifically, it relies on the fact that, for a fixed variable x, we have that $\text{lkhd}(\alpha_0, \alpha) \leq \text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)$. The likelihood as defined in the standard proof does not correspond to the case of picking permutations in $\Omega(n, w, L)$. In particular, $\mathbb{E}_{\pi \in \Omega(n,w,L)} \text{lkhd}(\alpha_0, \alpha)$ does not seem to make much sense. To correct this, we introduce the notion of Ω-likelihood. In what follows, we will consider that α_0 is an *ordered* partial assignment, that is to say that we also consider the order in which the variables are assigned as a part of a partial assignment.

Definition 6.1 *Let $F^{[\alpha_0]}$ be satisfiable and let $\mathcal{S}_{\alpha_0}$ be the set of value assignments $l = (x \mapsto b)$ such that $x \in \mathcal{U}(\alpha_0)$ and $F^{[\alpha_0[l]]}$ is satisfiable. To each of these elements, we assign a weight equal to the probability that x is chosen next in Ω, conditioned on the fact that the permutation starts with the variables of α_0 (in its order); and we define a probability distribution Ψ over $\mathcal{S}_{\alpha_0}$ according to these weights. We will denote these weights $w_{\alpha_0}(l)$ (or $w_{\alpha_0}(x, b)$ for every value assignment $l = (x \mapsto b)$). We define the random process $\text{AssignSL}^{\Omega}(F, \alpha_0)$ that produces an assignment of $\text{vbl}(F)$ as follows. Start with the assignment α_0, and repeat the following step until $\text{vbl}(F^{[\alpha_0]}) = \emptyset$: Choose a value assignment $l \in \mathcal{S}_{\alpha_0}$ according to Ψ and add l to α_0. At the end, output α_0. Let α be a total assignment on $\text{vbl}(F)$. Then the* Ω-likelihood of completing α_0 to α, *in writing $\text{lkhd}^{\Omega}(\alpha_0, \alpha)$, is defined as the probability that $\text{AssignSL}^{\Omega}(F, \alpha_0)$ returns α. For completeness, if $F^{[\alpha_0]}$ is not satisfiable, or if α_0 is chosen such that no variable x can be chosen next in Ω, we define $\text{lkhd}^{\Omega}(\alpha_0, \alpha) = 0$.*
We can now try to prove by induction that $\text{lkhd}^{\Omega}(\alpha_0, \alpha) \leq \text{lkhd}^{\Omega}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)$. We suppose that the statement is true for large assignments α_0 (if

α_0 is a complete assignment, the statement holds trivially). We try to express $\mathrm{lkhd}^\Omega(\alpha_0, \alpha)$ as an expression depending on $\mathrm{lkhd}^\Omega(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)$ by separating the $\mathrm{lkhd}^\Omega(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)$ from the rest of the expression when applying the definition of the expectation of the likelihood over all literals (which is equal to $\mathrm{lkhd}^\Omega(\alpha_0, \alpha)$. In the end, this will fail because we do not know how to relate the weight of a variable x' with regard to α_0 and with regard to $\alpha_0 \cup \{x \mapsto \alpha(x)\}$. Formally, we have:

$$
\begin{aligned}
&\mathrm{lkhd}^\Omega(\alpha_0, \alpha) \\
&= \mathop{\mathbb{E}}_{(x',b') \in_\Psi \mathcal{S}_{\alpha_0}} \left[\mathrm{lkhd}^\Omega(\alpha_0 \cup \{x' \mapsto b'\}, \alpha) \right] \\
&= \sum_{(x',b') \in \mathcal{S}_{\alpha_0}} \frac{w_{\alpha_0}(x', b')}{\sum_{(x'',b'') \in \mathcal{S}_{\alpha_0}} w_{\alpha_0}(x'', b'')} \mathrm{lkhd}^\Omega(\alpha_0 \cup \{x' \mapsto b'\}, \alpha) \\
&= \sum_{x' \in \mathcal{U}(\alpha_0)} \frac{w_{\alpha_0}(x', \alpha(x'))}{\sum_{(x'',b'') \in \mathcal{S}_{\alpha_0}} w_{\alpha_0}(x'', b'')} \mathrm{lkhd}^\Omega(\alpha_0 \cup \{x' \mapsto \alpha(x')\}, \alpha) \\
&= \frac{w_{\alpha_0}(x, \alpha(x))}{\sum_{(x'',b'') \in \mathcal{S}_{\alpha_0}} w_{\alpha_0}(x'', b'')} \mathrm{lkhd}^\Omega(\alpha_0 \cup \{x \mapsto \alpha(x)\} \\
&+ \sum_{x' \in \mathcal{U}(\alpha_0) \setminus \{x\}} \frac{w_{\alpha_0}(x', \alpha(x'))}{\sum_{(x'',b'') \in \mathcal{S}_{\alpha_0}} w_{\alpha_0}(x'', b'')} \mathrm{lkhd}^\Omega(\alpha_0 \cup \{x' \mapsto \alpha(x')\}, \alpha) \\
&\leq \frac{w_{\alpha_0}(x, \alpha(x))}{\sum_{(x'',b'') \in \mathcal{S}_{\alpha_0}} w_{\alpha_0}(x'', b'')} \mathrm{lkhd}^\Omega(\alpha_0 \cup \{x \mapsto \alpha(x)\}) \\
&+ \sum_{x' \in \mathcal{U}(\alpha_0) \setminus \{x\}} \left(\frac{w_{\alpha_0}(x', \alpha(x'))}{\sum_{(x'',b'') \in \mathcal{S}_{\alpha_0}} w_{\alpha_0}(x'', b'')} \right. \\
&\qquad \left. \cdot \mathrm{lkhd}^\Omega(\alpha_0 \cup \{x' \mapsto \alpha(x')\} \cup \{x \mapsto \alpha(x)\}, \alpha) \right),
\end{aligned}
$$

by induction hypothesis. Now the problem is that we cannot really continue this computation, because we do not know how $w_{\alpha_0}(x', \alpha(x'))$ compares to $w_{\alpha_0 \cup \{x \mapsto \alpha(x)\}}(x', \alpha(x'))$. We do not run into that sort of problems in the usual case, because all variables have the same probability of being next in any subset of permutations compatible with α_0, and so we can relate the likelihood before the new assignment and after the new assignment.

These problems make the choice of $\Omega(n, w, L)$ an unlikely candidate to be able to prove that PPSZ can work with a smaller permutation set within this proof framework. A proof that does not rely on induction might be able to use it, but such a proof seems out of our grasp so far.

6.2 A block-wise construction for permutations

The previous attempt left us with the idea that, for PPSZ to work on a restricted set of permutations, this set of permutations needed to have both some measure of w-wise independence and some measure of uniformity. In this section, we will explain a simple construction for permutations sets that may give us both these properties. As we will see, an analogous proof to the proof of Chapter 5 fails to go through because we are not able to control the bad correlations that may happen between two variables.

6.2.1 Construction of the permutation set

The crucial observation for this permutation set is that we do not necessarily need a full w-wise independence, but that only the sets of variables appearing in the critical clause trees for a given assignment are actually needed. An approximate construction should consequently be "good enough" if it works for a large number of critical clause trees.
The basic construction works as follows:

1. Create a set of n w-wise independent random variables, taking their values in $[1, ..., \sqrt{n}]$.
2. Assign each variable of the formula to one of the generated random variables: this corresponds to the *block* a variable is assigned to.
3. In each block, sort the variables in an arbitrary order (for instance the lexicographic order of the variable names).
4. For each of these *base permutations*, generate a set of $(\sqrt{n})!$ permutations by considering all the possible orders of blocks.
5. Execute PPSZ on each with each of these sets of permutations.

This generates a permutation set of size $(\sqrt{n})!$ for every set of the initial random variables. We now need to address the creation of these random variables, and the relation of our sets of permutations to the critical trees.

A set of n w-wise independent random variables

The lemmas and proofs of this section are based on an exercise given in the SAT class at ETHZ in 2012 [18]. We will prove the following lemma, which we will then use with $t = \sqrt{n}$ to get the construction discussed in the beginning of this section:

Lemma 6.2 *Let w be a constant and $t > 0$. There exists a code $\mathcal{C} \subseteq \{0,1\}^r$ of length $r = 2^{2w} w^2 \log^2 n$ of size $|\mathcal{C}| = t$ with the following property. Let $Y_1, Y_2, ..., Y_r$ be a supply of mutually independent random variables distributed uniformly among the numbers $\{0, ..., t-1\}$. Let $\{c_1, c_2, ..., c_t\} = \mathcal{C}$ be the codewords.*

Consider the n random variables $X_1, ..., X_n$ defined as

$$X_i = \sum_{j=1}^{r} (c_i)_j \cdot Y_j \mod t$$

Then for any w-tuple $i_1, i_2, ..., i_w \in \{1...t\}$ of pairwise distinct indices and for all $i \in \{i_1, ..., i_w\}, j \in \{1...w\}$, the probability that all values $X_{i_1}, ..., X_{i_w}$ are distinct and that X_i is the j-th value is at least $1/w - w/t$. Moreover, this code can be generated in subexponential time in n.

To prove this lemma, we will need an auxilliary lemma:

Lemma 6.3 *Let w be some constant and $t > 0$. There exists a code $\mathcal{C}' \subseteq \{0,1\}^r$ of length $r = 2^w w \log n$ of size $|\mathcal{C}'| = t^{1/2}$ with the property that for any w-tuple $c_1, ..., c_w$ of codewords, there exists indices $i_1, i_2, ..., i_w$ such that c_j is (among the w-tuple) the unique codeword where the i_j-th bit is one, for all $1 \leq j \leq w$. This code can be generated in subexponential time.*

Proof The existence argument is probabilistic. Suppose we draw $c_1, c_2, ...c_{|\mathcal{C}'|}$ independently and uniformly at random from $\{0,1\}^r$, Consider w fixed pairwise distinct indices $i_1, i_2, ..., i_w$. The probability that there is no index j such that $(c_{i_1})_j = 1$ whereas $(c_{i_k})_j = 0$ for all $2 \leq l \leq w$ is $(1 - 2^{-w})^r$. Since there are $|\mathcal{C}'|^w$ possibilities to pick w elements among $|\mathcal{C}'|$, the probability of a "bad" event to happen for our choice of $c_1, ..., c_{|\mathcal{C}'|}$ is, by the union bound, at most $|\mathcal{C}'|(1 - 2^{-w})^r$. If we now plug in the values from the lemma, this yields a probability of $t^{1/2w}(1 - 2^{-w})^{2^w w \log n} \leq t^{1/2w} e^{-w \log n} = o(1)$ if $t = o(n)$. Hence, the desired code exists. The subexponential time generation can be achieved by considering all possible sets of size t among all the possible codewords of size r. □

We will now use Lemma 6.3 to prove Lemma 6.2.

Proof (Lemma 6.2) We consider the code $\mathcal{C}'$ generated in Lemma 6.3 and enumerate them in a matrix of size $t^{1/2} \times 2^w w \log n$. We then replace every 1 of this matrix by a copy of $\mathcal{C}'$ in matrix form, and every 0 by a zero matrix of size $t^{1/2} \times 2^w w \log n$. We use this matrix as a representation for $\mathcal{C}$; it is of size $t \times 2^{2w} w^2 \log^2 n$ as desired, and this code can be generated in subexponential time since $\mathcal{C}'$ can be. We now need to show that this code verifies the desired property.

The matrix definition guarantees that, for all sets of w codewords, each codeword contains a bit that is set to 1 and that is set to 0 in all other codewords. Consequently, for each codeword c_i, there exists Y_j that only appears in the sum defining this codeword; hence, the values of the variables X_i are independent and distributed uniformly among $\{0, ..., t-1\}$.

Each pair fixed of variables X_i, X_j attains equal value with probability $1/t$. By a union bound, the values are all pairwise distinct with probability at least $1 - w^2/t$ Conditioning on this, every variable has the same probability

$1/w$ to be at any place in the set of values. Hence, for any w-tuple $i_1, ..., i_w \in \{1...\sqrt{n}\}$ of pairwise distinct indices and for all $i \in \{i_1, ..., i_w\}, j \in \{1, ..., w\}$, the probability that all values $X_{i_1}, ..., X_{i_w}$ are distinct (event E_1) and that X_i is the j-th value (event E_2) is

$$\Pr[E_1 \wedge E_2] = \Pr[E_2 \mid E_1] \cdot \Pr[E_1] \geq \frac{1}{w} \cdot \left(1 - \frac{w^2}{t}\right) = \frac{1}{w} - \frac{w}{t},$$

as desired. □

We also observe that, even if w is actually a slowly growing function of n (such as $\log \log n$), the code $\mathcal{C}$ can still be generated in subexponential time of t, so we can fully derandomize this construction by enumeration in subexponential time. If we now set $t = \sqrt{n}$, we can deterministically construct n w-wise independent variables over the domain $\{1, ..., \sqrt{n}\}$ in subexponential time.

A block-wise construction for the set of permutations

Using the construction from Lemma 6.2, we can now set $w = D$ (where D is a parameter of the PPSZ algorithm) and efficiently build sets of n D-wise independent variables that take their values between 1 and $\sqrt{n}$. We associate each of these variables with one variable in our k-CNF formula. We now build a "base" permutation for each of these sets as follows. We consider $\sqrt{n}$ blocks of variables. We put each formula variable in the block corresponding to its variable (for instance variable x_i associated to variable X_i that has value j is put in block j). Then, from these "base" permutations, we generate all $(\sqrt{n})!$ permutations corresponding to all the permutations of the $\sqrt{n}$ blocks of variables. We will then consider every set of permutations arise from a base permutation as a set of permutations to run PPSZ on. In the end we get $\sqrt{n}^{2^{2D} D^2 \log^2 n} (\sqrt{n})!$ permutations in total, which stays subexponential for slowly growing functions D of n.

6.2.2 Invalidating critical clause trees

For each base permutation, and for all satisfying assignments, we say that a critical clause tree is *invalid* if it has two nodes whose var-label end up in the same block. If this happens, we say that these two variables are *in conflict with each other*. For instance, a critical clause tree where the root critical clause has two variables in the same block would be invalidated. Ideally, we would like to say that the expected number of invalid trees is small, and hence that there exists a base permutation that has a low number of invalid trees; enumerating all the base permutations would then allow to guarantee that such a permutation is encountered. Let us see how this would work.

First, we use a crude bound on the number of possible critical clause trees that may ever arise from a given formula. The critical clause trees that we

are interested in all have less than D nodes, where D is a slowly growing function of n, say $\log\log n$ in what follows. The possible number of clauses for a k-SAT formula is $\mathcal{O}(n^k)$. We also need to consider all the possible subclauses, because they may also arise in later critical trees (some variables may become frozen and hence critical during the algorithm); a given clause can give rise to 2^k subclauses at most, so we get $\mathcal{O}(2^k n^k) = \mathcal{O}(n^k)$ clauses and subclauses in total. When building a tree, we first choose the branch to extend (and for this we have at most D possibilities, since we have at most D nodes), to which we assign exactly one critical clause. Hence, the number of possible critical clause trees is $\mathcal{O}((Dn^k)^D)$.
A conflict arises when more than two variables within these (at most) D variables end up in the same bucket. Since the attribution of the buckets (as provided by Lemma 6.2) is done such that it is D-wise independent, the probability that two fixed variables are in conflict is equal to $1/\sqrt{n}$. So the expected number of conflicts for a given tree is $\frac{\binom{D}{2}}{\sqrt{n}} = \mathcal{O}\left(\frac{D^2}{\sqrt{n}}\right)$; the total expected number of conflicts for all trees for a variable (that we bound by the total number of trees) is then $\mathcal{O}\left(\frac{(n^k)^D D^{D+2}}{\sqrt{n}}\right)$. This estimation is, unfortunately, too large to be able to conclude that a small enough number of trees is invalidated; even if our bound on the number of trees may be improved by a more careful counting, we would not be able to bound it tightly enough to be even $\mathcal{O}(\sqrt{n})$. It is not enough to show that almost all trees are valid, because it may happen that the likelihood of an assignment and the invalidation of the corresponding critical trees correlate in an uncontrollable or bad way.
To circumvent this problem, we invoke the sparsification lemma [7], which we state here:

Lemma 6.4 (Sparsification lemma [7]) *For all $\varepsilon > 0$ and positive k, there is a constant C so that any k-SAT formula Φ with n variables, can be expressed as $\Phi = \bigvee_{i=1}^{t} \Psi_i$, where $t \leq 2^{\varepsilon n}$ and each Ψ_i is a k-SAT formula of maximum degree C, where the degree of a variable is the number of clauses in which it appears. Moreover, this disjunction can be computed by an algorithm running in time poly$(n)2^{\varepsilon n}$.*
In what follows, we will consider that our formula is one of the formulas Ψ_i (and consequently that its variable have degree at most C, a constant). Let us now bound the number of critical clause trees that are associated to a given critical variable. The root node can have at most C labellings, since the critical variable can only be in C different clauses. When building the tree, we can choose at most D branches to extend. For the var-label of the new node, we have at most $k-1$ choices, and once the var-label is chosen, then we have only C choices. Hence, the number of critical clause trees for a single variables is bounded by $(kDC)^D$, and so the expected number of conflicts for all trees (all variables considered) is bounded by $\mathcal{O}\left(\frac{(kDC)^D D^2 n}{\sqrt{n}}\right)$, which, for D small enough (say $\log\log n$), is $o(n)$. Consequently, only a

negligible number of variables can possibly be impacted by the invalidating of some critical clause trees.

6.2.3 Probability of returning a given assignment

Ideally, with our permutation construction, we would be able to prove the same bounds (up to an ε) for the probability of returning a given assignment than in Chapter 5. In this section, we try to apply the method of the proof given in that chapter to our current problem, and we explain where this approach fails to yield any interesting bound.

In what follows, our set of permutations is one of the sets of $(\sqrt{n})!$ permutations that arise from a single "base permutation" as defined in the previous section. We call this set of permutations $\Omega(\mathcal{B}(\alpha_0))$ where $\mathcal{B}(\alpha_0)$ is the set of blocks of $V \backslash \mathrm{dom}(\alpha_0)$. We define a *block assignment* $\{B \mapsto C\}$ as the mapping between the ordered set of variables B and the ordered set of values C.

We start by defining block-likelihood as follows:

Definition 6.5 *Let $F^{[\alpha_0]}$ be satisfiable and let $\mathcal{S}_{\alpha_0}$ be the set of block assignments $\{B \mapsto C\}$ such that B is in $\mathcal{B}(\alpha_0)$ and $F^{[\alpha_0 \cup \{B \mapsto C\}]}$ is satisfiable. We define the random process $\mathrm{AssignSL}^B(F, \alpha_0)$ that produces an assignment of $vbl(F^{[\alpha_0]})$ as follows. Start with the assignment α_0, and repeat the following step until $vbl(F^{[\alpha_0]}) = \emptyset$: Choose an block assignment $\{B \mapsto C\}$ uniformly at random in $\mathcal{S}_{\alpha_0}$ and add it to α_0. At the end, output α_0. Let α be a total assignment on $vbl(F)$. Then the block-likelihood of completing α_0 to α, in writing $blkhd(\alpha_0, \alpha)$, is defined as the probability that $\mathrm{AssignSL}^B(F, \alpha_0)$ returns α. For completeness, if $F^{[\alpha_0]}$ is not satisfiable, or if α_0 is chosen such that it is not compatible with the current block decomposition, we define $blkhd(\alpha_0, \alpha) = 0$.*

We also need to slightly adapt the definition of the cost function. For this, we define three sets of variables as follows.

- The block-non-frozen variables are the variables that are not frozen when starting to process a block. We define this set as $\mathrm{V}_{\mathrm{bnf}}(\alpha_0)$, and $\mathrm{V}_{\mathrm{bnf}}(\alpha_0, B)$ as the subset of variables that are in the block B. For $x \in \mathrm{V}_{\mathrm{bnf}}(\alpha_0)$, we define $\mathrm{cost}(\alpha_0, \alpha, x) = S_k^{(D)}$.

- The block-forced variables are the variables that are currently forced. We define this set as $\mathrm{V}_{\mathrm{bfo}}(\alpha_0)$, and $\mathrm{V}_{\mathrm{bfo}}(\alpha_0, B)$ as the subset of variables that are in the block B. For $x \in \mathrm{V}_{\mathrm{bfo}}(\alpha_0)$, we define $\mathrm{cost}(\alpha_0, \alpha, x) = 0$.

- The block-frozen variables are the variables that are frozen, but that are not forced when starting to process a block. We define this set as $\mathrm{V}_{\mathrm{bfr}}(\alpha_0)$, and $\mathrm{V}_{\mathrm{bfr}}(\alpha_0, B)$ as the subset of variables that are in the block B. For $x \in \mathrm{V}_{\mathrm{bfr}}(\alpha_0)$, we define $\mathrm{cost}(\alpha_0, \alpha, x) = \Pr_{\pi \in \Omega(\mathcal{B}(\alpha_0))}[x \in \mathrm{V}_{\mathrm{bfr}}(\alpha_0, B(x))]$, where $B(x)$ is the block containing x.

These sets can also be seen as the sets of non-frozen, forced and frozen variables when only looking at them between the processing of the different blocks.

We would like to prove that $p(\alpha_0, \alpha) = \text{blkhd}(\alpha_0, \alpha) \cdot 2^{-cost(\alpha_0, \alpha)}$, where $p(\alpha_0, \alpha)$ is the probability that PPSZ returns α when starting with α, assigning the values blocks by blocks and using the aforementioned set of permutations. To do this, we proceed by induction on the size of α_0: if α_0 is complete, then the statement obviously holds. For a block B, let $\mathcal{B}$ be the set of block assignments $\{\{B, C\}\}$ such that B is picked u.a.r. in $\mathcal{B}(\alpha_0)$, and C is picked u.a.r. in $\mathcal{P}(B)$, the set of possible assignments for B (i.e. the assignment that are not explicitely forbidden by a block-forced variable value).

We have:

$$\begin{aligned}
p(\alpha_0, \alpha) &= \mathop{\mathbb{E}}_{B \in_{\text{u.a.r}} \mathcal{B}(\alpha_0); C} [p(\alpha \cup \{B \mapsto C\}, \alpha)] \\
&= \sum_{B \in \mathcal{B}(\alpha_0)} \frac{1}{|\mathcal{B}(\alpha_0)|} \sum_{C \in \mathcal{P}(B)} \frac{1}{|\mathcal{P}(B)|} p(\alpha_0 \cup \{B \mapsto C\}, \alpha) \\
&= \sum_{B \in \mathcal{B}(\alpha_0)} \frac{1}{|\mathcal{B}(\alpha_0)|} \cdot \frac{1}{|\mathcal{P}(B)|} p(\alpha_0 \cup \{B \mapsto \alpha(B)\}, \alpha) \\
&= \mathop{\mathbb{E}}_{B \in \text{u.a.r.} \mathcal{B}(\alpha_0)} \left[\frac{1}{|\mathcal{P}(B)|} p(\alpha_0 \cup \{B \mapsto \alpha(B)\}, \alpha) \right].
\end{aligned}$$

We now define a distribution ξ over the blocks of $\mathcal{B}(\alpha_0)$ weighted according to $\text{blkhd}(\alpha_0 \cup \{B \mapsto \alpha(B)\}, \alpha)$. We have that

$$\begin{aligned}
\text{blkhd}(\alpha_0, \alpha) &= \mathop{\mathbb{E}}_{(B,C) \in_{\text{u.a.r.}} \mathcal{S}_{\alpha_0}} [\text{blkhd}(\alpha_0 \cup \{B \mapsto C\}, \alpha)] \\
&= \sum_{B \in \mathcal{B}(\alpha_0)} \frac{1}{|\mathcal{S}_{\alpha_0}|} \text{blkhd}(\alpha_0 \cup \{B \mapsto \alpha(B)\}, \alpha)
\end{aligned}$$

which allows us to determine the sum of all the weights, and so the probability of any block B is given by

$$p_\xi(B) = \frac{\text{blkhd}(\alpha_0 \cup \{B \mapsto \alpha(B)\}, \alpha)}{|\mathcal{S}_{\alpha_0}| \cdot \text{blkhd}(\alpha_0, \alpha)}.$$

We now want to introduce this probability distribution in our current expres-

sion for $p(\alpha_0, \alpha)$:

$$\begin{aligned}
p(\alpha_0,\alpha) &= \mathop{\mathbb{E}}_{B\in\text{u.a.r.}\mathcal{B}(\alpha_0)}\left[\frac{1}{|\mathcal{P}(B)|}p(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)\right]\\
&= \sum_{B\in\mathcal{B}(\alpha_0)}\frac{1}{|\mathcal{B}(\alpha_0)|\cdot|\mathcal{P}(B)|}p(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)\\
&= \sum_{B\in\mathcal{B}(\alpha_0)}\frac{|\mathcal{S}_{\alpha_0}|\cdot\text{blkhd}(\alpha_0,\alpha)}{|\mathcal{B}(\alpha_0)|\cdot|\mathcal{P}(B)|}\cdot\frac{\text{blkhd}(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)}{|\mathcal{S}_{\alpha_0}|\cdot\text{blkhd}(\alpha_0,\alpha)}\\
&\cdot\frac{p(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)}{\text{blkhd}(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)}\\
&= \frac{|\mathcal{S}_{\alpha_0}|\cdot\text{blkhd}(\alpha_0,\alpha)}{|\mathcal{B}(\alpha_0)|}\cdot\mathop{\mathbb{E}}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}\left[\frac{p(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)}{|\mathcal{P}(B)|\text{blkhd}(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)}\right].
\end{aligned}$$

We apply the induction hypothesis and we get:

$$p(\alpha_0,\alpha)\geq\frac{|\mathcal{S}_{\alpha_0}|\cdot\text{blkhd}(\alpha_0,\alpha)}{|\mathcal{B}(\alpha_0)|}\mathop{\mathbb{E}}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}\left[\frac{2^{-\text{cost}(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)}}{|\mathcal{P}(B)|}\right].$$

Now we use Jensen's inequality (see Appendix A.1 on page 63) with the convex function $x\mapsto 2^{-x}$ and we get:

$$p(\alpha_0,\alpha)\geq\text{blkhd}(\alpha_0,\alpha)2^{\log\frac{|\mathcal{S}_{\alpha_0}|}{|\mathcal{B}(\alpha_0)|}-\mathbb{E}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}[\log|\mathcal{P}(B)|]+\mathbb{E}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}[-\text{cost}(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)]}.$$

The cost of the block-non-frozen variables of the block is $S_k^{(D)}$ now and will be 0 when the block is assigned; the probability that the block-frozen variables are not forced at the beginning of the block is 1, and this probability is 0 once the block is assigned. It could happen that the cost of other variables increases: if some block-non-frozen variable (that currently has cost $S_k^{(D)}$) becomes frozen and if the set of permutation happens to be bad for that specific variable (because of the invalidating of critical clause trees), then it could happen that the cost for that specific variable increases from $S_k^{(D)}$ to up to 1. This can only happen for at most $\mathcal{O}\left(\frac{(kDC)^D dD^n}{\sqrt{n}}\right)$ variables; we will call this potential cost increase $\delta_{\Omega,D,C}$. So we get

$$\begin{aligned}
&\mathop{\mathbb{E}}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}[\text{cost}(\alpha_0\cup\{B\mapsto\alpha(B)\},\alpha)]\\
&\leq\text{cost}(\alpha_0,\alpha)-\mathop{\mathbb{E}}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}\left[S_k^{(D)}\cdot|\text{V}_{\text{bnf}}(\alpha_0,B)|+|\text{V}_{\text{bfr}}(\alpha_0,B)|\right]+\delta_{\Omega,D,C}.
\end{aligned}$$

Moreover, $|\mathcal{P}(B)|=2^{|B|-\text{V}_{\text{bfo}}(\alpha_0,B)}$; hence we get

$$\begin{aligned}
&p(\alpha_0,\alpha)\geq\text{blkhd}(\alpha_0,\alpha)\\
&\cdot 2^{\log\frac{|\mathcal{S}_{\alpha_0}|}{|\mathcal{B}(\alpha_0)|}+\mathbb{E}_{B\in_{\tilde{\xi}}\mathcal{B}(\alpha_0)}[|\text{V}_{\text{bfo}}(\alpha_0,B)|-|B|+S_k^{(D)}|\text{V}_{\text{bnf}}(\alpha_0,B)|+|\text{V}_{\text{bfr}}(\alpha_0,B)|]-\text{cost}(\alpha_0,\alpha)-\delta_{\Omega,D,C}}.
\end{aligned}$$

For our final statement to hold, we would need that

$$\log \frac{|\mathcal{S}_{\alpha_0}|}{|\mathcal{B}(\alpha_0)|} + \mathop{\mathbb{E}}_{B \in_{\tilde{\xi}} \mathcal{B}(\alpha_0)} [S_k^{(D)} |\mathrm{V}_{\mathrm{bnf}}(\alpha_0, B)|] - \delta_{\Omega,D,C} \geq \mathop{\mathbb{E}}_{B \in_{\tilde{\xi}} \mathcal{B}(\alpha_0)} [|\mathrm{V}_{\mathrm{bnf}}(\alpha_0, B)|].$$

Unfortunately, this is not necessarily the case, even without accounting for the $\delta_{\Omega,D,C}$ term. Suppose that the only satisfying assignments at this point of the algorithm are the all-0 assignment and the all-1 assignment (restricted to the variables still to be assigned). Then all the variables are block-non-frozen; on the other hand, $|\mathcal{S}_{\alpha_0}|$ is exactly twice $|\mathcal{B}(\alpha_0)|$, and so $\log \frac{|\mathcal{S}_{\alpha_0}|}{|\mathcal{B}(\alpha_0)|} = 1$, which is too small even for constant values of $|\mathrm{V}_{\mathrm{bnf}}(\alpha_0, B)|$ (depending on k).
The problem of this approach is that it may happen that, even within small blocks, some uncontrollable correlations may happen between the variables. Using uniform random permutations allows us to control both the correlations between cost and likelihood and inter-variable cost correlations in a fairly natural way; managing to do so with any other set of permutations has so far proven to be challenging to say the least. It stays open whether an analysis of PPSZ with a smaller set of permutations can go through for the multiple satisfying assignments case. Approaching the problem by considering the derandomization of the "guessing" bits before the derandomization of the "permutation" bits might help circumventing this obstacle with regards to the global derandomization.

Chapter 7

PPSZ with oracle

In this chapter, we investigate the question of whether we can replace the random bits that correspond to non-frozen variables by fixed assignments. We will consider two types of oracle: a "nice" oracle will return 0 always when we ask it for the value of a non-frozen variable; a "poor" oracle will return an arbitrary bit (with the underlying assumption that these bits can be computed deterministically instead of flipped randomly). Poor oracles may be easier to construct, because we do not necessarily need to know the position of the non-frozen variables to be able to build them – for instance, one could imagine oracles made of a small set of codewords such that some codeword is guaranteed to yield a satisfying assignment for the formula, without defining precisely if a given bit is used for a non-frozen variable or for a frozen, not forced variable. On the other hand, nice oracles make proofs somewhat more accessible (because we know exactly what a non-frozen variable will be set to); moreover, if we can prove that something will not work for a nice oracle, then it is a strong indication that it will not work for a poor oracle either.
In this chapter, we mainly concern ourselves with nice oracles. We first prove that the success probability of PPZ with a nice oracle is at least the same as the success probability without an oracle. We also give a partial proof that a nice oracle does not hurt PPSZ either; this proof, however, relies on a strong conjecture that we have not been able to prove so far. In each of the sections, we conclude by giving some thoughts about the translation of the proof in the poor oracle situation.

7.1 PPZ with oracle

7.1.1 PPZ with a nice oracle

In this section, we prove the following theorem:

Theorem 7.1 *Let PPZ^O be a version of PPZ that, when encountering a non-frozen variable, sets it to 0. Then, when running $PPZ^O(F, V)$ on a k-CNF formula F over V, the probability over all permutations of V that a given variable is frozen, but is not forced when processing it, is at most $1 - 1/k$.*
This theorem implies the following corollary:

Corollary 7.2 *The probability that $PPZ^O(F, V)$ finds a satisfying assignment in a satisfiable k-CNF formula F over a set of n variables is at least $2^{-n-n/k}$, where PPZ^O is a version of PPZ that, when encountering a non-frozen variable, sets it to 0.*

Proof (of Corollary 7.2) When considering a fixed permutation, $\mathrm{PPZ^O}$ succeeds if and only if it agrees on the frozen value when encountering a frozen, non-forced variable. We can write:

$$\Pr_{\pi}[\mathrm{PPZ^O succeeds}] = \mathop{\mathbb{E}}_{\pi}[2^{-\mathrm{fr}(F,\pi)}],$$

where $\mathrm{fr}(F, \pi)$ is the number of frozen, non-forced variables encountered when processing F with π. Using Jensen's inequality (see Appendix A.1 on page 63), we can write

$$\Pr_{\pi}[\mathrm{PPZ^O succeeds}] \geq 2^{-\mathbb{E}_{\pi}[\mathrm{fr}(F,\pi)]}.$$

By Theorem 7.1 and linearity of expectation, the corollary follows. □
The proof of the theorem relies on the notions of *critical variable* and *critical clauses*; we repeat the definition of these concepts here.

Definition 2.2 (Critical variables and critical clauses [17]) *Given an assignment $\alpha \in sat_V(F)$, we call a variable $x \in V$* critical for *α if flipping the value of x in α stops it being satisfying. This requires at least one clause in F that has x or $\bar{x}$ as a unique literal that is set to 1 by α. Such a clause is called a* critical clause for *x.*

Proof (of Theorem 7.1) In this proof, we will first consider (≤ 3)-CNF formulas; we will then explain how the proof can be extended to $(\leq k)$-CNF formulas. We first make the following observation: since we are setting non-frozen variables to 0 whenever we encounter one, the random permutation used by a given run of the algorithm fully determines the satisfying assignment that can be returned: a run with a given permutation will either return the satisfying assignment it is associated to, or fail to find a satisfying assignment. We start by partitioning the permutations according to their associated satisfying assignment, and we will show that, when considering any set of permutations that is associated to a fixed satisfying assignment, any variable is either forced or set by the oracle with probability at least $1/3$. Since this holds for all sets of permutations, it is also true for the whole set of permutations, and hence the theorem holds.

Given a satisfying assignment α, we define Π_α as the set of permutations associated to it. Let $\pi \in \Pi_\alpha$ be some permutation. When processing the variables of F in the order defined by π, we will encounter two types of variables: the non-frozen variables (which can, at this point in the permutation, be set either way so that the resulting partial assignment is still compatible with a satisfying assignment), and the frozen variables, which can only get assigned to one value. If a variable x gets frozen, then it is critical, and it has a critical clause. Let $\{x, y, z\}$ be such a clause for the variable x. If x is the last of the variables $\{x, y, z\}$ in the permutation π, then it is automatically set to its value, since the clause is by then reduced to a unit clause. We re-partition Π_α into Π_α^{xyz}, Π_α^{xzy}, Π_α^{yxz}, Π_α^{yzx}, Π_α^{zxy}, and Π_α^{zyx}, where

$$\Pi_\alpha^{xyz} = \{\pi \in \Pi_\alpha \mid x, y, z \text{ appear in this order}\},$$

and the other sets are defined similarly.
We first consider a permutation $\pi \in \Pi_\alpha^{xyz}$. If x is non-frozen for this permutation, then it is set to 0 by the oracle, so we assume that x is frozen when we process it. Now we describe the permutation π by its non-frozen variables between x and z: it looks like $x\{NF1\}y\{NF2\}z$, where $\{NF1\}$ is the (ordered) set of non-frozen variables between x and y, and $\{NF2\}$ is the set of non-frozen variables between y and z. Some frozen variables may be interleaved in the permutation as well; we ignore them for our purpose. We now consider the "rotated" permutation $\{NF1\}y\{NF2\}zx$, where we move the variables from $\{NF1\} \cup \{y\} \cup \{NF2\} \cup \{z\}$ "up" from one place within the non-frozen variables: the first variable of $\{NF1\}$ is moved to the place of x, the second variable of $\{NF1\}$ is moved to the place of the first variable of $\{NF1\}$, y is put in the place of the last variable of $\{NF1\}$, z is put in the place of the last variable of $\{NF2\}$, and finally x is put in the place of z. For instance, the permutation

u_1	x	u_2	u_3	u_4	y	u_5	u_6	z

where u_1, u_4, and u_6 are non-frozen would become the permutation

u_1	u_4	u_2	u_3	y	u_6	u_5	z	x

where the elements in white stays in place whereas the elements in grey are rotated.
With this operation, the order of the non-frozen variables in the permutation has not changed. Hence, the permutation is in Π_α, and so it is in Π_α^{yzx}. If x is frozen, then this rotation transformation is injective from Π_α^{xyz} to Π_α^{yzx}. We can do a similar operation for permutations from Π_α^{xzy}, and we end up with an injective transformation from Π_α^{xzy} to Π_α^{zyx}.
We now consider a permutation $\pi \in \Pi_\alpha^{yxz}$ where x is not guessed. With the same notation as above, π is of the type $y\{NF3\}x\{NF4\}z$; we apply a similar rotation (but this time only on $x\{NF4\}z$) to get a permutation of

type $y\{NF3\}\{NF4\}zx$. This rotation transformation is injective from Π_α^{yxz} to Π_α^{yzx}, and we can define a similar injective transformation from Π_α^{zxy} to Π_α^{zyx}.
It may happen that a permutation from Π_α^{xyz} and a permutation from Π_α^{yxz} have the same image in Π_α^{yzx}; this is not an issue, because a permutation in Π_α^{yzx} still has at most two preimages. Hence, if x is frozen, at least $1/3$ of the permutations in Π_α have x as a last variable (and thus it is forced).
This argument trivially generalizes to $(\leq k)$-CNF formulas, where we would partition Π_α into $r!$ sets for a critical clause of size r ($r \leq k$), and where a permutation such that x is last has at most $r-1$ preimages in the other sets via similar rotations. □

7.1.2 PPZ with a poor oracle

The proof of the previous section relies on a number of assumptions. First, it seems only possible to build such an oracle if we know that a variable is non-frozen when we encounter it. Given this, the fact that we set it to 0 or to some arbitrary value is also an issue: when rotating the variables, if the arbitrary values are not all equal, then the assignment that is returned will not be the same. It may be possible to solve that problem by making some assumptions on the "poor" oracle, but we have not investigated in this direction so far.
Finally, also observe that this proof does not go through when considering smaller sets of permutations (such as the ones defined in Chapter 6), because we cannot guarantee that the rotated permutations are also part of the set of permutations that we use.

7.2 PPSZ with oracle

Ideally, we would like to prove the following conjecture:

Conjecture 7.3 *For any $\varepsilon > 0$, there exists D such that the probability that PPSZ^O(F, V, ∅, D) finds a satisfying assignment in a satisfiable $(\leq k)$-CNF formula F over a set V of n variables is at least $2^{-(S_k^{(D)}n+\varepsilon)n}$, where*

$$\lim_{D\to\infty} S_k^{(D)} = S_k = \int_0^1 \frac{t^{1/(k-1)-t}}{1-t}\,dt.$$

and where PPSZ^O is a version of PPSZ that, when encountering a non-frozen variable, sets it to 0.

7.2.1 Problems with the analysis of standard PPSZ

At the core of the analysis of standard PPSZ is the compensation of the effects of the likelihood and of the cost of individual assignments when

computing the probability that a satisfying assignment is returned, which is mainly expressed in the following lemma (proven in [17]):

Lemma 2.14 ([17]) *Let α_0 and α be fixed and compatible. For any fixed variable $x \in \mathcal{U}(\alpha_0)$, if we set x according to α, then*

(i) the likelihood of α 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 $F^{[\alpha_0]}$.

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

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

When choosing $x \in \mathcal{U}(\alpha_0)$ uniformly at random and setting it according to α, then

(iii) the likelihood of α increases on average as

$$\mathbb{E}[lkhd(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)] = \left(1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}\right) lkhd(\alpha_0, \alpha)$$

(iv) the cost of a fixed variable $y \in V_{fr}(\alpha_0)$ decreases on expectation as

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

where

$$s = \begin{cases} 1 & \text{if } y \in V_{fr}(\alpha_0) \\ S_k^{(D)} & \text{if } y \in V_{nf}(\alpha_0) \\ 0 & \text{if } y \in V_{fo}(\alpha_0) \end{cases}$$

This lemma is true as is in the PPSZ$^{\mathrm{O}}$ setting as well – nothing in it makes any reference to the fact that the value of non-frozen variables is chosen at random. However, it is not useable as is in the rest of the proof: some assignments will never be returned by PPSZ$^{\mathrm{O}}$, and some may not be returned for some permutations (a trivial example is the one where the all-1 assignment and another assignment are satisfying the formula; the all-1 assignment will never be returned by PPSZ$^{\mathrm{O}}$). Hence, setting x according to a fixed assignment α may not make sense in this context. Another breaking point appears when trying to relate the distribution of PPSZ$^{\mathrm{O}}$ (picking a variable x, and then a bit b) to the uniform distribution over $\mathcal{S}_{\alpha_0}$ (the set of literals ℓ such that $F^{[\alpha_0 \cup \{\ell\}]}$ is satisfiable) because the probability that a given (x, b) is chosen is, in PPSZ$^{\mathrm{O}}$ and for a non-frozen variable x, $\frac{1}{n(\alpha_0)}$ if $\alpha(x) = 0$, but 0 otherwise.

We then tried to define another likelihood. Instead of picking elements uniformly at random from $\mathcal{S}_{\alpha_0}$, we define the set $\mathcal{F}_{\alpha_0}$ as the set of the following elements:

- for all $x \in \mathrm{V}_{\mathrm{fo}}(\alpha_0) \cup \mathrm{V}_{\mathrm{fr}}(\alpha_0)$, $\mathcal{F}_{\alpha_0}$ contains the assignment (x, b) such that $F^{[\alpha_0 \cup \{x \mapsto b\}]}$ is satisfiable,
- for all $x \in \mathrm{V}_{\mathrm{nf}}(\alpha_0)$, $\mathcal{F}_{\alpha_0}$ contains the assignment $(x, 0)$.

and we define the random process $\mathrm{AssignF}(F, \alpha_0)$ that produces an assignment on V as follows. Start with the assignment α_0, and repeat the following step until $\mathrm{vbl}(F^{[\alpha_0]}) = \emptyset$: Choose a value assignment $\ell \in \mathcal{F}_{\alpha_0}$ uniformly at random and add ℓ to α_0. At the end, output α_0. We then define $\mathrm{lkhd}^{\mathrm{O}}(\alpha_0, \alpha)$ as the probability that $\mathrm{AssignF}(F, \alpha_0)$ returns α. But we did not manage to come up with an analogue of Lemma 2.14 for that case: in particular, setting a variable x according to a fixed assignment α may not make sense with regard to $\mathcal{F}_{\alpha_0}$ at some point (because x is non-frozen at that point and $\alpha(x) = 1$), but it may make sense at a later point (if x becomes non-frozen in a way that is still compatible with α).

7.2.2 Discarding the likelihood

The difficulties that we encountered with coming up with a proper definition for likelihood in the $\mathrm{PPSZ}^{\mathrm{O}}$ case may come from the fact that, once the permutation π is given, the algorithm can only possibly return a single satisfying assignment: it will either return the assignment that corresponds to setting non-frozen variables to 0, or fail to return a satisfying assignment. Consequently, it may make sense to discard the notion of likelihood altogether. We can also define $\alpha(\pi)$ as the satisfying assignment that is returned if $\mathrm{PPSZ}^{\mathrm{O}}$ returns a satisfying assignment when invoked with the permutation π.

We can define the cost of a variable x as follow:

- if $x \in \mathrm{V}_{\mathrm{fo}}(\alpha_0)$, $\mathrm{cost}(\alpha_0, x) = 0$,
- if $x \in \mathrm{V}_{\mathrm{fr}}(\alpha_0)$, $\mathrm{cost}(\alpha_0, x) = \Pr_\pi[x \in \mathrm{Guessed}(F, \alpha_0, \alpha(\pi), \pi, D)]$,
- if $x \in \mathrm{V}_{\mathrm{nf}}(\alpha_0)$, $\mathrm{cost}(\alpha_0, x) = S_k^{(D)}$.

As previously, $\mathrm{cost}(\alpha_0) = \sum_x \mathrm{cost}(\alpha_0, x)$.

The problem in that case is that it is not clear if $\Pr[x \in \mathrm{Guessed}(F, \alpha_0, \alpha(\pi), \pi, D)]$ is bounded by $S_k^{(D)}$ as in the standard case. If it is the case, though, we can prove that Conjecture 7.3 holds as well. We state the following conjecture:

Conjecture 7.4 *Let $x \in V_{fr}(\alpha_0)$. Then*

$$\Pr_\pi[x \in \mathit{Guessed}(F, \alpha_0, \alpha(\pi), \pi, D)] \leq S_k^{(D)}.$$

In the standard case, this follows from the analysis of the unique case and from the observation that, if a variable is frozen, then the proof goes through for any fixed assignment. In the $\mathrm{PPSZ}^{\mathrm{O}}$ assignment, since a given permutation yields a unique satisfying assignment, integrating over the place of x is

not feasible anymore because we are not considering the same assignment (and hence the same critical clause tree) for all permutations. Solving this problem seems to be the main difficulty in proving Conjecture 7.4.
Observe that an analogue of Conjecture 7.4 holds for PPZ^{O} if we replace $S_k^{(D)}$ by $\frac{k-1}{k}$ (both in the conjecture and in the cost definition); consequently, what follows is another proof of Theorem 7.1.
Let us now see how the proof of Conjecture 7.3 works if we assume that Conjecture 7.4 is true. We define $p(\alpha_0)$ as the probability that PPSZ^{O} returns a satisfying assignment when starting with a partial assignment α_0, and we use an induction proof over the size of the partial assignment α_0. The base case trivially holds if α_0 is a full assignment.
Let x and b be random variables: $x \in \mathcal{U}(\alpha_0)$ u.a.r., b is either the forced value by D-implication if $x \in \text{V}_{\text{fo}}(\alpha_0)$, 0 if $x \in \text{V}_{\text{nf}}(\alpha_0)$, and u.a.r. in $\{0,1\}$ if $x \in \text{V}_{\text{fr}}(\alpha_0)$. Let $p(\alpha_0)$ be the probability that ppsz with oracle returns some satisfying assignment. We have:

$$p(\alpha_0) = \mathop{\mathbb{E}}_{x \in_{\text{u.a.r}} \mathcal{U}(\alpha_0); b}[p(\alpha_0 \cup \{x \mapsto b\})].$$

(x,b) are all elements of $\mathcal{F}_{\alpha_0} \cup \{(x,b) \in \text{V}_{\text{fr}}(\alpha_0) \times \{1\}\}$. If $(x,b) \notin \mathcal{F}_{\alpha_0}$, then $p(\alpha_0 \cup \{x \mapsto b\}) = 0$. So we have

$$p(\alpha_0) = \Pr[(x,b) \in \mathcal{F}(\alpha_0)] \cdot \mathop{\mathbb{E}}_{x;b}[p(\alpha_0 \cup \{x \mapsto b\}) \mid (x,b) \in \mathcal{F}_{\alpha_0}]$$

$(x;b)$ is in $\mathcal{F}_{\alpha_0}$ if $x \in \text{V}_{\text{nf}}(\alpha_0)$, if $x \in \text{V}_{\text{fo}}(\alpha_0)$, and if $x \in \text{V}_{\text{fr}}(\alpha_0)$ and has the right value. So

$$\begin{aligned}\Pr[(x,b) \in \mathcal{F}_{\alpha_0}] &= \Pr[x \in \text{V}_{\text{nf}}(\alpha_0)] + \Pr[x \in \text{V}_{\text{fo}}(\alpha_0)] + \frac{1}{2}\Pr[x \in |\text{V}_{\text{fr}}(\alpha_0)|] \\ &= \frac{2n(\alpha_0) - |\text{V}_{\text{fr}}(\alpha_0)|}{2n(\alpha_0)} \\ &= 1 - \frac{|\text{V}_{\text{fr}}(\alpha_0)|}{2n(\alpha_0)}.\end{aligned}$$

Our working expression becomes

$$p(\alpha_0) = \left(1 - \frac{|\text{V}_{\text{fr}}(\alpha_0)|}{2n(\alpha_0)}\right) \mathop{\mathbb{E}}_{x \in_{\text{u.a.r}} \mathcal{U}(\alpha_0); b}[p(\alpha_0 \cup \{x \mapsto b\}) \mid (x,b) \in \mathcal{F}_{\alpha_0}],$$

and then

$$p(\alpha_0) = \left(1 - \frac{|\text{V}_{\text{fr}}(\alpha_0)|}{2n(\alpha_0)}\right) \mathop{\mathbb{E}}_{(x,b) \in_{\text{u.a.r.}} \mathcal{F}_{\alpha_0}}[p(\alpha_0 \cup \{x \mapsto b\})],$$

because when we pick x u.a.r. in $\mathcal{U}(\alpha_0)$ it corresponds to a single $(x,b) \in \mathcal{F}_{\alpha_0}$.

So now we apply Jensen's inequality (see Appendix A.1 on page 63) and our induction hypothesis:

$$\begin{aligned} p(\alpha_0) &\geq 2^{\log\left(1-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{2n(\alpha_0)}\right)-\mathbb{E}[-\log(p(\alpha_0\cup\{x\mapsto b\}))]} \\ &\geq 2^{\log\left(1-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{2n(\alpha_0)}\right)-\mathbb{E}[\mathrm{cost}(\alpha_0\cup\{x\mapsto b\})]}. \end{aligned}$$

To finish the computation, we prove the following lemma:

Lemma 7.5 *If $(x,b) \in \mathcal{F}_{\alpha_0}$ is chosen u.a.r., then*

$$\mathop{\mathbb{E}}_{(x,b)\in_{u.a.r.}\mathcal{F}_{\alpha_0}}[cost(\alpha_0)\cup\{x\mapsto b\}] \leq cost(\alpha_0) - \frac{|V_{fr}(\alpha_0)|}{n(\alpha_0)} - \frac{|V_{nf}(\alpha_0)|S_k^{(D)}}{n(\alpha_0)}.$$

Proof We analyze the cost decrease contribution of the different types of variables separately. Each variable forced in state α_0 contributes zero cost both before and after the step. For a non-frozen variable $y \in \mathrm{V_{nf}}(\alpha_0)$, with probability $1/n(\alpha_0)$, the variable is chosen next, and its cost drops from $S_k^{(D)}$ to 0. For a frozen variable, the probability that y is guessed is reduced by $1/n(\alpha_0)$ after one step, because with probability $1/n(\alpha_0)$, y comes next in π and is guessed now. This $1/n(\alpha_0)$ is counted in $\mathrm{cost}(\alpha_0)$ but not in $\mathbb{E}[\mathrm{cost}(\alpha_0\cup\{x\mapsto b\}]$. Hence, adding over all variables, we get the desired result. □

Now we put everything together. Our expression becomes

$$p(\alpha_0) \geq 2^{\log\left(1-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{2n(\alpha_0)}\right)-\mathrm{cost}(\alpha_0)+\frac{|\mathrm{V_{fr}}(\alpha_0)|}{n(\alpha_0)}+\frac{|\mathrm{V_{nf}}(\alpha_0)|S_k^{(D)}}{n(\alpha_0)}},$$

and so we want to prove that

$$\log\left(1-\frac{|\mathrm{V_{fr}}(\alpha_0)|}{2n(\alpha_0)}\right)+\frac{|\mathrm{V_{fr}}(\alpha_0)|}{n(\alpha_0)}+\frac{|\mathrm{V_{nf}}(\alpha_0)|S_k^{(D)}}{n(\alpha_0)} \geq 0.$$

Observe that $|\mathrm{V_{fr}}(\alpha_0)|$ can only go from 0 to $n(\alpha_0)$. Hence, it's enough to prove that, for $0 \leq x \leq 1$,

$$f(x) := \log\left(1-\frac{x}{2}\right)+x \geq 0.$$

We have $f(0) = f(1) = 0$, and

$$f'(x) = \frac{1}{\ln 2(x-2)} + 1\cdot,$$

which is positive for $x = 0$ and admits a single zero between 0 and 1 for $x = 2 - \frac{1}{\ln 2}$, which concludes the proof, as f starts from 0, is increasing from 0 to its maximum, and then decreases to 0.

Also observe that this proof does not rely on the value of $S_k^{(D)}$. If Conjecture 7.4 does not completely hold, we may be able to use this leeway in order to prove Conjecture 7.3.

7.2.3 Conclusion and outlook

If we want to keep $\Pr_\pi[x \in \text{Guessed}(F, \alpha_0, \alpha(\pi), \pi, D)]$ as the cost for frozen, non forced variables, then Conjecture 7.4 must be proven for the complete proof to work. It may however be that this is not the right way to define costs for PPSZ$^{\text{O}}$, and that another cost function definition may allow us to conclude the proof of Conjecture 7.3.
As for the situation for a poor oracle, the positive point is that the current proof for the multiple assignment case does not a priori depend on the place of the non-frozen variable or on the value that we set for said non-frozen variable. The negative point is obviously that these issues may arise in the proof of Conjecture 7.4. Moreover, even if this conjecture holds, it may not translate easily to a version for a poor oracle. If an analogue of Conjecture 7.4 holds for a poor oracle, though, then the analogue of Conjecture 7.3 would hold for a poor oracle as well, since nothing in our proof depends on the specifics of the oracle.

Chapter 8

Conclusion

8.1 Results

In this thesis, we have explored a number of avenues relating to the derandomization of PPSZ. We started by studying the previous work on the derandomization of PPZ by Paturi, Pudlák and Zane [12], and on the derandomization of PPSZ in the unique satisfying assignment case by Rolf [13]. We then provided an alternative proof of the running time of PPSZ in the randomized case: this proof allows us to give bounds for the probability that a given satisfying assignment is returned, and it implies the previous bound by Hertli [5]; this alternative proof might be useful to tackle other questions related to PPSZ and its analysis.

To derandomize PPSZ, we need to derandomize the bits used for the permutation, the bits used for guessing the forced variables, and the bits used to set the non-frozen variables. As a first step, we tried to answer the question whether we could prove bounds for the randomized PPSZ algorithm when considering smaller permutation sets than the complete set of permutations of the set of variables of a formula; for a small enough set, the idea would then be to enumerate all possibilities. This approach was used with some measure of success in [12] and [13]. We have not managed to derive bounds for this case, but we now have some idea of why this problem is difficult to tackle: it is hard to control the inter-variable correlations if the permutation set is not uniform, and limited independence does not seem to be sufficient to guarantee this much in this regard; and without these guarantees, the analysis of that case (and whether or not PPSZ works at all with a smaller set of permutations) has so far been outside of our grasp.

For the non-frozen variables, our approach was to try to consider them set externally to an arbitrary value by some oracle. Since a non-frozen variable can, by definition, be set to any value when it is encountered, it seems like a reasonable assumption that the success probability of the algorithm would not be negatively affected. We have shown that it is indeed the case for the

PPZ algorithm for the 0-oracle; the full proof of this for the PPSZ algorithm still eludes us: when conjecturing that the probability of a frozen variable to be guessed stayed the same, we were able to prove that, independently of the considered oracle, the probability of success for PPSZ stayed the same as well, but the proof of the conjecture has resisted our attempts so far, even for the 0-oracle, and there is no guarantee that a proof for a 0-oracle would translate well into a proof for a weaker oracle that assigns arbitrary values to the non-frozen variables.
That resistance seems to come at least in part from the same issue as the derandomization of the permutation set. Setting non-frozen variables to fixed values partitions the permutation set into permutations that lead to a given satisfying assignment; but within one of these sets, we miss the independence and uniformity properties that we would like to have for the analysis in its current iteration.
We did not explore much the direction of building such an oracle; such a direction could raise interesting questions for future work on the topic.

8.2 Future directions

In this section, we suggest some directions for future work in the endeavour of derandomizing PPSZ.

8.2.1 Building an oracle for non-frozen variables

Assuming that our proof plan can yield something, one direction that we did not investigate much is the building of an oracle for non-frozen variables. It is unclear if it would be easier to consider the problem of frozen guessed variables separately or as part of this oracle. It is not clear either whether any oracle can be built in any reasonable (subexponential or low-exponential) amount of time. There are, however, arguments in favor of the feasability of such a task. If a formula has a lot of frozen variables, then the satisfying assignment can be coded with a small number of bits, corresponding to the non-frozen variables (of which we do not have too many by hypothesis), and to the frozen guessed variables (of which we should not have too many either because the frozen variables should get forced at some point, at least for some permutation). If we do not need too many bits, then we can allow ourselves to consider all possible bitstrings of the corresponding length. On the other hand, if a formula has a lot of satisfying assignments, then there exists a relatively small set of assignments that is guaranteed to contain a satisfying assignment, as proven in this lemma:

Lemma 8.1 *Consider a $\mathcal{F}$ the set of $\leq$ k-CNF formula which have at least 2^{cn} satisfying assignments, where $0 < c \leq 1$ is a constant. Then exists a set of assignments*

$\mathcal{A}$ of size $\mathcal{O}(2^{(1-c)n}n^k)$ such that, for every $F \in \mathcal{F}$, there exists $\alpha \in A$ such that α satisfies F.

Proof The proof relies on a probabilistic argument. Suppose that we pick ℓ assignments uniformly at random from all the 2^n possible assignments over n variables. The probability that a fixed formula F is satisfied by a random assignment α is at least $2^{(c-1)n}$; hence the probability that F is not satisfied by a set of ℓ random assignments is $(1-2^{(c-1)n})^\ell$. The expected number of formulas from $\mathcal{F}$ that are not satisfied by any assignment from the ℓ assignments is $|\mathcal{F}| \cdot (1-2^{(c-1)n})^\ell \leq |\mathcal{F}| \cdot e^{-2^{(c-1)n}.\ell}$. This means that if $\ell > 2^{(1-c)n} \cdot \ln|\mathcal{F}|$, then the expected number of unsatisfied formulas is strictly less than 1. The size of $\mathcal{F}$ has, as an upper bound, the total number of $\leq k$-CNF formulas; the total number of $\leq k$-CNF formulas is bounded by $2^{(2n)^k}$ (every $\leq k$-clause contains up to k literals that are chosen in a size of size $2n$, accounting for positive and negative literals, and every formula can choose any number of these $\leq k$-clauses). Hence, choosing $\ell \geq 2^{(1-c)n}(2n)^k$ is sufficient for the lemma to hold. □

This proof, however, does not allow us to explicitely construct the set $\mathcal{A}$. Moreover, there is no guarantee that a formula that has a lot of non-frozen variables also has a lot of satisfying assignments. To pick a very simplistic example, when starting to process a formula that only has the all-0 and the all-1 assignment as satisfying assignments, that formula actually has n non-frozen variables. Of course, in this case, setting the first variable will freeze all the other variables; but it may be possible to build a set of assignments that is small and that contains a high number of non-frozen variables that stay non-frozen for some non-trivial number of iterations. Two approaches can be then attempted. It may be possible to prove that we can also cover that case with a relatively small number of bitstrings. It might also be possible to show that such cases do not happen in the realm of k-SAT; this direction of work would probably steer towards questions around the possible structure of satisfying assignments for k-SAT, which is also a topic of interest (see for instance [4] for a discussion of the connectivity of the solution space).

Finally, it might be helpful to consider the building of an oracle not as a complete string, but as a set of block strings. The idea would be to split the variables into blocks of size $\sqrt{n}$ originally. This does not help us much per se, because if we consider all possible oracles for $\sqrt{n}$ variables and repeat that for each $\sqrt{n}$ blocks, we still end up with the same number of possibilities. However, considering blocks allows us to easily interleave a "forcing" operation: before trying the oracles for a given block, we first start by setting to their values all the variables from that block that are forced at that point ("block-forced", in the terminology of Section 6.2.3 on page 44). Since the block size is $o(n)$, the probability that a frozen variable is block-forced has the same upper bound as in the usual analysis, up to a different epsilon. The idea, when using these blocks, is to reduce the size of the oracle for the

blocks that come late in the permutation when a lot of variables are frozen (because then we will have a lot of block-forced variables, which we do not need in the oracle). Obviously, if there is a large number of non-frozen variables, then this approach will not force a lot of variables, but this may help the analysis to go through if considering a case distinction approach.
To summarize, here are a few questions that can be asked around the concept of oracle for non-frozen variables:

- Can Lemma 8.1 be transformed into a deterministic construction of a set $\mathcal{A}$?
- What can be done for the case where there is a lot of non-frozen variables but only a few satisfying assignments? Can such a case even happen?
- Is it possible to deterministically and efficiently (for some definition of efficiently) build an oracle for non-frozen variables?
- Would some insight on the structure of the solution space for k-SAT help to build such an oracle?

8.2.2 Other questions

The oracle situation aside, this thesis also allowed us to formulate a number of other questions:

- Can PPZ be fully derandomized, i.e. with the same runtime as the randomized version expected runtime?
- Can PPSZ be executed with a smaller set of permutations with the same success probability, or is there a counter-example that proves that it cannot?
- Can PPZ with a "nice" oracle be executed with a smaller set of permutations, or is there a counter-example that proves that it cannot?
- Can PPSZ be run with an oracle (be it "nice" or "poor") without affecting its success probability too much? Furthermore, is it possible to do so with a smaller set of permutations?
- Can another approach to the problem yield better results than what we have tried in this thesis? In particular, there has been some previous work on the reduction of k-SAT to Unique-k-SAT ([2]); a deterministic polynomial reduction from k-SAT to Unique-k-SAT, if such exists, would directly yield a derandomization of PPSZ.

It is our hope that this thesis started paving the way for future research, and that the questions we ask here will eventually find an answer.

Appendix A

Auxilliary statements and deferred proofs

Unless specified otherwise, the results of this appendix are stated and proven as is in [17].

A.1 Jensen inequality

Theorem A.1 *Let I be a real interval. If $f : I \to \mathbb{R}$ is a convex function and X is a random variable that attains values in I only, then*

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X]),$$

provided that both expectations exist.

Proof Let $\mu = \mathbb{E}[X]$. Let λ be such that $f(x) \geq f(\mu) + \lambda(x - \mu)$ for all $x \in I$. λ exists because f is convex. Due to linearity of expectation,

$$\mathbb{E}[f(X)] \geq \mathbb{E}[f(\mu) + \lambda(X - \mu)] = f(\mu) + \lambda(\mathbb{E}[X] - \mu) = f(\mu) = f(\mathbb{E}[X]),$$

and we are done. □

We can give the analogue for concave functions:

Theorem A.2 *Let I be a real interval. If $f : I \to \mathbb{R}$ is a concave function and X is a random variable that attains values in I only, then*

$$\mathbb{E}[f(X)] \leq f(\mathbb{E}[X]),$$

provided that both expectations exist.

Proof Let $\mu = \mathbb{E}[X]$. Let λ be such that $f(x) \leq f(\mu) + \lambda(x - \mu)$ for all $x \in I$. λ exists because f is concave. Due to linearity of expectation,

$$\mathbb{E}[f(x)] \leq \mathbb{E}[f(\mu) + \lambda(X - \mu)] = f(\mu) + \lambda(\mathbb{E}[x] - \mu) = f(\mu) = f(\mathbb{E}[X]),$$

and we are done. □

A.2 Binary coefficients and entropy function

Lemma A.3 *For $n \geq 1$ and $0 < \varepsilon < 1/2$, it holds that*

$$\sum_{i=0}^{\lfloor \varepsilon n \rfloor} \binom{n}{i} \leq 2^{H(\varepsilon)n},$$

where

$$H(\varepsilon) = -\varepsilon \log \varepsilon - (1-\varepsilon)\log(1-\varepsilon).$$

Proof We have

$$\begin{aligned}
1 &= (\varepsilon + (1-\varepsilon))^n \\
&= \sum_{i=0}^{n} \binom{n}{i} \varepsilon^i (1-\varepsilon)^{n-i} && \text{(binomial theorem)} \\
&= \sum_{i=0}^{n} \binom{n}{i} (1-\varepsilon)^n \left(\frac{\varepsilon}{1-\varepsilon}\right)^i \\
&\geq \sum_{i=0}^{\lfloor \varepsilon n \rfloor} \binom{n}{i} (1-\varepsilon)^n \left(\frac{\varepsilon}{1-\varepsilon}\right)^i && \text{(truncation of the sum)} \\
&\leq \sum_{i=0}^{\lfloor \varepsilon n \rfloor} \binom{n}{i} (1-\varepsilon)^n \left(\frac{\varepsilon}{1-\varepsilon}\right)^{\varepsilon n} && \left(\frac{\varepsilon}{1-\varepsilon} \leq 1 \text{ and } i \leq \varepsilon n\right) \\
&= \sum_{i=0}^{\lfloor \varepsilon n \rfloor} \binom{n}{i} \varepsilon^{\varepsilon n} (1-\varepsilon)^{n-\varepsilon n},
\end{aligned}$$

which, combined with

$$2^{-nH(\varepsilon)} = \varepsilon^{n\varepsilon}(1-\varepsilon)^{n(1-\varepsilon)},$$

yields the desired statement. □

A.3 FKG inequality

Let $\mathcal{A} = \{A_1, A_2, ..., A_r\}$ 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 $E = \{\{A \in \mathcal{A} \mid A = 1\} \in S_E\}$, or, informally speaking, if knowing the values of $\mathcal{A}$ leads to knowing whether E occurs. Moreover, if S_E is upwards heritary, 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.4 (FKG inequality) *Let $\mathcal{A} = \{A_1, A_2, ..., A_r\}$ be a collection of independent binary random variables and E_1 and E_2 events which are determined by $\mathcal{A}$ and monotonically increasing in $\mathcal{A}$. Then*

$$\Pr[E_1 \wedge E_2] \geq \Pr[E_1] \cdot \Pr[E_2].$$

Proof We proceed by induction on r.
For the base case, let $r = 1$. Then, 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 $A_1 = 1$. If $E_1 = E_2$, the statement is trivial. Let now E_1 be the first of these cases and E_2 be the other. 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:

$$\begin{aligned}\Pr[E_1] \cdot \Pr[E_2] = &(p \Pr[E_1 \mid A_1 = 1] + (1-p) \Pr[E1 \mid A_1 = 0]) \\ &\cdot (p \Pr[E_2 \mid A_1 = 1] + (1-p) \Pr[E2 \mid A_1 = 0])\end{aligned}$$

If we denote $e_{ij} = \Pr[E_i | A_1 = j]$, we get

$$\Pr[E_1] \cdot \Pr[E_2] = p^2 e_{11} e_{21} + p(1-p)(e_{11}e_{20} + e_{10}e_{21}) + (1-p)^2 e_{10} e_{20}.$$

The mutual independence of $\mathcal{A}$ 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 weights so that the larger summand gets the larger weight, the weighted sum can only increase. This yields

$$\begin{aligned}\Pr[E_1] \cdot \Pr[E_2] &\leq p^2 e_{11} e_{21} + p(1-p)(e_{10}e_{20} + e_{11}e_{21}) + (1-p)^2 e_{10} e_{20} \\ &= p e_{11} e_{21} + (1-p) e_{10} e_{20}.\end{aligned}$$

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 $A_2, A_3, ...A_r$ 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

$$e_{11} e_{21} \leq \Pr[E_1 \wedge E_2 \mid A_1 = 1].$$

Analogously:

$$e_{10}e_{20} \leq \Pr[E_1 \wedge E_2 \mid A_1 = 0].$$

Consequently, we get

$$\begin{aligned}&\Pr[E_1] \cdot \Pr[E_2] \\ &\leq p \Pr[E_1 \wedge E_2 \mid A_1 = 1] + (1-p) \Pr[E_1 \wedge E_2 \mid A_1 = 0] = \Pr[E_1 \wedge E_2]\end{aligned}$$

as desired, by applying once more the law of total probability. □

A.4 An inequality about logarithms

Lemma A.5 *For $x \geq 0$, we have*

$$\log_d(1+x) \geq \log_d(e)\frac{x}{1+x}.$$

Proof As $\log_d(x)$ is an antiderivative of $\log_d(e)1/x$ and $\log(1) = 0$, we have

$$\log_d(1+x) = \int_1^{1+x} \log_d(e)\frac{1}{t}\,\mathrm{d}t \geq \int_1^{1+x} \log_d(e)\frac{1}{1+x}\,\mathrm{d}t = \log_d(e)\frac{x}{1+x}.$$

as desired. □

A.5 w-wise independent probability spaces

This section aims at giving a proof of the following theorem:

Theorem 3.1 ([1],[13]) *For every n, w such that $1 \leq w \leq n$, there exists a probability space $\Omega(n,w)$ of size $\mathcal{O}(n^{w/2})$ and w-wise independent random variables $y_1, ..., y_n$ over $\Omega(n,w)$, each of which takes value 0 or 1 with probability $1/2$. $\Omega(n,w)$ can be constructed in polynomial time.*
This theorem is a corollary of the proof of the following theorem:

Theorem A.6 ([1]) *Suppose $n = 2^k - 1$ and $d = 2t+1$. Then there exists a symmetric probability space Ω of size $2(n+1)^t$ and d-w-wise independent random variables $y_1, ..., y_n$ over Ω each of which takes the values 0 and 1 with probability $\frac{1}{2}$. The space and the variables are explicitly constructed, given a representation of the field $F = GF(2^k)$ as a k-dimensional algebra over $GF(2)$.*

Proof ([1]) Let $x_1, ..., x_n$ be the n nonzero elements of F, represented as column vectors of length k over $GF(2)$. Let H be the following $1+kt$ by n matrix over $GF(2)$:

$$H = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_2^3 & x_2^3 & \dots & x_n^3 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{2t-1} & x_2^{2t-1} & \dots & x_n^{2t-1} \end{pmatrix}$$

The following lemma holds:

Lemma A.7 ([1]) *Any set of $d = 2t + 1$ columns of H is linearly independent over $GF(2)$.*

Proof ([1]) Let $J \subset \{1, 2, ..., n\}$ be a subset of cardinality $|J| = 2t + 1$ of the set of indices of the columns of H. Suppose that $\sum_{i \in J} z_j H_j = 0$, where H_j denotes the jth column of H and $z_j \in GF(2)$. We want to prove that $z_j = 0$ for all $j \in J$. By the assumption,

$$\sum_{j \in J} z_j x_j^i = 0 \tag{A.1}$$

for $i = 0$ and for every odd i satisfying $1 \leq i \leq 2t - 1$ (because these are the powers of x that exist in H). Suppose, now, that $a = 2^b \cdot \ell$, where $\ell \leq 2t - 1$ is an odd number. By squaring the equation (A.1) b times, where $i = \ell$, using the fact that $(u + v)^2 = u^2 + v^2 \mod 2$ and the fact that since each z_j is either 0 or 1, the equality $z_j = z_j^2$ holds for all j, we conclude that the equation $(A.1)$ holds for $i = a$. Consequently, equation (A.1) holds for all $0 \leq i \leq 2t$. This is a homogeneous system of $2t + 1$ linear equations in $2t + 1$ variables. The matrix of the coefficients is a Vandermonde matrix, which is nonsingular. Thus the only solution is the trivial one $z_j = 0$ for all $j \in J$, completing the proof of the lemma. □

Returning to the proof of the theorem, we define $\Omega = \{1, 2, ..., 2(n+1)^t\}$ and let $A = (a_{ij}), i \in \Omega, 1 \leq j \leq n$ be the $(0, 1)$-matrix whose $2(n + 1)^t = 2^{kt+1}$ rows are all the linear combinations (over $GF(2)$) of the rows of H. The sample space Ω is now endowed with the uniform probability measure, and the random variable y_j is defined by the formula $y_j(i) = a_{ij}$ for all $i \in \Omega, 1 \leq j \leq n$. It remains to show that the variables y_j are d-wise independent, and that each of them takes the values 0 and 1 with equal probability. For this, we have to show that, for every set J of up to d columns of A, the rows of the $|\Omega|$ by $|J|$ submatrix $A_J = (a_{ij})$, $i \in \Omega$, $j \in J$ take on each of the $2^{|J|}$ $(0, 1)$ vectors of length J equally often. By Lemma A.7, the columns of the corresponding submatrix H_J of H are linearly independent. The number of rows of A_J that are equal to any given vector is precisely the number of linear combinations of the rows of H_J that are equal to this vector. This number is the number of solutions of a system of $|J|$ linearly independent linear equations in $kt + 1$ variables, which is $2^{kt+1-|J|}$, independent of the vector of free coefficients. This completes the proof of the theorem. □

A.6 A correlation inequality

Lemma 5.3 ([17]) *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 $\mathbb{E}[A] = \bar{a}$ and $\mathbb{E}[B] = \bar{b}$. Then*

$$\mathbb{E}[A \cdot B] \leq a\bar{b} + b\bar{a} - ab.$$

Proof We can write

$$\mathbb{E}[A \cdot B] = \mathbb{E}[(A - a) \cdot B] + a\,\mathbb{E}[B]$$

and then use $B \leq b$ and $A \geq a$ to obtain

$$\mathbb{E}[A \cdot B] \leq b\,\mathbb{E}[A - a] + aE[B] = b\bar{a} - ba + a\bar{b},$$

as claimed. □

Bibliography

[1] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley, second edition, 2004.

[2] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique *k*-SAT: An isolation lemma for k-CNFs. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 135–141. IEEE, 2003.

[3] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[4] P. Gopalan, Ph. G. Kolaitis, E. N. Maneva, and C. H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. In *In Proceedings of ICALP 2006*, pages 346–357, 2006.

[5] Timon Hertli. 3-SAT faster and simpler – Unique-SAT bounds for PPSZ hold in general. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 277–284. IEEE, 2011.

[6] Isabelle Hurbain. Understanding the PPSZ algorithm for ClSP. Semester thesis, Eidgenössische Technische Hochschule ETH Zürich, 2013.

[7] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 653–662. IEEE, 1998.

[8] Leonid Levin. Universal search problems (Russian: Универсальные задачи перебора, Universal'nye perebornye zadachi). *Problems of Information Transmission (Russian:* Проблемы передачи информации, *Problemy Peredachi Informatsii)*, 9(3):115–116, 1973. Translated to English by

Trakhtenbrot, B. A. (1984). "A survey of Russian approaches to perebor (brute-force searches) algorithms". Annals of the History of Computing 6 (4): 384–400.

[9] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing HSSW algorithm for 3-SAT. In *Computing and Combinatorics*, pages 1–12. Springer, 2011.

[10] Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's *k*-SAT algorithm. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 245–252. ACM, 2011.

[11] Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for *k*-SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.

[12] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 566–574. IEEE, 1997.

[13] Daniel Rolf. Derandomization of PPSZ for Unique-*k*-SAT. In *Theory and applications of satisfiability testing*, pages 216–225. Springer, 2005.

[14] Dominik A. Scheder. *Algorithms and Extremal Properties of SAT and CSP*. PhD thesis, Eidgenössische Technische Hochschule ETH Zürich, Zürich, 2011.

[15] Uwe Schöning. A probabilistic algorithm for *k*-SAT and constraint satisfaction problems. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 410–414. IEEE, 1999.

[16] Mark N. Wegman and J. Lawrence Carter. New classes and applications of hash functions. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 175–182. IEEE, 1979.

[17] Emo Welzl. Boolean Satisfiability – Combinatorics and Algorithms, Lecture notes, 2013.

[18] Emo Welzl and Robin Moser. Satisfiability of Boolean Formulas – Special assignment set 3 – Exercise 2, 2012. http://www.ti.inf.ethz.ch/ew/courses/SAT12/spa3.pdf.