3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General

Timon Hertli
timon.hertli@inf.ethz.ch

Institute for Theoretical Computer Science
Department of Computer Science
ETH Zürich, 8092 Zürich, Switzerland

May 28, 2018

Abstract

The PPSZ algorithm by Paturi, Pudlák, Saks, and Zane [7] is the fastest known algorithm for Unique k-SAT, where the input formula does not have more than one satisfying assignment. For $k \geq 5$ the same bounds hold for general k-SAT. We show that this is also the case for $k = 3, 4$, using a slightly modified PPSZ algorithm. We do the analysis by defining a *cost* for satisfiable CNF formulas, which we prove to decrease in each PPSZ step by a certain amount. This improves our previous best bounds with Moser and Scheder [2] for 3-SAT to $O(1.308^n)$ and for 4-SAT to $O(1.469^n)$.

1 Introduction

k-SAT and especially 3-SAT is one of the most prominent NP-complete problems. While a polynomial algorithm seems very unlikely, much effort has been put into finding "moderately exponential algorithms", i.e. algorithms running in time $O(c^n)$ for $c < 2$, where n denotes the number of variables of the input formula. Unique k-SAT is the variant of the k-SAT problem where the input CNF formula is promised to have a unique or no satisfying assignment. In 1998, Paturi, Pudlák, Saks, and Zane [7] presented a randomized algorithm for Unique 3-SAT that runs in time $O(1.30704^n)$, where n is the number of variables of the formula. For general 3-SAT, only a running time of $O(1.3633^n)$ could be shown using a complicated analysis. Shortly afterwards, Schöning [10] proposed a very simple algorithm with running time $O(1.33334^n)$ for 3-SAT. In 2004, Iwama and Tamaki [4] showed that Schöning's algorithm can be combined with PPSZ to get a running time of $O(1.32373^n)$[1]. This bound was subsequently improved by Rolf [9], Iwama, Seto, Takai, and Tamaki [3] and in [2] with Moser and Scheder to $O(1.32216^n)$, $O(1.32113^n)$, $O(1.32065^n)$, respectively. However, these bounds are still far from the bound $O(1.30704^n)$ for Unique 3-SAT.

PPSZ [7] does the following: First, the input formula $\mathbf{F}$ is preprocessed by s-bounded resolution, meaning that all clauses obtainable by resolution when clauses of size at most s are considered are added to $\mathbf{F}$. Then PPSZ goes through the variables in random order. In each step, a variable x is permanently replaced by a Boolean value a as follows: If

[1] Using the new version of [7] immediately gives the bound $O(1.32267^n)$, as stated in [9].

there a clause $\{x\}$ or $\{\bar{x}\}$ in the current formula, then a is chosen accordingly and we call x *forced*. Otherwise a is chosen uniformly at random from $\{0,1\}$ and we call x *guessed*. In Unique k-SAT it was shown that the probability that a variable is guessed is bounded from above by some quantity S depending on k on ϵ. Using this, it is not hard to show that there exists an algorithm for Unique k-SAT running in time $O(2^{S\cdot n})$.

1.1 Our Contribution

We give an analysis of a slightly adapted PPSZ algorithm that achieves the same bound for k-SAT as for Unique k-SAT, which gives new bounds for $k = 3, 4$. The previously best known bounds from [2] are improved for 3-SAT from $O(1.32065^n)$ to $O(1.30704^n)$ and for 4-SAT from $O(1.46928^n)$ to $O(1.46899^n)$.

Theorem 1. *There exists a randomized algorithm for 3-SAT with one-sided error that runs in time $O(1.30704^n)$.*

Theorem 2. *There exists a randomized algorithm for 4-SAT with one-sided error that runs in time $O(1.46899^n)$.*

Our analysis is directly based on the analysis for Unique k-SAT of [7], we do not use the part that considers general k-SAT. Let $\mathbf{F}$ be a satisfiable CNF formula. A variable x of F is called *frozen*[2] if it has the same value in all satisfying assignments, and *non-frozen* otherwise. In [2] it was shown how to use this distinction to improve PPSZ: A frozen variable is good for PPSZ, while assigning any Boolean value to a non-frozen variable preserves satisfiability. Using a preprocessing step, about half of the variables can be assumed to be frozen for PPSZ. In this paper, we use a similar but more fine-grained approach:

We assign to each k-CNF formula $\mathbf{F}$ a *cost* $c(\mathbf{F}) \leq S \cdot n$, where S is the upper bound that a variable is guessed in Unique k-SAT. The main theorem then shows that PPSZ finds a satisfying assignment of $\mathbf{F}$ with probability at least $2^{-c(\mathbf{F})}$. By a routine argument this gives a randomized algorithm with running time $O(2^{S\cdot n})$. Using the cost function, we are able to consider just one PPSZ step. We show that the cost decreases, depending on how many variables are frozen. If all variables are frozen, the cost decreases by 1. If some variables are non-frozen, the decrease is smaller, but due to the presence of non-frozen variables, satisfiability is preserved with higher probability in this step. With this, the theorem can be obtained by induction.

In Section 2, we review the PPSZ algorithm and prove its properties we need. In Section 3, we introduce the cost function and do the analysis using the statements of Section 2.

1.2 Notation

We adapt the notational framework as used in [11]. Let V be a finite set of propositional *variables*. A *literal* l over $x \in V$ is a variable x or a complemented variable $\bar{x}$. If $l = \bar{x}$, then $\bar{l}$, the complement of l, is defined as x. We assume that all literals are distinct. A *clause* over V is a finite set of literals over pairwise distinct variables from V. A formula in *CNF* (Conjunctive Normal Form) is a pair $\mathbf{F} = (F, V)$ where F is a finite set of clauses over V. V is the set of variables of $\mathbf{F}$ and denoted by $V(\mathbf{F})$. We define $n(\mathbf{F}) := |V(\mathbf{F})|$, the number of variables of $\mathbf{F}$. If $\mathbf{F}$ is understood from the context, we sometimes write

[2] We previously called frozen variables *critical*. Such variables are also referred to as *backbone*. Thanks to Ramamohan Paturi for making us aware of the existing terminology.

n for $n(\mathbf{F})$. A clause containing exactly one literal is called a *unit clause*. We say that $\mathbf{F} = (F, V)$ is a $(\leq k)$-*CNF* formula if every clause of F has size at most k.

Let V be a finite set of variables. A (truth) *assignment* on V is a function $\alpha : V \to \{0, 1\}$ which assigns a Boolean value to each variable. A literal $u = x$ (or $u = \bar{x}$) is *satisfied by* α if $\alpha(x) = 1$ (or $\alpha(x) = 0$). A clause is *satisfied by* α if it contains a satisfied literal and a formula is *satisfied by* α if all of its clauses are. A formula is *satisfiable* if there exists a satisfying truth assignment to its variables. Given a CNF formula $\mathbf{F}$, we denote by $\text{sat}(\mathbf{F})$ the set of assignments on $V(\mathbf{F})$ that satisfy $\mathbf{F}$. *k-SAT* is the decision problem of deciding if a $(\leq k)$-CNF formula has a satisfying assignment.

Formulas can be manipulated by permanently assigning values to variables. If $\mathbf{F}$ is a given CNF formula and $x \in V(\mathbf{F})$ then assigning $x \mapsto 1$ satisfies all clauses containing x (irrespective of what values the other variables in those clauses are possibly assigned later) whilst it truncates all clauses containing $\bar{x}$ to their remaining literals. Additionally, x is removed from the variable set of $\mathbf{F}$. We will write $\mathbf{F}^{[x \mapsto 1]}$ (and analogously $\mathbf{F}^{[x \mapsto 0]}$) to denote the formula arising from doing just this. For notational convenience, we also write x for $x \mapsto 1$ and $\bar{x}$ for $x \mapsto 0$, i.e. we write a literal instead of a variable-value pair. With this, we can view an assignment α also as the set of literals l that are satisfied by α. If the literal l corresponds to $x \mapsto a$, we write $\mathbf{F}^{[l]}$ instead of $\mathbf{F}^{[x \mapsto a]}$. By choosing an element from a finite set u.a.r., we mean choosing it uniformly at random. Unless otherwise stated, all random choices are mutually independent. We denote by log the logarithm to the base 2. For the logarithm to the base e, we write ln.

2 The PPSZ Algorithm

In this section we present and slightly modify the PPSZ algorithm from [7]. We also introduce the concept of *frozen variables* from [2] (called critical variables there) and present the statements about the PPSZ algorithm we need later. To analyze PPSZ, we can ignore unsatisfiable formulas, as in that case PPSZ never returns a satisfying assignment. In the rest of this paper we fix an integer $k \geq 3$ and let $\mathbf{F} = (F, V)$ be an arbitrary satisfiable $(\leq k)$-CNF. For our analysis, we need to change the PPSZ algorithm slightly. The PPSZ algorithm was defined using a preprocessing step of s-bounded resolution, i.e. resolution when only considering clauses of size at most s. We change this to a weaker concept we call s-implication[3]. We call a literal s-implied if it is implied by a subformula with at most s clauses:

Definition 3. *Let $\mathbf{F} = (F, V)$ be a satisfiable CNF formula. We say that a literal l is* s-implied *by $\mathbf{F}$ if there is a subset G of F with $|G| \leq s$ such that all satisfying assignments of $\mathbf{G} = (G, V)$ set l to 1. For notational convenience, we also say that a variable x is s-implied, if one of the literals x or $\bar{x}$ is s-implied.*

We call a CNF formula $\mathbf{F}$ s-implication free *if no literal l is s-implied.*

Algorithm 1 PPSZ(CNF formula $\mathbf{F}$, integer s)

Choose β u.a.r. from all assignments on $V(\mathbf{F})$
Choose π u.a.r. from all permutations of $V(\mathbf{F})$
return PPSZ($\mathbf{F}, \beta, \pi, s$)

[3]The concept of s-implication is from the lecture note draft by Dominik Scheder.

Algorithm 2 PPSZ(CNF formula $\mathbf{F}$, assignment β, permutation π, integer s)

$V \leftarrow V(\mathbf{F})$
Let α be a partial assignment over V, initially the empty assignment
for all $x \in V$, according to π **do**
  while there is an s-implied literal $l = y \mapsto a$ in $\mathbf{F}$ **do**
    $\mathbf{F} \leftarrow \mathbf{F}^{[y \mapsto a]}$
    $\alpha(y) \leftarrow a$
  end while
  if $x \in V(\mathbf{F})$ **then**
    $\mathbf{F} \leftarrow \mathbf{F}^{[x \mapsto \beta(x)]}$
    $\alpha(x) \leftarrow \beta(x)$
  end if
end for
return α

Our analysis requires the following modification of PPSZ: Instead of processing the variables strictly step by step, we check after each step for which variables we know the value (by s-implication) and immediately set these accordingly. While the change to s-implication makes PPSZ only weaker, this modification makes the algorithm stronger; our approach does not work with the PPSZ algorithm proposed in [7], more on this is written in the conclusion. In the following we fix s large enough for the bounds we want to show, as described later. It is easily seen that the PPSZ algorithm we present runs in polynomial time if s is a constant. We now give some definitions used in the analysis:

Definition 4 ([7])**.** *Let β and π be chosen randomly as in* PPSZ$(\mathbf{F}, s)$*. We define the* success probability *of PPSZ as the probability that it returns a satisfying assignment.*

$$p_{\text{success}}(\mathbf{F}, s) := \Pr_{\pi,\beta}\left(\text{PPSZ}(\mathbf{F}, \beta, \pi) \in \text{sat}(\mathbf{F})\right).$$

Consider a run of PPSZ$(\mathbf{F}, s)$*. For $x \in V(\mathbf{F})$ we call x* forced *if the value of x is determined by s-implication; we call it* guessed *otherwise. For $\alpha \in \text{sat}(\mathbf{F})$ we define the probability that x is guessed w.r.t. α as follows:*

$$p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) := \Pr_{\pi}\left(x \text{ is guessed in } \text{PPSZ}(\mathbf{F}, \alpha, \pi, s)\right).$$

To make notation easier, we extend the notation and allow α also to be over a variable set $W \supset V(\mathbf{F})$; the variables in $W \setminus V(\mathbf{F})$ will then be ignored. We also allow $x \notin V(\mathbf{F})$ and define in this case $p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) := 0$.

Definition 5 ([2])**.** *We say that $x \in V(\mathbf{F})$ is* frozen *if all satisfiable assingments of $\mathbf{F}$ agree on x. We say that x is* non-frozen *otherwise.*

The probability that a frozen variable is guessed can be bounded:

Theorem 6 ([7, 2])**.** *If x is a frozen variable, then $p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) \leq S_k + \epsilon_k(s)$, where $S_k := \int_0^1 \frac{t^{1/(k-1)} - t}{1-t} dt$ and $\epsilon(s)$ goes to 0 for $s \to \infty$.*

Proof Sketch. If there is a unique (or sufficiently isolated) satisfying assignment, then [7] gives us an upper bound for the probability that a variable is guessed. In [2], we have showed that this bound also holds for an arbitrary satisfying assignment, as long as the variable is *frozen*. It is easily seen that this bound holds if we use s-implication instead

of a preprocessing step of s-bounded resolution: In the analysis of [7], so-called critical clause trees are used to bound p_{guessed}. The only clauses D used in the proof there are these with a resolution deduction using at most m clauses of $\mathbf{F}$ with size at most k each, for some appropriately chosen constant m. Then D can be obtained by $(m \cdot k)$-bounded resolution from $\mathbf{F}$. This also means that D is implied by at most m clauses. If we restrict $\mathbf{F}$ to some literals and obtain $\mathbf{F}'$, then the clause D restricted to these literals is now implied by at most m clauses of $\mathbf{F}'$. Hence appearance of a unit clause in the algorithm of [7] now becomes s-implication of a literal here for all unit clauses considered in the analysis of [7]. □

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

k	S_k	2^{S_k}
3	0.3862944	1.307032
4	0.5548182	1.468984
5	0.6502379	1.569427
6	0.7118243	1.637874

In [7] it was shown using Jensen's inequality how to use a bound for the probability that a variable is guessed to give an upper bound for the running time of Unique k-SAT. For $k \geq 5$, the same bound holds also for k-SAT using a more elaborate argument.

Theorem 7 ([7]). *For $\epsilon > 0$, there exists a randomized algorithm for Unique k-SAT with one-sided error that runs in time $O(2^{S_k n + \epsilon n})$. For $k \geq 5$, this is also true for k-SAT.*

In this paper we prove that for k-SAT we have the same bound as Unique k-SAT for *all* k. For $k \geq 5$, this can be seen as an alternative proof for general k-SAT. Note however that because we immediately fix implied variables, the algorithm is slightly different. In the remainder of this section, we will introduce additional notation and state some properties of PPSZ we will need later.

Definition 8. *We denote the* non-frozen *variables of $\mathbf{F}$ by $V_{\mathrm{N}}(\mathbf{F})$ and set $n_{\mathrm{N}}(\mathbf{F}) := |V_{\mathrm{N}}(\mathbf{F})|$. We denote the* frozen *variables by $V_{\mathrm{F}}(\mathbf{F})$ and set $n_{\mathrm{F}}(\mathbf{F}) := |V_{\mathrm{F}}(\mathbf{F})|$. The* satisfying literals*, denoted by $\mathrm{SL}(\mathbf{F})$, are the literals l over $V(\mathbf{F})$ s.t. $\mathbf{F}^{[l]}$ is satisfiable.*

The satisfying literals consist of all literals over non-frozen variables, and for each frozen variable of the literal that corresponds to the satisfying assignments of $\mathbf{F}$. It follows that $|\mathrm{SL}(\mathbf{F})| = 2n_{\mathrm{N}}(\mathbf{F}) + n_{\mathrm{F}}(\mathbf{F})$.

The following alternative definition of $\mathrm{PPSZ}(\mathbf{F}, s)$ is easily seen to be the same algorithm. We will use this later to bound $p_{\text{success}}(\mathbf{F}, s)$.

Observation 9. *We can alternatively characterize $\mathrm{PPSZ}(\mathbf{F}, s)$ as follows: We first set all s-implied literals in $\mathbf{F}$ accordingly, and let α be the assignment consisting of these literals. $\mathbf{F}$ is now s-implication free. If $n(\mathbf{F}) = 0$, then we return α. Otherwise we choose x from $V(\mathbf{F})$ u.a.r. and a from $\{0, 1\}$ u.a.r. and let $l := x \mapsto a$. Then we run $\mathrm{PPSZ}(\mathbf{F}^{[l]}, s)$ and combine the returned assignment with $\alpha \cup \{l\}$.*

It follows that if $\mathbf{F}$ is s-implication free and if $n(\mathbf{F}) \geq 1$, then

$$p_{\text{success}}(\mathbf{F}, s) = \frac{1}{2n} \sum_{l \in \mathrm{SL}(\mathbf{F})} p_{\text{success}}(\mathbf{F}^{[l]}, s).$$

We need two statements about p_{guessed} for our proof. The first tells us that if we restrict $\mathbf{F}$ to a literal of α the probability that x is guessed w.r.t. α cannot increase.

Lemma 10. *For $l \in \alpha$ and $\alpha \in \mathrm{sat}(\mathbf{F})$, we have $p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s) \leq p_{\text{guessed}}(\mathbf{F}, x, \alpha, s)$.*

Proof. Let $l = y \mapsto \alpha(y)$. Assume x is s-implied in $\mathrm{PPSZ}(F, \alpha, \pi, s)$. Let π' be the permutation obtained by removing y from π. We claim that x is s-implied in $\mathrm{PPSZ}(F^{[l]}, \alpha, \pi', s)$: Consider the clause set G that s-implies x in $\mathrm{PPSZ}(F, \alpha, \pi, s)$. It follows from the definition of s-implication that restricting G to l gives a clause set that s-implies x, and hence x is s-implied in $\mathrm{PPSZ}(F^{[l]}, \alpha, \pi', s)$. The statement is now easily seen, as π' has the distribution of a permutation uniformly at random chosen from all permutations on $V(F) \setminus \{y\}$. □

The second statement allows us to relate the probability that a variable x of $\mathbf{F}$ is guesssed to the probability that x is guessed if $\mathbf{F}$ is restricted by a random literal of α. Intuitively, assume we have an upper bound for p_{guessed}. With some probability, x is guessed right now. Hence if x is not guessed right now, the probability to be guessed in the remainder must slightly decrease.

Lemma 11. *For $\alpha \in \mathrm{sat}(\mathbf{F})$ and $x \in V(\mathbf{F})$ s.t. x is not s-implied, we have*

$$p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) - \frac{1}{n} = \frac{1}{n(\mathbf{F})} \sum_{l \in \alpha} p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s).$$

Proof. Let π be a random permutation on $V(\mathbf{F})$ and let y be the variable that comes first in π. We have by definition

$$p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) = \Pr_{\pi}\left(x \text{ is guessed in } \mathrm{PPSZ}(\mathbf{F}, \alpha, \pi, s)\right).$$

By the law of total probability, this is

$$= \mathbf{E}_y \left[\Pr_{\pi}\left(x \text{ is guessed in } \mathrm{PPSZ}(\mathbf{F}, \alpha, \pi, s) \mid y \text{ comes first in } \pi\right)\right].$$

If $x = y$, then x is always guessed, as x is not s-implied. If $x \neq y$, then the probability under the expectation is easily seen to be $p_{\text{guessed}}(\mathbf{F}^{[y \mapsto \alpha(y)]}, x, \alpha, s)$. Writing the expectation as a sum and using that $p_{\text{guessed}}(\mathbf{F}^{[y \mapsto \alpha(y)]}, y, \alpha, s)$ is defined as 0 gives us

$$p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) = \Pr_y[x = y] 1 + \Pr_y[x \neq y] \frac{1}{n(\mathbf{F}) - 1} \sum_{l \in \alpha} p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s).$$

Trivially $\Pr_y[x = y] = \frac{1}{n(\mathbf{F})}$ and $\Pr_y[x \neq y] = \frac{n(\mathbf{F})-1}{n(\mathbf{F})}$; the statement follows now easily. □

3 Analysis using a Cost Function

To define the cost function, we first need to give a probability distribution on the set of all satisfying assignments of a CNF formula. We do this by defining a random process that repeatedly picks a satisfying literal:

Definition 12. *We define the random process* $\mathrm{AssignSL}(\mathbf{F})$ *that produces an assignment on $V(\mathbf{F})$ as follows: Start with the empty assignment α, and repeat the following step until $V(\mathbf{F}) = \emptyset$: Choose a satisfying literal $l \in \mathrm{SL}(\mathbf{F})$ and add l to α; then let $\mathbf{F} \leftarrow \mathbf{F}^{[l]}$. At the end, output α.*

Let α be an assignment on $V(F)$. Then $p(\mathbf{F}, \alpha)$ is defined as the probability that $\mathrm{AssignSL}(\mathbf{F})$ *returns α. If α is defined on some $W \supset V(F)$, $p(\mathbf{F}, \alpha)$ is defined as the probability that* $\mathrm{AssignSL}(\mathbf{F})$ *returns α restricted to $V(F)$.*

From the definition we observe the following:

Observation 13. AssignSL($\mathbf{F}$) *always returns a satisfying assignment of* F. *Furthermore* $p(\mathbf{F}, \alpha)$ *defines a probability distribution on* $\mathrm{sat}(\mathbf{F})$. *If* $n(\mathbf{F}) = 0$, *then* $p(\mathbf{F}, \alpha) = 1$. *Otherwise we have the relation*

$$p(\mathbf{F}, \alpha) = \frac{1}{|\mathrm{SL}(\mathbf{F})|} \sum_{l \in \alpha} p(\mathbf{F}^{[l]}, \alpha).$$

Note that this distribution is not the uniform distribution: As an example consider the CNF formula corresponding to $x \vee y$. The probability that both x and y are set to 1 is $1/4$, while the probability that exactly one of x and y is set to 0 is $3/8$ each.

Using the probability distribution $p(\mathbf{F}, \alpha)$, we define a cost function on satisfiable k-CNF formulas. In the following fix an integer $s \geq 0$ and let $S := S_k - \epsilon_k(s)$ s.t. $p_{\mathrm{guessed}}(\mathbf{F}, x, \alpha, s) \leq S$ for all satisfiable k-CNF $\mathbf{F}$ where x is frozen, as in Theorem 6.

Definition 14. *For a* $(\leq k)$*-CNF formula* $\mathbf{F}$ *with variable set* $V(\mathbf{F})$ *we define the* cost of x in $\mathbf{F}$ *as*

$$c(\mathbf{F}, x) := \begin{cases} 0 & \text{if } x \notin V(\mathbf{F}) \\ S & \text{if } x \in V_{\mathrm{N}}(\mathbf{F}) \\ \sum_{\alpha \in \mathrm{sat}(\mathbf{F})} p(\mathbf{F}, \alpha) p_{\mathrm{guessed}}(\mathbf{F}, x, \alpha, s) & \text{if } x \in V_{\mathrm{F}}(\mathbf{F}) \end{cases}$$

We define the cost of $\mathbf{F}$ *as* $c(\mathbf{F}) := \sum_{x \in V(\mathbf{F})} c(\mathbf{F}, x)$.

The cost of a variable that does not occur in the formula is set to 0 for notational convenience. It follows from the definition that $c(\mathbf{F}, x) \leq S$ and hence $c(\mathbf{F}) \leq n(\mathbf{F})S$. The cost function gives a lower bound on the success probability of PPSZ:

Theorem 15. $p_{\mathrm{success}}(\mathbf{F}, s) \geq 2^{-c(\mathbf{F})}$.

To obtain Theorems 1 and 2, we choose s such that $\epsilon_k(s)$ becomes small enough and $2^S < 1.30704$ for 3-SAT and $2^S < 1.46899$ for 4-SAT. By $O(2^{Sn})$ independent repetitions of PPSZ, the claimed randomized exponential algorithm can then be obtained by a routine argument. In the remainder of this section, we prove Theorem 15. We need the following lemma about $p(\mathbf{F}, \alpha)$:

Lemma 16. *For* $l \in \alpha$, *we have* $p(\mathbf{F}^{[l]}, \alpha) \geq p(\mathbf{F}, \alpha)$. *If* l *is over a frozen variable, then* $p(\mathbf{F}^{[l]}, \alpha) = p(\mathbf{F}, \alpha)$, *and* $c(\mathbf{F}^{[l]}) \leq c(\mathbf{F})$.

Proof. Consider AssignSL($\mathbf{F}$) given that l is chosen at some point of time, let α' denote the output. The distribution of $\alpha' \setminus \{l\}$ is the same as the output of AssignSL($\mathbf{F}^{[l]}$), as is easily checked by induction. If l is *not* chosen in AssignSL($\mathbf{F}$), then the output is never α. Hence the probability that AssignSL($\mathbf{F}^{[l]}$) returns $\alpha \setminus \{l\}$ is at least the probability that AssignSL($\mathbf{F}$) returns α. This proves the first statement. If l is over a frozen variable, then in AssignSL($\mathbf{F}$) l must be chosen at some point, and equality holds. The inequality on the costs now follows from Lemma 10. ☐

If $\mathbf{F}$ is s-implication free the cost decreases by a certain amount depending on how many variables are frozen and non-frozen:

Theorem 17. *Suppose* $\mathbf{F}$ *is* s*-implication free. For* l *chosen u.a.r from* $\mathrm{SL}(\mathbf{F})$, *we have*

$$\mathbf{E}_l[c(\mathbf{F}^{[l]})] \leq c(\mathbf{F}) - n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} - n_{\mathrm{F}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|}.$$

If all variables are frozen, the cost decreases by 1. If all variables are non-frozen, the cost decreases by $S < 1$. We will prove Theorem 17 later and use it now to prove Theorem 15:

Proof of Theorem 15. We prove $p_{\text{success}}(\mathbf{F}, s) \geq 2^{-c(\mathbf{F})}$ by induction on $n(\mathbf{F})$. If $n(\mathbf{F}) = 0$, the statement is trivial. Assume the statement holds for formulas with less than $n(\mathbf{F})$ variables, so that for $l \in \mathrm{SL}(\mathbf{F})$, we have $p_{\text{success}}(\mathbf{F}^{[l]}) \geq 2^{-c(\mathbf{F}^{[l]})}$. If $\mathbf{F}$ is *not* s-implication free, then let l be the first s-implied literal fixed in PPSZ such that $p_{\text{success}}(\mathbf{F}) = p_{\text{success}}(\mathbf{F}^{[l]})$. The literal l must be over a frozen variable, and from the last statement of Lemma 16 we have $c(\mathbf{F}) \geq c(\mathbf{F}^{[l]})$ and hence $2^{-c(\mathbf{F}^{[l]})} \geq 2^{-c(\mathbf{F})}$ and using the induction hypothesis we are done.

Now assume that $\mathbf{F}$ is s-implication free. Using Observation 9 and the induction hypothesis gives us

$$p_{\text{success}}(\mathbf{F}, s) = \frac{1}{2n(\mathbf{F})} \sum_{l \in \mathrm{SL}(\mathbf{F})} p_{\text{success}}(\mathbf{F}^{[l]}, s) \geq \frac{1}{2n(\mathbf{F})} \sum_{l \in \mathrm{SL}(\mathbf{F})} 2^{-c(\mathbf{F}^{[l]})}.$$

If we choose $l \in \mathrm{SL}(\mathbf{F})$ u.a.r. we can write the sum as an expectation and then use Jensen's inequality and obtain

$$p_{\text{success}}(\mathbf{F}, s) \geq \frac{|\mathrm{SL}(\mathbf{F})|}{2n(\mathbf{F})} \mathbf{E}_l \left[2^{-c(\mathbf{F}^{[l]})} \right] \geq \frac{|\mathrm{SL}(\mathbf{F})|}{2n(\mathbf{F})} 2^{-\mathbf{E}_l[c(\mathbf{F}^{[l]})]} = 2^{\log(\frac{|\mathrm{SL}(\mathbf{F})|}{2n(\mathbf{F})}) - \mathbf{E}_l[c(\mathbf{F}^{[l]})]}.$$

To prove the statement, we need to show that the exponent is at least $-c(\mathbf{F})$, i.e.

$$L := \log\left(\frac{|\mathrm{SL}(\mathbf{F})|}{2n(\mathbf{F})}\right) - \mathbf{E}_l[c(\mathbf{F}^{[l]})] + c(\mathbf{F}) \geq 0.$$

We bound the left-hand with Theorem 17 and obtain

$$\begin{aligned} L &\geq \log\left(\frac{|\mathrm{SL}(\mathbf{F})|}{2n(\mathbf{F})}\right) - c(\mathbf{F}) + n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} + n_{\mathrm{F}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|} + c(\mathbf{F}) \\ &= \log\left(\frac{|\mathrm{SL}(\mathbf{F})|}{2n(\mathbf{F})}\right) + n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} + n_{\mathrm{F}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|} \\ &= \log\left(\frac{|\mathrm{SL}(\mathbf{F})|}{n(\mathbf{F})}\right) - 1 + n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} + n_{\mathrm{F}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|}. \end{aligned}$$

Using twice $|\mathrm{SL}(\mathbf{F})| = n_{\mathrm{F}}(\mathbf{F}) + 2n_{\mathrm{N}}(\mathbf{F})$, this is

$$= \log\left(1 + \frac{n_{\mathrm{N}}(\mathbf{F})}{n(\mathbf{F})}\right) + n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} - 2n_{\mathrm{N}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|}.$$

With the inequality $\log(1+x) \geq \log(e) \frac{x}{1+x}$ (which is easily seen by writing $\log(1+x)$ as an integral), we have

$$\begin{aligned} L &\geq \log(e) \frac{\frac{n_{\mathrm{N}}(\mathbf{F})}{n(\mathbf{F})}}{\frac{|\mathrm{SL}(\mathbf{F})|}{n(\mathbf{F})}} + n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} - 2n_{\mathrm{N}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|} \\ &= \log(e) \frac{n_{\mathrm{N}}(\mathbf{F})}{|\mathrm{SL}(\mathbf{F})|} - (2 - 2S) \frac{n_{\mathrm{N}}(\mathbf{F})}{|\mathrm{SL}(\mathbf{F})|}. \end{aligned}$$

It can be easily seen from the definition that S_k increases for larger k. Hence $S \geq S_3 = 2\ln 2 - 1 \approx 0.3863$ and $(2 - 2S) \leq 4 - 4\ln 2 < 1.23 < 1.44 < \log(e)$, which implies $L \geq 0$ and completes the proof. ∎

It is interesting to see that we still have some leeway in the last step. One checks that $\log(e) = (2 - 2\frac{S_3}{1+S_3})$ which means that our method works as long the upper bound S that a frozen variable is guessed is at least $\frac{S_3}{1+S_3} \approx 0.2787$, corresponding to an algorithm with running time roughly $O(1.214^n)$.

3.1 Remaining Proofs

We now need to prove Theorem 17. The theorem follows from the following two lemmas:

Lemma 18. *If $x \in V_{\mathrm{N}}(\mathbf{F})$, then for l chosen u.a.r from $\mathrm{SL}(\mathbf{F})$ we have*

$$\mathbf{E}_l[c(\mathbf{F}^{[l]}, x)] \leq c(\mathbf{F}, x) - \frac{2S}{|\mathrm{SL}(\mathbf{F})|}.$$

Note that this lemma holds even if $\mathbf{F}$ has s-implied literals. However, we only use it if $\mathbf{F}$ is s-implication free.

Proof. x is non-frozen, so by definition $c(\mathbf{F}, x) = S$. As the cost of any variable is at most S, we have

$$E_l[c(\mathbf{F}^{[l]}, x)] \leq S \Pr_l[x \in V(\mathbf{F}^{[l]})] = S \Pr_l[l \text{ is not over } x] = S\frac{|\mathrm{SL}(\mathbf{F})| - 2}{|\mathrm{SL}(\mathbf{F})|}.$$

□

Lemma 19. *If $x \in V_{\mathrm{F}}(\mathbf{F})$ and not s-implied, then for l chosen u.a.r from $\mathrm{SL}(\mathbf{F})$ we have*

$$\mathbf{E}_l[c(\mathbf{F}^{[l]}, x)] \leq c(\mathbf{F}, x) - \frac{1}{|\mathrm{SL}(\mathbf{F})|}.$$

Proof. By writing the expectation as a sum and inserting the definition, we have

$$\mathbf{E}_l[c(\mathbf{F}^{[l]}, x)] = \frac{1}{|\mathrm{SL}(\mathbf{F})|} \sum_{l \in \mathrm{SL}(\mathbf{F})} \sum_{\alpha' \in \mathrm{sat}(\mathbf{F}^{[l]})} p(\mathbf{F}^{[l]}, \alpha') p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha', s).$$

Note that we have extended p and p_{guessed} such that also assignments over a set $W \supset V(\mathbf{F}^{[l]})$ are allowed and the additional variables are ignored. We now claim that we can exchange the sums and get

$$\mathbf{E}_l[c(\mathbf{F}^{[l]}, x)] = \frac{1}{|\mathrm{SL}(\mathbf{F})|} \sum_{\alpha \in \mathrm{sat}(\mathbf{F})} \sum_{l \in \alpha} p(\mathbf{F}^{[l]}, \alpha) p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s).$$

To prove this, we need to show that there is a bijection between the set $\{(\alpha, l) \mid \alpha \in \mathrm{sat}(\mathbf{F}), l \in \alpha\}$ and the set $\{(l, \alpha') \mid l \in SL(\mathbf{F}), \alpha' \in \mathbf{F}^{[l]}\}$ s.t. $\alpha = \alpha'$ on $V(\mathbf{F}^{[l]})$. One can easily check that $f(\alpha, l) := (l, \alpha \setminus \{l\})$ with $f^{-1}(l, \alpha') = (\alpha' \cup \{l\}, l)$ is such a bijection.

Now the outer sum is over $\alpha \in \mathrm{sat}(\mathbf{F})$, as in the definition of $c(\mathbf{F}, x)$. Hence it is sufficient to prove that for all $\alpha \in \mathrm{sat}(\mathbf{F})$ we have

$$\frac{1}{|\mathrm{SL}(\mathbf{F})|} \sum_{l \in \alpha} p(\mathbf{F}^{[l]}, \alpha) p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s) \leq p(\mathbf{F}, \alpha) p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) - p(\mathbf{F}, \alpha)\frac{1}{|\mathrm{SL}(\mathbf{F})|}.$$

We multiply this by $\mathrm{SL}(\mathbf{F})$ and divide it by $p(\mathbf{F}, \alpha)$ (which is trivially positive) and get equivalently

$$T := \sum_{l \in \alpha} \frac{p(\mathbf{F}^{[l]}, \alpha)}{p(\mathbf{F}, \alpha)} p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s) \leq |\mathrm{SL}(\mathbf{F})| p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) - 1. \tag{1}$$

It remains to show (1). We bound the left-hand side T. First we split it into two sums:

$$T = \sum_{l \in \alpha} p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s) + \sum_{l \in \alpha} \frac{p(\mathbf{F}^{[l]}, \alpha) - p(\mathbf{F}, \alpha)}{p(\mathbf{F}, \alpha)} p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s).$$

Now we use Lemma 11 multiplied by $n(\mathbf{F})$ on the first sum. By Lemma 16 all summands of the second sum are positive, so we can use Lemma 10 to bound $p_{\text{guessed}}(\mathbf{F}^{[l]}, x, \alpha, s)$ from above by $p_{\text{guessed}}(\mathbf{F}, x, \alpha, s)$. We obtain

$$T \leq n(\mathbf{F}) p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) - 1 + \sum_{l \in \alpha} \frac{p(\mathbf{F}^{[l]}, \alpha) - p(\mathbf{F}, \alpha)}{p(\mathbf{F}, \alpha)} p_{\text{guessed}}(\mathbf{F}, x, \alpha, s).$$

Observation 13 tells us that $\sum_{l \in \alpha} p(\mathbf{F}^{[l]}, \alpha) = |\mathrm{SL}(\mathbf{F})| p(\mathbf{F}, \alpha)$, and so

$$\sum_{l \in \alpha} \frac{p(\mathbf{F}^{[l]}, \alpha) - p(\mathbf{F}, \alpha)}{p(\mathbf{F}, \alpha)} = |\mathrm{SL}(\mathbf{F})| - n(\mathbf{F}).$$

Therefore

$$T \leq n(\mathbf{F}) p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) - 1 + (|\mathrm{SL}(\mathbf{F})| - n(\mathbf{F})) p_{\text{guessed}}(\mathbf{F}, x, \alpha, s),$$

which is equal to $|\mathrm{SL}(\mathbf{F})| p_{\text{guessed}}(\mathbf{F}, x, \alpha, s) - 1$ and hence (1) holds. □

Theorem 17 can now be easily proved:

Proof of Theorem 17. We need to show that $\mathbf{F}$ is s-implication free and l chosen u.a.r from $\mathrm{SL}(\mathbf{F})$, we have

$$\mathbf{E}_l[c(\mathbf{F}^{[l]})] \leq c(\mathbf{F}) - n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} - n_{\mathrm{F}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|}.$$

Using the definition of the cost we obtain

$$\mathbf{E}_l[c(\mathbf{F}^{[l]})] = \mathbf{E}_l \left[\sum_{x \in V(\mathbf{F}^{[l]})} c(\mathbf{F}^{[l]}, x) \right] = \mathbf{E}_l \left[\sum_{x \in V(\mathbf{F})} c(\mathbf{F}^{[l]}, x) \right].$$

Then linearity of expectation gives

$$\mathbf{E}_l[c(\mathbf{F}^{[l]})] = \sum_{x \in V(\mathbf{F})} \mathbf{E}_l[c(\mathbf{F}^{[l]}, x)].$$

Now we can plug in Lemmas 18 and 19 to get

$$\mathbf{E}_l[c(\mathbf{F}^{[l]})] \leq \sum_{x \in V(\mathbf{F})} c(\mathbf{F}, x) - n_{\mathrm{N}}(\mathbf{F}) \frac{2S}{|\mathrm{SL}(\mathbf{F})|} - n_{\mathrm{F}}(\mathbf{F}) \frac{1}{|\mathrm{SL}(\mathbf{F})|}.$$

Using the definition of $c(\mathbf{F})$ gives the statement. □

4 Conclusion

We have shown an analysis of a slightly adapted PPSZ algorithm that gives the same bound for general k-SAT as for Unique k-SAT. For $k \geq 5$, this was already known, but our analysis might be considered more intuitive. For $k = 3$ and for $k = 4$ this gives improved running time bounds; for $k = 3$ the bound significantly improves from $O(1.32065^n)$ to $O(1.30704^n)$. The fastest known randomized algorithm for 3-SAT is now again rather simple compared with the algorithm proposed in [2]. It is noteworthy that this is the first algorithm for 3-SAT that is faster than, but independent of Schöning's algorithm [10]. The best known bounds for Unique k-SAT and k-SAT match now, but it is still an open question if this holds in general, as conjectured by Calabro et al. [1].

For *deterministic* algorithms, the picture is a bit different. Recently Schöning's algorithm has been fully derandomized by Moser and Scheder [6] yielding a deterministic algorithm for 3-SAT running in time $O(1.33334^n)$. This has been improved very recently by Makino, Tamaki, and Yamamoto [5] to $O(1.3303^n)$. For Unique k-SAT, PPSZ has been fully derandomized by Rolf in 2005 [8], giving a deterministic algorithm for Unique 3-SAT running in time $O(1.30704^n)$, as the randomized version. Our new approach to generalize from Unique k-SAT to k-SAT in PPSZ might be used to derandomize PPSZ for general k-SAT.

We have adapted PPSZ slightly by immediately using s-implied literals. In the original PPSZ, s-implied variables are good because they behave like non-frozen variables in the sense that restricting to them preserves satisfiability. However, while non-frozen variables still have an expected cost reduction of $\frac{2S}{|\mathrm{SL}(\mathbf{F})|} \approx \frac{0.773}{|\mathrm{SL}(\mathbf{F})|}$, the cost reduction of s-implied variables is 0, as they are guessed with probability 0 and hence already have cost 0. Our approach needs cost reduction at least $\frac{2-\log(e)}{\mathrm{SL}(\mathbf{F})|} \approx \frac{0.557}{|\mathrm{SL}(\mathbf{F})|}$. It might be interesting to check if our approach can be improved to overcome this problem and accommodate the original PPSZ algorithm.

Acknowledgements

I am very grateful to Heidi Gebauer, Dominik Scheder, and Emo Welzl for checking my ideas. Special thanks go to Robin Moser for continuous assistance in realizing this paper.

References

[1] C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi. The complexity of Unique k-SAT: an isolation lemma for k-CNFs. *J. Comput. System Sci.*, 74(3):386–393, 2008.

[2] T. Hertli, R. A. Moser, and D. Scheder. Improving PPSZ for 3-SAT using critical variables. In *Proc. of STACS 2011*, pages 237–248, 2011.

[3] K. Iwama, K. Seto, T. Takai, and S. Tamaki. Improved randomized algorithms for 3-SAT. In *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin / Heidelberg, 2010.

[4] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 328–329 (electronic), New York, 2004. ACM.

[5] K. Makino, S. Tamaki, and M. Yamamoto. Derandomizing HSSW algorithm for 3-SAT. *CoRR*, abs/1102.3766, 2011.

[6] R. A. Moser and D. Scheder. A full derandomization of Schoening's k-SAT algorithm. *CoRR*, abs/1008.4067, 2010.

[7] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364 (electronic), 2005.

[8] D. Rolf. Derandomization of PPSZ for unique-k-SAT. In *Theory and applications of satisfiability testing*, volume 3569 of *Lecture Notes in Comput. Sci.*, pages 216–225. Springer, Berlin, 2005.

[9] D. Rolf. Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:111–122, 2006.

[10] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (New York, 1999)*, pages 410–414. IEEE Computer Soc., Los Alamitos, CA, 1999.

[11] E. Welzl. Boolean satisfiability – combinatorics and algorithms (lecture notes), 2005. `http://www.inf.ethz.ch/~emo/SmallPieces/SAT.ps`.