Faster k-SAT Algorithms using Biased-PPSZ

Thomas Dueholm Hansen
University of Copenhagen
Copenhagen, Denmark
thomasdueholm@gmail.com

Haim Kaplan
Tel Aviv University
Tel Aviv, Israel
haimk@tau.ac.il

Or Zamir
Tel Aviv University
Tel Aviv, Israel
orzamir@mail.tau.ac.il

Uri Zwick
Tel Aviv University
Tel Aviv, Israel
zwick@tau.ac.il

ABSTRACT

The PPSZ algorithm, due to Paturi, Pudlak, Saks and Zane, is currently the fastest known algorithm for the k-SAT problem, for every $k > 3$. For 3-SAT, a tiny improvement over PPSZ was obtained by Hertli. We introduce a *biased* version of the PPSZ algorithm using which we obtain an improvement over PPSZ for every $k \geq 3$. For $k = 3$ we also improve on Herli's result and get a much more noticeable improvement over PPSZ, though still relatively small. In particular, for Unique 3-SAT, we improve the current bound from 1.308^n to 1.307^n.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**.

KEYWORDS

satisfiability, randomized algorithm

ACM Reference Format:
Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. 2019. Faster k-SAT Algorithms using Biased-PPSZ. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC '19), June 23–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3313276.3316359

1 INTRODUCTION

Satisfiability of Boolean formulas (usually known as SAT), is one of the core problems of Computer Science. Given a Boolean formula, the task is to decide whether there is an assignment of Boolean values, 0 (*false*) or 1 (*true*), to the variables of the formula under which the formula evaluates to 1 (*true*). The Boolean formula is commonly given in Conjunctive Normal Form (CNF), i.e., as a conjunction of disjunctions of literals. Each disjunction is called a *clause*. A *literal* is a variable or its negation. A formula in which each clause contains at most k literals is a k-CNF formula. The problem of deciding whether a k-CNF formula is satisfiable is called k-SAT.

STOC '19, June 23–26, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6705-9/19/06…$15.00
https://doi.org/10.1145/3313276.3316359

Cook [2] and Karp [7] have shown that 3-SAT is NP-complete. In contrast, 2-SAT can be solved in linear time. Many problems have been shown to be NP-complete using reductions from 3-SAT, which is thus viewed as one of the canonical NP-complete problems.

As the size of a k-CNF formula over n variables is at most $O(n^k)$, the running time of the trivial algorithm which enumerates over all possible assignments is $O^*(2^n)$, where $O^*(f(n)) = O(f(n) \cdot n^\ell)$, for some ℓ. The best known algorithms for solving k-SAT still have exponential running times in n. Let $c_k \in [1, 2]$ be the smallest constant for which k-SAT can be solved in $(c_k + o(1))^n$ time. Much effort was put into obtaining improved upper bounds on c_k, especially for 3-SAT, which has become a benchmark problem for exponential time algorithms.

A famous conjecture, called the *Exponential Time Hypothesis* (ETH, see [6]), is that 3-SAT cannot be solved in sub-exponential time, i.e., that $c_3 > 1$. A stronger conjecture, known as the *Strong Exponential Time Hypothesis* (SETH), which is also popular, claims essentially that $\lim_{k\to\infty} c_k = 2$. Both conjectures are yet to be proved or disproved.

The first non-trivial upper bound on c_k, for any $k \geq 3$, was obtained by Monien and Speckenmeyer [8]. They used a deterministic *branching algorithm* to show that $c_3 \leq 1.619$ and $c_k \leq 2^{1-\Theta(2^{-k})}$. Improved deterministic algorithms were then obtained, culminating with the bound $c_3 \leq 1.476$ obtained by Rodošek [13].

Further improved upper bounds were then obtained using randomized algorithms. Paturi, Pudlak and Zane [10] described an extremely simple and elegant randomized algorithm, now known as the PPZ algorithm, that established that $c_k \leq 2^{1-\frac{1}{k}}$. While not improving the bound for 3-SAT, the improvement for large values of k was enourmous. Shortly afterwards, with the help of Saks, Paturi et al. [9] obtained an improved version of the PPZ algorithm, now known as the PPSZ algorithm, that showed that $c_3 \leq 1.364$ and $c_k \leq 2^{1-(1-o(1))\frac{\pi^2}{6}\frac{1}{k}}$. They also showed that their 3-SAT algorithm runs in 1.308^n time, if the formula has a *unique* satisfying assignment (this restricted problem is called Unique 3-SAT).

Schöning [16] presented a very simple randomized algorithm, based on a simple *random walk*, running in time $(4/3)^n$ for 3-SAT, slightly faster than the bound obtained by PPSZ without relying on the uniqueness assumption. Hertli [3] extended the analysis of PPSZ to show that the bound obtained by PPSZ under the uniqueness assumption also holds for the general case, i.e., $c_3 \leq 1.308$. Hertli [4] also showed that the PPSZ exponent for Unique 3-SAT

can be improved by about 10^{-24}, i.e., by a *tiny* bit. Hertli's improvement uses an algorithm by Wahlström [17] that is faster than PPSZ on *short* formulas, i.e., formulas with a relatively small number of clauses. Qin and Watanabe [12] push the ideas of Herli a tiny bit further and reduce the exponent by about 10^{-19}. Hertli's [4] improvement over PPSZ works *only* for $k = 3$. Hertli mentions explicitly the problem of "breaking the PPSZ barrier" for $k > 3$.

Scheder and Steinberger [15] simplified Hertli's analysis [3] of the non-unique case, and also showed that small improvements over the PPSZ algorithm for the unique case imply small(er) improvements also for the non-unique case.

For Unique k-SAT, PPSZ was derandomized by Rolf [14]. For general k-SAT no such result is known but the best known deterministic algorithms are still based on derandomizations of the algorithms mentioned above. All the algorithms presented in this paper can be easily derandomized.

1.1 Our Contribution - A Biased Version of PPSZ

We propose a *biased* version of the PPSZ algorithm using which noticeably better bounds can be obtained for k-SAT, for every $k \geq 3$. Analyzing the new algorithm is challenging and we are only making the first steps in this direction. However, the non-optimal analysis that we currently have already allows us to obtain a non-infinitesimal improvement for Unique 3-SAT, and the first improvement over PPSZ for every $k > 3$, answering Herli's [4] open problem. For Unique 3-SAT we reduce the running time from 1.308^n to 1.307^n.[1] We believe that our current analysis only scratches the surface and that our ideas can lead to much larger improvements.

At a high level, the PPSZ algorithm can be described as follows. Go over the variables of the input formula in random order. For each variable try to 'infer' its value from the values of the variables already set. If this does not succeed, *guess* the value of the variable. If a satisfying assignment is not found, repeat. The original PPSZ algorithm [9] uses *bounded resolution* to try to infer values of variables. Hertli [3] noticed that the current analysis of PPSZ also works when the bounded resolution is replaced by weaker *bounded implication*, i.e., an implication by a small subset of the clauses. Scheder and Steinberger [15] refer to the method used to infer values of variables as the *heuristic* used. In any case, if the heuristic fails to infer the value of a variable, the PPSZ algorithm guesses the value of this variable by flipping an *unbiased* coin.

Introducing bias: Even if the heuristic failed to determine the value of the variable, there might be cases in which one of the values is *more likely* than the other value. This variable should then be guessed with a *bias*. If the bias is correct, this increases the success probability of the algorithm. We show that exploitable biases *must occur*, in significant amounts, in every input formula.

The remaining question then is how to identify situations in which bias should be used. As we are aiming for an exponential time algorithm, we can afford to spend sub-exponential time on *enumerating* where the bias is, and what its magnitude.

The biased-PPSZ algorithm that we propose is a *meta-algorithm*, parameterized by a partition of the variables into *types*. Types may be defined in a fairly general manner. The only requirement is that there are not too many of them, and that when a variable is reached during a run of the algorithm, its type can be determined in sub-exponential time. (The type of a variable usually depends on the random permutation chosen.) The algorithm enumerates, up to some precision, the bias to be used for guessing variables of each type, if they have to be guessed.

To obtain our results, we use a concrete instantiation of the above algorithm with a fairly simple collection of types. To simplify the analysis of the algorithm we currently need to use a preliminary step of choosing a maximal collection $\mathcal{D}$ of *disjoint clauses*. A maximal set of disjoint clauses, and bias, were used by Hofmeister et al. [5], in a different context, to obtain a smarter initialization for Schöning's random walk algorithm [16].

Variables that do not appear in any clause of $\mathcal{D}$ belong to a default type. Variables of this type are not guessed with a bias. The type of a variable x that does appear in a clause $C \in \mathcal{D}$ depends on the values of the literals in C that are already known when x is inspected, and on the (approximate) position of x in the permutation. The set $\mathcal{D}$ of disjoint clauses serves as a *scaffolding* that simplifies the analysis of the algorithm by removing many dependencies. We believe that further improved results can be obtained without using such a scaffolding, but the analysis becomes more challenging.

1.2 Our Concrete Results

We use concrete instantiations of the biased-PPSZ algorithm to obtain three concrete results:

- An improved algorithm for k-SAT for every $k \geq 3$. For $k > 3$, this is the first improvement over PPSZ, answering an open problem of Hertli [4]. The aim here is just to show that an improvement can be obtained for every $k \geq 3$, demonstrating the power of our techniques. The improvement we obtain is for Unique k-SAT. By [15], this implies some improvement also for k-SAT.
- An 1.307^n algorithm for Unique 3-SAT. The aim here is to show that noticeable improvements can be obtained even using our current analysis, which is probably far from being optimal. This improves over the 1.308^n running time of the PPSZ algorithm.
- An 1.305^n algorithm for Unique NAE-3-SAT. (In NAE-3-SAT, each clause is required to contain a literal that evaluates to 0, and a literal that evaluates to 1. Equivalently, NAE-3-SAT is a sub-problem of 3-SAT in which clauses appear in pairs. For every clause $x \vee y \vee z$ there is also a clause $\bar{x} \vee \bar{y} \vee \bar{z}$.) As far as we know, no improved algorithm was known for NAE-3-SAT before. NAE-3-SAT provides a simple "play ground" for explaining how bias arises, how it can be exploited, and what are the difficulties encountered in the analysis of algorithms that use bias.

1.3 Comparison with Herli's Improvement

Hertli [4] improved on PPSZ for 3-SAT, but not for k-SAT, for $k > 3$. His improvement over PPSZ is tiny (10^{-24}). Hertli's improvement is obtained using a collection of nice ideas. The analysis of PPSZ relies crucially on *critical clauses*. The uniqueness assumption implies that each variable has at least one critical clause. Herti [4] observes that

[1] With a slight "tongue in cheek". The base of the exponent of PPSZ is $1.30703\ldots$. Our current base is $1.30699\ldots$.

if many variables have more than one critical clause, then the PPSZ algorithm, without any change, is already faster. He then shows that a formula in which essentially all variables have only one critical clause is either *short*, i.e., has a small number of clauses, or has some other properties that can be exploited. To deal with short formulae, Hertli uses an algorithm of Wahlström [17]. Hertli [4] also uses bias in some settings, but in a way different than ours.

While Hertli [4] shows that the PPSZ algorithm, or at least its current analysis, is not optimal for $k = 3$, he does so in a fairly ad-hoc manner, and his result does not seem to extend to $k > 3$. It is also not clear how to use his ideas to obtain substantially improved algorithms. (Qin and Watanabe [12] tried to push Herli's ideas to the limit and only obtained a 10^{-19} improvement.)

We view the introduction of the biased-PPSZ algorithm as a new *conceptual* idea that could lead to much more significant improvements. Our current analysis of the biased-PPSZ algorithm relies, among other things, on strengthening some of the observations that are also used by Hertli [4].

1.4 Organization of the Paper

The rest of the paper is organized as follows. In Section 2 we review the PPZ and PPSZ algorithms and introduce our biased-PPSZ algorithm. The PPZ and PPSZ algorithms are presented in Sections 2.1 and 2.2. The generic version of the biased-PPSZ algorithm is presented in Section 2.3 and the concrete version analyzed in this paper is presented in Section 2.4. In Section 3 we briefly review the analysis of the PPSZ algorithm. In Section 4 we show how the high level analysis of PPSZ can be extended into an analysis of biased-PPSZ, for any system of types. The challenge is to show that the concrete system of types suggested in Section 2.4 does indeed yield improved results. In Section 5 we present a simple analysis of the biased-PPSZ algorithm for Unique NAE-3-SAT. This provides a simple setting in which our ideas can be explained without too many complications. In Section 6, we consider cases in which the analysis of PPSZ can be improved. In Section 7 we show how the simple analysis for NAE-3-SAT can be extended to a simple analysis for Unique k-SAT, for every $k \geq 3$, obtaining the first improvement over PPSZ for $k > 3$. We conclude in Section 8 with some remarks and open problems.

The tighter analysis for NAE-3-SAT giving a bound of 1.305^n for Unique NAE-3-SAT, and the 1.307^n bound for Unique 3-SAT are deferred to the full version of the paper.

2 THE PPZ, PPSZ AND BIASED-PPSZ ALGORITHMS

In this section we describe the PPZ [10] and PPSZ [9] algorithms. Our description of PPSZ follows that of Hertli [3]. We then describe our biased variant of PPSZ, first as a generic meta algorithm, and then as a concrete instantiation of it that we analyze in this paper.

2.1 The PPZ Algorithm

The simple idea behind PPZ and PPSZ is the following. Choose a random permutation of the variables. Go over the variables one by one in the random order chosen. If the value that should be assigned to a variable to satisfy the formula can be *easily deduced* from the previously assigned values, assuming that these values are correct, then assign that value to the variable. Otherwise, simply *guess* the value of the variable, with equal probabilities to 0 and 1, and independently of all previous guesses.

The crux of the matter is what is meant by *easily deduced*. PPZ uses the following simple idea. Under the assumption that there is a unique satisfying assignment α, each variable x has a *critical clause* C_x, i.e., a clause in which the literal of x is the only literal assigned the value 1 by α. (If there is no such critical clause, then the assignment α' obtained by flipping the value of x is also a satisfying assignment.) If the at most $k-1$ other variables in C_x appear before x in the permutation, then the value that should be assigned to x is easily deduced just by looking at C_x and we say that x is *forced*. Otherwise, we say that x is *guessed*. As the permutation chosen is random, the probability that x is forced is at least $\frac{1}{k}$. (It may be larger than $\frac{1}{k}$ as a variable may have more than one critical clause, or a critical clause of length less than k.) Let $G = G(\pi)$ be the number of guessed variables. The probability that the algorithm succeeds in finding the satisfying assignment if π was chosen is $2^{-G(\pi)}$. Using Jensen's inequality we get that the probability that it succeeds with a random permutation is $\mathbb{E}_\pi[2^{-G(\pi)}] \geq 2^{-\mathbb{E}_\pi[G(\pi)]} \geq 2^{-(1-\frac{1}{k})n}$. By repeating this process $2^{(1-\frac{1}{k})n}$ times, we get a constant probability of success. For $k = 3$, the expected running time, on a satisfiable instance is at most $2^{2n/3} < 1.588^n$.

2.2 The PPSZ Algorithm

The PPSZ algorithm takes this idea further. For x to be forced, the other variables in C_x do not have to appear before x in the permutation. It is enough, for example, that they are forced by the values already assigned. In [9] the authors use *bounded resolution* to try to deduce the correct value of x, given the already chosen values. Hertli [3] uses a weaker approach that seems to yield the same result. It simply enumerates over all collections of $D = D(n)$ clauses, where D is a parameter, and checks whether there is such a collection such that in all its satisfying assignments, x gets the same value. If such a collection is found, then x is assigned the corresponding value. Otherwise, x is guessed, with equal probabilities. (If both values of x are implied, the iteration can be aborted.) The parameter $D = D(n)$ is chosen such that the test takes sub-exponential time.

We let $P_D(x)$ be a procedure that implements the above process, i.e., checks whether the value of x is implied by some collection of D clauses. We assume that $P_D(x)$ returns either 0 or 1, if this is an implied value for x, or ? if the value of x is not implied.

Thus, an iteration of the PPSZ algorithm is:

- Pick a random permutation π of the variables.
- For each variable x, according to the order in π:
  - If $P_D(x) \neq ?$, assign the value $P_D(x)$ to x.
  - Otherwise, assign to x a random value, with equal probabilities.

We do enough iterations of the algorithm until we either find a satisfying assignment or choose to declare that no satisfying assignment exists. The challenge, then, is to lower bound the probability that such an iteration finds a satisfying assignment. The parameter $D = D(n)$ is chosen such that each iteration takes sub-exponential time.

2.3 The Biased-PPSZ Algorithm

PPSZ tries to deduce the value of a variable from all values set so far. If this fails, it 'gives up' and guesses the value of this variable using an unbiased coin. This seems a bit wasteful. Even if the value of a variable cannot be logically determined, there might be a way to infer what is *more likely* to be the value of this variable, and guess with a bias towards this value.

In an attempt to implement this idea, we partition the variables to be guessed into disjoint *types*. We let $\mathcal{T}$ be the set of types. We require that $|\mathcal{T}| = o(n)$. We assume that if we reach a variable in a run of PPSZ and need to guess its value, then we can determine its type in sub-exponential time.

The goal is to construct a set of types $\mathcal{T}$, that may depend on the input formula, such that the values of guessed variables of each type are *not* distributed uniformly. For example, if we somehow know that 60% of the guessed variables in some type $T \in \mathcal{T}$ should be set to 1, we should clearly guess the value of such variables with a biased coin, having a probability 0.6 for 1.

At first sight it may not be clear why types that exhibit such bias should exist. We show below, however, that fairly simple types that do exhibit some bias could be defined. We also believe that our current ideas only scratch the surface.

A type is useful even if we do not know the bias it induces, as we can *enumerate* (or *guess*) its bias, up to some precision. More specifically, we choose some value $e(n) = o(n)$, and try all biases of the form $i/e(n)$, for $i = 1, \ldots, e(n) - 1$. (We never guess with a bias of 0 or 1.) An iteration of the *generic* biased-PPSZ algorithm is thus:

- Pick a random order π for the variables.
- For every choice of values $\beta_T \in \frac{1}{e(n)}\{1, 2, \ldots, e(n) - 1\}$ for $T \in \mathcal{T}$, do:
  - Go over the variables according to π, and for each variable:
    - If $P_D(x) \neq ?$, assign the value $P_D(x)$ to x.
    - Otherwise:
      - Identify the type T of x.
      - Guess the value of x with probability β_T for 1.

We choose $e(n)$ as a monotonically increasing function such that $|\mathcal{T}| \log e(n) \in o(n)$ and $e(n) \in \omega(1)$. Thus, the enumeration takes sub-exponential time. Note that for any choice of $\mathcal{T}$, the proposed algorithm is at least as good as PPSZ, as if we let $\beta_T = \frac{1}{2}$, for every $T \in \mathcal{T}$, we simply run regular PPSZ.

2.4 Biased-PPSZ with Types Based on a Maximal Set of Disjoint Clauses

We next describe a concrete instantiation of the generic algorithm for k-SAT. The algorithm starts by constructing a maximal set $\mathcal{D}$ of *disjoint clauses* taken from the input formula. (Clauses are disjoint if they do not share variables.) Such a maximal set can be easily constructed greedily. (As we pointed out, we believe that using such a set of disjoint clauses is *not* the optimal thing to do. But, the set of disjoint clauses serves as a *scaffolding* that simplifies the analysis of the algorithm by removing many of the dependencies involved.)

We note that if $\mathcal{D}$ is small enough, we can get an improved algorithm by enumerating over the values of all the variables appearing in $\mathcal{D}$. We are then left with a $(k-1)$-SAT formula that can be solved much more efficiently than a k-SAT formula. For $k = 3$ we are left with a 2-SAT formula that can be solved in polynomial time. We may thus assume that $\mathcal{D}$ is relatively large, so a significant fraction of the variables of the formula appear in the clauses of $\mathcal{D}$.

The idea of using a maximal set of disjoint clauses appears, in a different context, in Hofmeister et al. [5] where it is used, along with bias, to obtain a smarter initialization for Schöning's random walk algorithm [16].

As a 'warm up', we begin by discussing the NAE-3-SAT subproblem of 3-SAT in which identifying types that yield significant bias is especially easy. A 3-SAT formula φ is a NAE-3-SAT formula if the clauses in φ appear in pairs, for every clause $x \vee y \vee z$ there is also a clause $\bar{x} \vee \bar{y} \vee \bar{z}$. The *weight* of a clause is defined to be the number of literals in it that evaluate to 1 under the unique satisfying assignment. (To get a unique satisfying assignment for a NAE-3-SAT formula, we assume that the first variable is set to 0.) In a NAE-3-SAT formula, all clauses are of weight 1 or 2. (In a 3-SAT formula, clauses can also be of weight 3.) Thus, if the variables of a clause appear in random order, then with a probability of $\frac{2}{3}$, the value of the second literal is *different* from the value of the first literal. Thus, if we have to guess the value of the second variable in a clause, when the value of the first variable is already known, it seems to be a good idea to guess it with a bias towards the opposite value of the first literal.

This suggests, as a first attempt, a very simple type system $\mathcal{T} = \{T_\perp, T_0, T_1\}$, where $x \in T_a$, for $a = 0, 1$, if x appears in a disjoint clause $C_x \in \mathcal{D}$ and when x has to be guessed, the value of exactly one of the other literals in C_x is known, and has the value $\bar{a}$, if x is unnegated in C_x, or a, if x is negated. The type $T_\perp$ contains all the remaining variables. Note that the type of a variable depends on the permutation π. Type T_a, for $a = 0, 1$, contains variables that are to be guessed with a bias towards a, while type $T_\perp$ contains variables that are to be guessed without bias. (We will shortly refine this type system.)

A problem we face while trying to rigorously analyze the proposed algorithm for NAE-3-SAT is that while it is true that the second literal to appear in a clause has a probability of $\frac{2}{3}$ of having a different value than the first, it is not clear that this also holds *conditioned* on the second literal being guessed. Can the *adversary* find a formula in which this conditional probability is much smaller than $\frac{2}{3}$, possibly even $\frac{1}{2}$, in which case we loose all the advantage we are hoping to gain?

With a slight refinement of these types, the only way the adversary can do this is by increasing substantially the forcing probabilities, to levels beyond that promised by the analysis of PPSZ, in which case we gain even without using biased guesses.

The refinement needed takes into account the position, or the *time*, of a variable x in the permutation π. As we shall see in Section 3, it is convenient to assume that the permutation π used by the algorithm is obtained by drawing for each variable x an independent uniformly distributed variable $\sigma(x)$ from $[0, 1]$. The permutation π is then obtained by sorting these times.

A variable appearing at time p, where p is small, has a small probability of being forced. (The formal proof of this intuitive statement is given in Section 3.) Thus, if such a variable is the second to appear in one of the disjoint clauses, and is guessed, we still expect the probability that its literal has value different from the first literal

to be close to $\frac{2}{3}$. As p grows, this probability might become smaller. We thus refine the types by taking into account the time in which each variable arrives. For some parameter $b(n)$, we let $T_{a,i}$ be the set of variables x belonging to T_a such that $\sigma(x) \in \left[\frac{i}{b(n)}, \frac{i+1}{b(n)}\right)$, for $i = 0, 1, \ldots, b(n)-1$. The type $T_\perp$ remains unchanged. Variables of $T_{a,i}$ will still be guessed with a bias towards a, but the bias will decrease with i. In Section 5 we show, by focusing on small values of p, that this gives some improvement for NAE-3-SAT. A refined analysis for NAE-3-SAT, that gives a much larger improvement for NAE-3-SAT is given in the full version of the paper.

With the insight gained from NAE-3-SAT, we now propose a concrete type system for a k-SAT formulas. We again rely on a maximal collection $\mathcal{D}$ of disjoint clauses of the input formula φ. The type of variable x of φ that appears in a clause $C_x \in \mathcal{D}$ depends on three factors:

(1) *prefix* – the sequence of values already known in C_x.
(2) *structure* – the set of clauses in φ containing only the variables appearing in C_x.
(3) *time* – A discretized version of $\sigma(x)$, i.e., $\lfloor \sigma(x) b(n) \rfloor$.

In NAE-3-SAT we only considered prefixes of size 1. In k-SAT we consider prefixes of all sizes, i.e., $0, 1, \ldots, k-1$. We next explain the concept of *structure*. We consider, for concreteness, the case $k = 3$. We may assume, without loss of generality, that all the variables in the clauses of $\mathcal{D}$ appear unnegated. Thus, the clause C_x is of the form $x \vee y \vee z$ for some y, z. It is possible that φ contains other clauses on the same three variables x, y, z. (In NAE-3-SAT, this is always the case.) If, for example, φ contains the clauses $x \vee y \vee z$, $x \vee \bar{y} \vee z$ and $\bar{x} \vee \bar{y} \vee z$, and no other clauses on x, y, z, we say that the structure of x is $\{000, 010, 110\}$. (As we are allowed to swap y and z, the structures $\{000, 010, 110\}$ and $\{000, 001, 101\}$ are considered the same.) As an illustration, note that if the structure of x is $\{000, 011, 101, 110\}$, then the four clauses on x, y, z imply the linear equation $x \oplus y \oplus z = 1$. If x has this structure, and has a prefix of length 2, then x is forced. It is, of course, possible that all variables appearing in disjoint clauses would have the trivial structure $\{000\}$.

We still have a separate type $T_\perp$ for all the variables that do not appear in the disjoint clauses.

The concrete algorithm, with a proper choice of the parameter $0 < \gamma < 1$, is then:

(1) Greedily build a maximal set $\mathcal{D}$ of disjoint k-clauses of the input formula φ.
(2) If $|\mathcal{D}| \leq \gamma n$
  (a) Enumerate over the $(2^k - 1)^{|\mathcal{D}|}$ possible assignments to the variables appearing in $\mathcal{D}$.
  (b) For each assignment (recursively) solve the $(k-1)$-SAT instance left after assigning the values to the variables of $\mathcal{D}$.
(3) If $|\mathcal{D}| > \gamma n$
  (a) Define $\mathcal{T}$ based on the prefix, structure and time of each variable, as defined above.
  (b) Run the biased-PPSZ algorithm with $\mathcal{T}$.

We analyze this algorithm in Section 5.

3 ANALYSIS OF PPSZ

In this section we review the analysis of the PPSZ algorithm. The results in this section are not new, but essential for understanding the analyses of various versions of the biased-PPSZ algorithm. Our presentation is influenced by that of Hertli [3]. For concreteness, we focus on the case $k = 3$. The analysis extends easily to $k > 3$. (See [3, 9] for the details.)

A *critical clause* of a variable x is a clause in which the literal of x is the only literal in the clause that gets the value 1 under the unique satisfying assignment. The following lemma generalizes the claim that each variable has a critical clause.

Lemma 3.1. *Let φ be a k-SAT formula with a unique satisfying assignment α. For a subset $S \subset V$ of variables, let α_S be the assignment defined by $\alpha_S(x) = \overline{\alpha(x)}$, for every $x \in S$, and $\alpha_S(x) = \alpha(x)$, for every $x \in V \setminus S$. (In other words, α_S is obtained from α by flipping the value of the variables in S.) For every $S \neq \emptyset$, there exists a clause in the formula in which all the variables in the clause whose literals are assigned the value 1 by α are from S, and all variables in the clause whose literals are assigned the value 0 by α are from $V \setminus S$. In particular, this clause must contain a literal of a variable from S.*

Proof. As $\alpha_S \neq \alpha$, the formula is not satisfied by the assignment α_S. Thus, there must exist a clause which is satisfied by α but not by α_S. As all literals in this clause are assigned the value 0 by α_S, it follows that all variables whose literals are assigned 1 by α belong to S, and all variables whose literals are assigned 0 by α must belong to $V \setminus S$. □

To continue with the analysis, we think of the random permutation π as the permutation defined by picking an independent uniform random *time* $\sigma(x) \in [0, 1]$ for each variable x, and sorting the variables according to these values.

Conditioning on $\sigma(x) = p$ (for some $p \in [0, 1]$), our current goal is to lower bound the probability that x is forced in an iteration of the PPSZ algorithm. (A variable is *forced* if we can deduce its value from some subset of D clauses.) We denote such a lower bound that holds for every variable by $q(p) = q^{(3)}(p)$. (We are considering here 3-SAT formulas. As mentioned, the analysis can be extended to $k \geq 3$. Thus, a lower bound on the probability of each variable being forced, without conditioning on $\sigma(x)$, is $\int_0^1 q(p)dp$. As each variable x has a critical clause C_x, it follows that $q(p) \geq p^2$, as p^2 is the probability that the other variables in C_x appear before x in the permutation. (This is basically the analysis of the PPZ algorithm which essentially uses $D = 1$.) However, it is clear that this bound is far from being tight when $D > 1$. Intuitively, we would want to claim that $q(p) \geq (p + (1-p)q(p))^2$, as we 'interpret' $p + (1-p)q(p)$ as the probability that a given variable other than x in C_x either appears before x, or is forced at time p. (This is intuition, not a rigorous claim.) This would imply that $q(p) \geq (\frac{p}{1-p})^2$ for $0 \leq p \leq \frac{1}{2}$, and $q(p) = 1$ for $p > \frac{1}{2}$. The problem with this naïve argument is the dependency between the events. However, as shown in [3, 9], this claim is true, up to lower order terms, as we shall also show below. As our main purpose here is to present the ideas behind the PPSZ algorithm and its analysis in a simple manner, we ignore these lower order terms. (For the completely rigorous analysis, see

[3, 9].) Thus, the probability of each variable being forced is at least

$$\mu := \int_0^1 q(p)dp = \int_0^{\frac{1}{2}} \left(\frac{p}{1-p}\right)^2 dp + \frac{1}{2} \;=\; 2 - 2\ln 2 \;\approx\; 0.6137\ .$$

This gives us the following theorem,

Theorem 3.2 ([9]). *The success probability of an iteration of PPSZ on a uniquely satisfied 3-SAT formula is at least* $2^{-(1-\mu+o(1))n} \geq 1.308^{-n}$.

Proof. Denote by $G = G(\pi)$ the number of variables that have to be guessed in an iteration of PPSZ with a permutation π, assuming that the value of each guessed variable is guessed correctly according to the satisfying assignment. The success probability of such iteration is exactly $2^{-G(\pi)}$. Using Jensen's inequality and the linearity of expectation, as we did before, we have $\mathbb{E}_\pi[2^{-G(\pi)}] \geq 2^{\mathbb{E}_\pi[-G(\pi)]} \geq 2^{-(1-\mu+o(1))n}$. □

To justify the claim $q(p) \geq (p + (1-p)q(p))^2$ (up to lower order terms), PPSZ [9] define for each variable x in φ a tree T_x called a *critical clause tree* of x.

Definition 3.3 (Critical clause trees [9])**.** A *critical clause tree* T_x of a variable x in φ is defined as a binary tree generated by the following process. Every node in the tree has both a variable and a clause of φ corresponding to it.

- Begin with a root node r corresponding to the variable x. (Its corresponding clause is yet to be assigned.)
- As long as there exists a leaf v of the tree without an assigned clause, do:
  - Let S be the set of variables on the path from r to v, including r and v.
  - Let C be a clause of φ *not* satisfied by the assignment α_S.
  - Assign C to v and extend the tree by adding to v children corresponding to the variables of $V(C) \setminus S$, if such variables exist, where $V(C)$ are the variables appearing in C.

It is clear from the above definition that two nodes labeled by the same variable cannot appear on the same path from the root of the tree. Note also that if $V(C) \subseteq S$, where C is the clause chosen at v, then v remains a leaf of T_x. It follows that T_x is always finite.

Theorem 3.4 ([9]). *Let φ be a 3-SAT formula with a unique satisfying assignment. Let x be a variable and let T_x be a critical clause tree for x. Let S be a subset of variables and let R be the set of clauses corresponding to nodes of T_x reachable from the root after all nodes labeled by variables of S have been removed from the tree. If the values of all the variables of S are fixed to their values in the unique assignment, then the correct value of x is implied by the clauses of R.*

Proof. Let α be the unique satisfying assignment of φ. Let φ_R denote the subformula of φ composed only of the clauses of R. We prove that φ_R has no satisfying assignment α' in which $\alpha'(x) \neq \alpha(x)$, while $\alpha'(y) = \alpha(y)$, for every $y \in S$. Assume, by contradiction that there is such α'.

Suppose that $r = v_0, v_1, \ldots, v_k$ is a path in T_x and let x_i be the variable corresponding to v_i. We claim that if $\alpha'(x_i) \neq \alpha(x_i)$, for $i = 0, 1, \ldots, k$, then either α' is not a satisfying assignment of φ_R, or v_k must have a child v_{k+1} in T_x, labeled by a variable x_{k+1}, such that $\alpha'(x_{k+1}) \neq \alpha(x_{k+1})$, i.e., the path can be extended. Indeed, let C be the clause corresponding to v_k. By the definition of critical clause trees, C is not satisfied by the assignment α'' in which $\alpha''(x_i) \neq \alpha(x_i)$, for $i = 0, 1, \ldots, k$, while $\alpha''(y) = \alpha(y)$ for all other variables. Note that α' and α'' agree on $x_0, x_1, \ldots, x_k$. If all the variables appearing in C are from $\{x_0, x_1, \ldots, x_k\}$, we get that α' does not satisfy C, and hence is not a satisfying assignment of φ_R. If C contains other variables, then to satisfy C, the assignment α' must differ from α'', and hence from α, in at least one of these variables. The node corresponding to this variable is the required child v_{k+1} of v_k.

The root itself is such a path. We iteratively extend this path until we either conclude that α' does not satisfy φ_R, or until we reach a node whose corresponding variable is from S, in which case we again reach a contradiction, as we assumed that $\alpha'(y) = \alpha(y)$ for every $y \in S$. □

We now present a probabilistic model in which the relation $q(p) \geq (p + (1-p)q(p))^2$ is rigourous.

Lemma 3.5. *Let T be an infinite binary tree in which each node is labeled by a variable. Each variable is assigned an independent and uniformly distributed random number from* $[0, 1]$. *For a certain $0 < p < 1$, all nodes,* other than the root, *whose variables are assigned values less than p, and all their descendents, are removed from the tree. Let $q(p)$ be the probability that the remaining tree is* finite. *Then, $q(p) \geq (p + (1-p)q(p))^2$, and as a consequence $q(p) \geq \left(\frac{p}{1-p}\right)^2$, for $0 \leq p \leq \frac{1}{2}$, and $q(p) = 1$, for $\frac{1}{2} \leq p \leq 1$. These inequalities are tight if and only if all the variables labeling the nodes are distinct.*

Proof. Assume, at first, that all the variables are distinct. We then have $q(p) = (p + (1-p)q(p))^2$, as the remaining tree is finite if and only if each child of the root is either cut off, with probability p, or is not cut off, but has finite remaining subtree, which happens with probability $(1-p)q(p)$. Note that all events are now completely independent.

We next remove the assumption that all the variables are distinct. Let I_v be the event that the variable of a node v is assigned a value less than p, and that v is thus removed from the tree. The event of having a finite remaining tree is *monotone* in the events $\{I_v\}$. Thus, by the FKG inequality (see, e.g., Alon and Spencer [1]), dependencies among the events $\{I_v\}$ only increase the probability of having a finite remaining tree. (It is not difficult to check that the special case of the FKG inequality that we are using holds in our case even though we are considering an infinite number of events.) □

Let $q_D(p)$ be the probability that the tree has a *D-frontier* in the probabilistic model of Lemma 3.5, i.e., that the remaining tree is of size at most D. By definition, $q_D(p) \to q(p)$, as D tends to ∞. In other words, $|q_D(p) - q(p)|$ converges to zero as $D \to \infty$. We note that $|q_D(p) - q(p)|$ is defined independently of any specific instance formula or input size. In particular, if $D = D(n) = \omega_n(1)$ is any increasing function of n, then $q_D(p) = q(p) - o_n(1)$.

We now get back to critical clause trees.

Theorem 3.6. *Let T_x be a critical clause tree of some variable x. Then, the probability that T_x has a D-frontier at time p is at least $q_D(p)$. Thus, if $D = D(n) = \omega_n(1)$, this probability is $q(p) - o_n(1)$.*

Proof. The probabilistic model considered here is very similar to the one considered in Lemma 3.5. The only differences are that T_x is a finite tree and that some of its nodes may only have a single child. This clearly *increases* the probability of having a D-frontier. □

Corollary 3.7. *The probability of each variable to be forced is at least $\mu - o_n(1)$, where $\mu := 2 - 2\ln 2 \approx 0.6137056$, assuming $D = \omega_n(1)$.*

Proof. A variable is forced if and only if when we reach it in the permutation there is already a frontier of size $\leq D$ in its critical clause tree. This probability tends to $\mu = \int_0^1 q(p)dp = 2 - 2\ln 2$ as $D \to \infty$. □

As shown in [9], similar analysis works for general k-SAT. There, instead of binary trees we consider $(k-1)$-ary critical clause trees. We denote by $q^{(k)}(p)$ the probability of such an infinite $(k-1)$-ary tree to have a finite frontier at time p, in a similar way to the definition of $q(p) = q^{(3)}(p)$. We let $S_k = 1 - \int_0^1 q^{(k)}(p)dp$. The running time of PPSZ for k-SAT formulas is $2^{(S_k+o(1))n}$ by a very similar analysis. In the full version of the paper we show that $p^{k-1} \leq q^{(k)}(p) \leq \left(\frac{p}{1-p}\right)^{k-1}$, which is a rough estimation that suffices for our purpose of providing a slightly improved bound for k-SAT.

4 HIGH LEVEL ANALYSIS OF BIASED-PPSZ

The analysis of PPSZ extends naturally to the generic biased-PPSZ algorithm as follows. Let β_T be the *bias* used for type T. For every permutation π of the variables and each type $T \in \mathcal{T}$, let $G_T(\pi)$ be the number of guessed variables of type T. Also, let $G_T^{(0)}(\pi)$ and $G_T^{(1)}(\pi)$ be the number of guessed variables of type T that are 0 and 1, respectively, in the unique satisfying assignment. Thus, $G_T(\pi) = G_T^{(0)}(\pi) + G_T^{(1)}(\pi)$. The probability that a single run of the algorithm finds the unique satisfying assignment is:

$$\begin{aligned}
&\mathbb{E}_\pi\left[\prod_{T\in\mathcal{T}} \beta_T^{G_T^{(1)}(\pi)}(1-\beta_T)^{G_T^{(0)}(\pi)}\right] \\
&\quad\geq \prod_{T\in\mathcal{T}} \beta_T^{\mathbb{E}_\pi[G_T^{(1)}(\pi)]}(1-\beta_T)^{\mathbb{E}_\pi[G_T^{(0)}(\pi)]} \\
&\quad= 2^{\left(\sum_{T\in\mathcal{T}} \mathbb{E}_\pi[G_T^{(1)}(\pi)]\log(\beta_T)+\mathbb{E}_\pi[G_T^{(0)}(\pi)]\log(1-\beta_T)\right)},
\end{aligned}$$

where we used Jensen's inequality, and the log's are base 2.

Define $G_T = \mathbb{E}_\pi[G_T(\pi)]$ and $\beta_T^* = \mathbb{E}_\pi[G_T^{(1)}(\pi)]/G_T$. We can then rewrite the lower bound for the success probability as:

$$2^{\left(\sum_{T\in\mathcal{T}} G_T\cdot\left(\beta_T^*\log(\beta_T)+(1-\beta_T^*)\log(1-\beta_T)\right)\right)}$$

Note that the function $f(x) = \beta_T^*\log(x) + (1-\beta_T^*)\log(1-x)$ is maximized at $x = \beta_T^*$. The best choice of β_T for the algorithm is therefore $\beta_T = \beta_T^*$. Since β_T^* is unknown, we approximate it by enumerating over many values of β_T. This ensures that for some choice of β_T, where $\beta_T \neq 0, 1$, we have $|\beta_T^* - \beta_T| \leq \frac{1}{e(n)}$ and thus:

$$\begin{aligned}
&\beta_T^*\log(\beta_T) + (1-\beta_T^*)\log(1-\beta_T) \\
&\quad\geq \beta_T^*\log(\beta_T^*) + (1-\beta_T^*)\log(1-\beta_T^*) - O(\tfrac{1}{e(n)}) \\
&\quad= -H(\beta_T^*) - O(\tfrac{1}{e(n)}),
\end{aligned}$$

where $H(x) = -x\log x - (1-x)\log(1-x)$ is the binary entropy function. We thus get that:

$$\begin{aligned}
&\sum_{T\in\mathcal{T}} G_T\cdot(\beta_T^*\log(\beta_T) + (1-\beta_T^*)\log(1-\beta_T)) \\
&\geq -\sum_{T\in\mathcal{T}} G_T\cdot\left(H(\beta_T^*) + O(\tfrac{1}{e(n)})\right) \\
&\geq -\sum_{T\in\mathcal{T}} G_T\cdot H(\beta_T^*) - o(n),
\end{aligned}$$

where the second inequality uses the fact that $e(n) \in \omega(1)$ and $\sum_{T\in\mathcal{T}} G_T \leq n$. These are summarized by the following lemma.

Lemma 4.1. *For some choice of $\beta_T \in \frac{1}{e(n)}\{1, 2, \ldots, e(n)-1\}$ for all $T \in \mathcal{T}$, each run of the generic algorithm finds the unique satisfying assignment with probability at least $2^{-\left(\sum_{T\in\mathcal{T}} G_T\cdot H(\beta_T^*)\right)-o(n)}$.*

The PPSZ algorithm is essentially the biased-PPSZ algorithm with only one type T, where $\beta_T = \frac{1}{2}$, i.e., no bias is used. In this case $G = G_T = \mathbb{E}_\pi[G_T(\pi)]$ is exactly the expected number of guessed variables.

To show that the biased-PPSZ improves on the standard PPSZ algorithm we need to define a type system for which we can show a gap between β_T^* and $1/2$ for a significant fraction of the guessed variables.

5 SIMPLE ANALYSIS FOR NAE-3-SAT

Consider the following simple and naïve algorithm. Construct a maximal set $\mathcal{D}$ of disjoint clauses. For simplicity assume that all variables in the clauses of $\mathcal{D}$ are unnegated. We let $V(\mathcal{D})$ be the set of variables that appear in the clauses of $\mathcal{D}$. Also, recall that we assume that the formula has a unique satisfying assignment denoted by α.

Let t be a threshold to be chosen later. The algorithm is identical to PPSZ, except that if a variable $x \in V(\mathcal{D})$, i.e., a variable (whose positive literal is) contained in a clause $C_x \in \mathcal{D}$, appears before time t, is the *second* variable to appear from C_x, and it is not forced, then assign probability $2/3$ to the value that gives it a value different from the value of the variable already known in C_x, and probability $1/3$ to the other value.

For a permutation π, let $G(\pi)$ be the total number of variables guessed in π, with or without bias, let $G^+(\pi)$ be the number of variables guessed with a correct $\frac{2}{3}:\frac{1}{3}$ bias, and let $G^-(\pi)$ be the number of variables guessed with an incorrect such bias. For brevity, we sometimes omit π and write G instead of $G(\pi)$. The success probability of the algorithm is

$$\begin{aligned}
&\mathbb{E}_\pi\left[\left(\frac{2}{3}\right)^{G^+}\left(\frac{1}{3}\right)^{G^-}\left(\frac{1}{2}\right)^{G-G^+-G^-}\right] = \mathbb{E}_\pi\left[\left(\frac{4}{3}\right)^{G^+}\left(\frac{2}{3}\right)^{G^-}\left(\frac{1}{2}\right)^{G}\right] \\
&= \mathbb{E}_\pi\left[\left(\frac{1}{2}\right)^{G-(\lg\frac{4}{3})G^+ + (\lg\frac{3}{2})G^-}\right] \geq \left(\frac{1}{2}\right)^{\mathbb{E}_\pi\left[G-(\lg\frac{4}{3})G^+ + (\lg\frac{3}{2})G^-\right]},
\end{aligned}$$

where the last inequality follows from Jensen's inequality.

For a fixed variable $x \in V(\mathcal{D})$, let $g_k(p)$, for $k = 1, 2, 3$, be the probability that x is the k-th variable to arrive in its clause, and that it is *guessed*, given that $\sigma(x) = p$. Note that this is the *actual* probability for the specific variable x. For brevity we omit x from the notation, but these values are usually different for different variables. Also let $g(p)$ be the probability that x is guessed, given that $\sigma(x) = p$. Clearly,

$$g(p) \;=\; g_1(p) + g_2(p) + g_3(p) \;.$$

We know that $g(p) \le \bar{g}(p)$, where $\bar{g}(p) = 1 - \left(\frac{p}{1-p}\right)^2$. For small values of p we have $\bar{g}(p) \simeq 1 - p^2 - \ldots$. Furthermore, let $g_2^+(p)$ be the probability that x is the second variable to appear in its clause, that the value of x *differs* from the value of the variable in C_x that already appeared, and that x is guessed, given that $\sigma(x) = p$. Let $g_2^-(p)$ be the analogous probability when the value of x is required to be *equal* to the variable of C_x that already appeared. Note that $g_2^+(p)$ and $g_2^-(p)$ are the probabilities that x is guessed with the correct, respectively incorrect, bias, given that $\sigma(x) = p$. Clearly $g_2(p) = g_2^+(p) + g_2^-(p)$.

The contribution of a variable $x \in V(\mathcal{D})$ to $\mathbb{E}_\pi\left[G - (\lg\frac{4}{3})G^+ + (\lg\frac{3}{2})G^-\right]$ is

$$\int_0^t \left(g(p) - (\lg\tfrac{4}{3})g_2^+(p) + (\lg\tfrac{3}{2})g_2^-(p)\right)dp + \int_t^1 g(p)dp \;.$$

We concentrate on the first integral, as the second one is identical to the integral appearing in the standard analysis of PPSZ. We let

$$\gamma^x(p) = g(p) - (\lg\tfrac{4}{3})g_2^+(p) + (\lg\tfrac{3}{2})g_2^-(p) \;.$$

Our goal is to bound $\gamma^x(p) = \gamma(p)$, showing that it is smaller than $g(p)$, for $0 \le p \le t$, thus getting an improvement over PPSZ. Recall that all these values are defined for a specific variable $x \in V(\mathcal{D})$.

Let $x \in V(\mathcal{D})$ and let y, z be the other variables appearing in C_x. We distinguish between two cases. The first is that $\alpha(x) \ne \alpha(y) = \alpha(z)$, i.e., x is the *distinct* variable in its clause. The second is that $\alpha(x) = \alpha(y) \ne \alpha(z)$, i.e., x is the *non-distinct* variable in its clause. (We assume, without loss of generality that $\sigma(x) \ne \sigma(z)$.

Assume at first that $\alpha(x) \ne \alpha(y) = \alpha(z)$. In this case $g_2^+(p) = g_2(p)$ and $g_2^-(p) = 0$. Our goal is to upper bound

$$\gamma_1(p) = g(p) - (\lg\tfrac{4}{3})g_2(p) \;,$$

where we let $\gamma_1(p) = \gamma^x(p)$ where x is the distinct variable in its clause. The "danger" is that $g_2(p)$ may be tiny, or even 0, in which case we do not get a chance to guess with the correct bias, and do not seem to win over PPSZ. We show that this can only happen if $g(p) < \bar{g}(p)$, in which case we also win.

Recall that $g(p) = g_1(p) + g_2(p) + g_3(p)$. Note that $g_1(p) \le (1-p)^2$ and $g_3(p) \le p^2$, as these are the probabilities that x appears first and third, respectively, given that $\sigma(x) = p$, without requiring that x is guessed. Thus,

$$g_2(p) \;\ge\; g(p) - (1-p)^2 - p^2 \;=\; g(p) - (1 - 2p(1-p)) \;,$$

and hence

$$\begin{aligned}\gamma_1(p) \;&\le\; g(p) - (\lg\tfrac{4}{3})(g(p) - (1 - 2p(1-p)))\\ &\le\; \bar{g}(p) - (\lg\tfrac{4}{3})(\bar{g}(p) - (1 - 2p(1-p)))\end{aligned}$$

The last inequality follows from the fact that the coefficient of $g(p)$ in the first expression is $1 - \lg\frac{4}{3} = \lg\frac{3}{2} > 0$, i.e., the expression is increasing with $g(p)$, and from the fact that $g(p) \le \bar{g}(p)$. We immediately see that $\gamma_1(p) \le \bar{g}(p)$ when $1 - 2p(1-p) \le \bar{g}(p)$, which holds for $0 \le p < 0.41025$.

Consider now the case $\alpha(x) = \alpha(y) \ne \alpha(z)$. Our goal is to bound

$$\gamma_2(p) \;=\; g(p) - (\lg\tfrac{4}{3})g_2^+(p) + (\lg\tfrac{3}{2})g_2^-(p) \;,$$

where $\gamma_2(p) = \gamma^x(p)$ when x is the non-distinct variable in its clause. Recall that $g(p) = g_1(p) + g_2^+(p) + g_2^-(p) + g_3(p)$. As $g_1(p) \le (1-p)^2$, $g_2^-(p) \le p(1-p)$ and $g_3(p) \le p^2$, we get that

$$g_2^+(p) \;\ge\; g(p) - (1-p)^2 - p(1-p) - p^2 \;=\; g(p) - (1 - p(1-p)) \;.$$

Hence,

$$\begin{aligned}\gamma_2(p) \;&=\; g(p) - (\lg\tfrac{4}{3})g_2^+(p) + (\lg\tfrac{3}{2})g_2^-(p)\\ &\le\; g(p) - (\lg\tfrac{4}{3})(g(p) - (1 - p(1-p))) + (\lg\tfrac{3}{2})p(1-p) \;.\\ &\le\; \bar{g}(p) - (\lg\tfrac{4}{3})(\bar{g}(p) - (1 - p(1-p))) + (\lg\tfrac{3}{2})p(1-p) \;,\end{aligned}$$

where the last inequality follows as in the previous case. Unfortunarely, this time it is not true that $\gamma_2(p) \le \bar{g}(p)$.

Let x, y, z be the variables appearing in a clause of $\mathcal{D}$, and assume that x is the distinct variable. Let $\gamma_{x,y,z}(p) = \frac{1}{3}(\gamma^x(p) + \gamma_y(p) + \gamma_z(p))$. By the two bounds above, we get that

$$\begin{aligned}\gamma_{x,y,z}(p) \;&\le\; \tfrac{1}{3}\gamma_1(p) + \tfrac{2}{3}\gamma_2(p)\\ &=\; \tfrac{1}{3}\left(\bar{g}(p) - (\lg\tfrac{4}{3})(\bar{g}(p) - (1 - 2p(1-p)))\right)\\ &+\; \tfrac{2}{3}\left(\bar{g}(p) - (\lg\tfrac{4}{3})(\bar{g}(p) - (1 - p(1-p))) + (\lg\tfrac{3}{2})p(1-p)\right)\\ &=\; \bar{g}(p) - (\lg\tfrac{4}{3})\left(\bar{g}(p) - \left(1 - \tfrac{4}{3}\left(1 - \tfrac{1}{2}\tfrac{\lg\frac{3}{2}}{\lg\frac{4}{3}}\right)p(1-p)\right)\right)\\ &=\; \bar{g}(p) - (\lg\tfrac{4}{3})\left(\bar{g}(p) - (1 - \eta\, p(1-p))\right) \;,\end{aligned}$$

where $\eta = \frac{4}{3}\left(1 - \frac{1}{2}\frac{\lg\frac{3}{2}}{\lg\frac{4}{3}}\right) = 0.393719\ldots$ The important thing is that $\eta > 0$, so that $\bar{g}(p) \ge 1 - \eta p(1-p)$ for small enough values of p. We let t be the largest p for which the inequality holds. It turns out that $t = 0.2009\ldots$, i.e., the inequality holds even for values of p that are not that small.

The gain over PPSZ for the average of x, y, z, for a given p, is:

$$\delta \;=\; (\lg\tfrac{4}{3})\int_0^t (\bar{g}(p) - (1 - \eta p(1-p)))dp \;\simeq\; 0.00129151$$

Let $3|\mathcal{D}| = cn$, i.e., c is the fraction of variables that appear in $\mathcal{D}$. The gain in the exponent over PPSZ is then $c\delta$. If $6^{cn/3} < 1.305^n$, then we can enumerate over all the variables of $\mathcal{D}$ and solve the remaining 2-SAT instance in polynomial time. Thus, we may assume that $c \ge 0.445712$, and then $c\delta \ge 0.00057$.

6 CASES IN WHICH PPSZ BEHAVES BETTER

We begin by describing a few cases in which the PPSZ algorithm, without any changes, behaves better than promised by the standard analysis. These observations are used in the analysis of the biased-PPSZ algorithm presented in Section 7. We again focus on the case $k = 3$, but all the results extend easily to $k > 3$. Omitted proofs appear in the full verison of the paper.

The first scenario in which we prove that PPSZ behaves better is when there are many variables with more than one critical

clause. This is fairly intuitive, as since a variable with more than one critical clause is more likely to be forced. The easiest way of seeing it is the following. Suppose that x has two critical clauses, one containing literals of variables y_1, z_1 and the other literals of variables y_2, z_2. Note that y_1, z_1, y_2, z_2 are not necessarily distinct, but $|\{y_1, z_1, y_2, z_2\}| \geq 3$. The probability of x to be forced at time p is at least

$$\begin{aligned} & \Pr(y_1,z_1 \text{ appear before } x \text{ or } y_2,z_2 \text{ appear before } x) \\ = \ & \Pr(y_1,z_1 \text{ appear before } x) + \Pr(y_2,z_2 \text{ appear before } x) \\ - \ & \Pr(y_1,z_1,y_2,z_2 \text{ appear before } x) \\ \geq \ & 2p^2 - p^3 \end{aligned}$$

which is larger than $q(p) = (\frac{p}{1-p})^2$ when p is small enough. A similar observation was made in [4]. The following lemm gives a stronger bound.

Lemma 6.1. *Let x be a variable with more than one critical clause, then $Pr(x \text{ is forced} \mid \sigma(x) = p) \geq \frac{2-p}{1-p} \cdot p^2 - p^3$. In particular, the probability that x is forced is at least $\mu + \varepsilon_1$ for $\varepsilon_1 > 0.0035$.*

The proof of Lemma 6.1, which appears in the full version of the paper, looks more deeply into the structure of the critical clause tree of x and its relation to the second critical clause of x.

A second case we study is a case in which variables have critical clauses that are *related* in the sense that y appears in a critical clause of x and vice versa. In that case, we show that we also gain in the forcing probability of at least one of the variables x and y. Intuitively, either x has a second critical clause and then Lemma 6.1 is applicable, or every critical clause tree of x contains y as a child of the root, as it appears in its unique critical clause. Assuming that y does not have a second critical clause as well, we show that the child of the root corresponding to y in any critical clause tree of x must have at most one child.

Lemma 6.2. *Let x, y, z_1, z_2 be variables such that x has a critical clause containing literals of y and z_1, and y has a critical clause containing literals of x and z_2. Then, either x or y has a probability of at least $\mu + \varepsilon_2$ to be forced, for $\varepsilon_2 > 0.0035$.*

Lemma 6.2 is used in the rest of the paper in order to restrict the dependance between the events of different variables being forced.

7 ANALYSIS FOR k-SAT

The goal of this section is to give a simple proof that the algorithm of Section 2.4 gives a small improvement for Unique k-SAT, for every $k \geq 3$. Using the results of [15], this implies an improvement over PPSZ for k-SAT, for every $k \geq 3$, also without the uniqueness assumption. The focus is on making the proof as simple as possible, so no attempt is made to optimize constants. Our goal, is therefore, to prove the following theorem. Recall that $S_k = 1 - \int_0^1 q^{(k)}(p)dp$ is an upper on the probability that a variable is guessed by the PPSZ algorithm.

Theorem 7.1. *For every $k \geq 3$ there exists $\varepsilon_k > 0$ such that biased-PPSZ solves Unique k-SAT instances in time $2^{(S_k - \varepsilon_k)n}$.*

In Section 5 we presented a simple proof that biased-PPSZ is faster than PPSZ on NAE-3-SAT instances. Let us review the properties of NAE-3-SAT formulas that made the analysis there work. For $i, j \in \{0, 1\}$, denote by p_{ij} the probability that if we uniformly draw a clause C from $\mathcal{D}$, then the values of the literals of the first two variables of C to appear, in the order of arrival, are i and j. Let $K_t := (3t^2 - 2t^3)|\mathcal{D}|$. In expectation we have $p_{ij}K_t$ clauses from $\mathcal{D}$ such that at least two of their variables appear before time t and have ordered literal values i and j. The number of forced variables that appear before time t, according to the standard analysis, is $o(K_t)$ as $t \to 0$. Thus, for a small enough t, we expect to have about $p_{ij}K_t$ clauses from $\mathcal{D}$ such that at least two of their variables appear before time t, have ordered literal values i and j, *and these two variables are guessed*. If this is the case, we should guess the values of the first two variables in clauses of $\mathcal{D}$ that contain at least two variables appearing before time t, according to the distribution $(p_{00}, p_{01}, p_{10}, p_{11})$. The probability of being correct in all these guesses is about

$$p_{00}^{p_{00}K_t} p_{01}^{p_{01}K_t} p_{10}^{p_{10}K_t} p_{11}^{p_{11}K_t} = 2^{-H(p_{00},p_{01},p_{10},p_{11})K_t},$$

where H stands for the entropy of the distribution. If $H(p_{00}, p_{01}, p_{10}, p_{11}) < 2$, the same type of analysis yields an improvement over PPSZ also for such a 3-SAT formula.

For general k, we can use the same intuitive argument: the probability that a variable that appears before time t is forced, according to the standard analysis, is $\Theta(t^{k-1})$. Thus, the probability that a variable appears before time t and is forced is $\Theta(t^k)$. On the other hand, the probability that at least $k-1$ out of the k variables of a clause appear before time t is $\Theta(t^{k-1})$. Thus, if we expect a bias on the first $k-1$ literals of clauses from $\mathcal{D}$, then for a small enough t we get an improvement using biased guesses, as the number of forced variables is too small to interfere.

More precisely, for a given input formula φ and a set of disjoint clauses $\mathcal{D}$, let $a_{\mathbf{v}}$, for every $\mathbf{v} \in \{0,1\}^{k-1}$, be the probability that if we choose a random clause from $\mathcal{D}$ and randomly permute the order of its literals, then the values assigned to the first $k-1$ literals of the clause by the unique satisfying assignment of φ are those of $\mathbf{v}$. We are now ready to prove the following theorem.

Theorem 7.2. *For every k and $\varepsilon > 0$, there exists $\delta > 0$ such that if $H(\{a_{\mathbf{v}}\}_{\mathbf{v}\in\{0,1\}^{k-1}}) \leq (k-1) - \varepsilon$, then biased-PPSZ runs in time $2^{S_k n - \delta|\mathcal{D}|}$.*

Proof. Let $t = t(\varepsilon, k)$ be a parameter to be chosen later. We analyze the success probability of the following algorithm (given a set $\mathcal{D}$ of disjoint clauses):

- Enumerate the values of $\{a_{\mathbf{v}}\}_{\mathbf{v}\in\{0,1\}^{k-1}}$, up to some $o_n(1)$ error. (This takes subexponential time.)
  - Pick a random time $\sigma(x)$ for each variable x. Use the times to define a random permutation π.
  - For each clause $C \in \mathcal{D}$:
    - Let $x_1, x_2, \ldots, x_k$ the variables in C such that $\sigma(x_1) < \sigma(x_2) < \ldots < \sigma(x_k)$.
    - If $\sigma(x_{k-1}) < t$, guess the values of $x_1, x_2, \ldots, x_{k-1}$ according to the distribution $\{a_{\mathbf{v}}\}_{\mathbf{v}\in\{0,1\}^{k-1}}$.
  - Run a standard iteration of PPSZ, from the beginning of π. Do not guess variables whose values are already assigned.

This algorithm is a restricted version of the algorithm presented in Section 2.4. Thus, any bound we prove for this algorithm also holds for the one of Section 2.4.

For a given σ, let $K_t^{\mathbf{v}}(\sigma)$ be the number of clauses in $\mathcal{D}$ such that at least $k-1$ of their variables appear before time t with ordered literal values corresponding to $\mathbf{v}$. Let $V^{<t}(\sigma)$ be the number of variables covered by $\mathcal{D}$ that appear before time t. Let $G^{>t}(\sigma)$ be the number of variables covered by $\mathcal{D}$ that appear after time t and are guessed. Let $R(\sigma)$ be the number of variables not covered by $\mathcal{D}$ that are guessed. We let $K_t^{\mathbf{v}}$, $V^{<t}$, $G^{>t}$ and R be the corresponding random variables when σ is random.

The success probability of the algorithm, if σ is chosen, is at least

$$\left(\prod_{\mathbf{v}\in\{0,1\}^{k-1}} a_{\mathbf{v}}^{K_t^{\mathbf{v}}(\sigma)}\right)\left(\frac{1}{2}\right)^{V^{<t}(\sigma)-(k-1)(\sum_{\mathbf{v}} K_t^{\mathbf{v}}(\sigma))+G^{>t}(\sigma)+R(\sigma)}$$

since the probability of fixing correctly the values of each $(k-1)$-tuple of variables counted in $K_t^{\mathbf{v}}(\sigma)$ is $a_{\mathbf{v}}$, the probability of fixing correctly the value of any *other* variable (not necessarily guessed) appearing before time t (and covered by $\mathcal{D}$) is at least $\frac{1}{2}$, and the probability of fixing correctly any other guessed variable is $\frac{1}{2}$.

The success probability is lower bounded by the expectation of this expression over σ. Using Jensen's inequality we have

$$\mathbb{E}\left[\left(\prod_{\mathbf{v}} a_{\mathbf{v}}^{K_t^{\mathbf{v}}}\right)\left(\frac{1}{2}\right)^{V^{<t}-(k-1)(\sum_{\mathbf{v}} K_t^{\mathbf{v}})+G^{>t}+R}\right]$$

$$\geq 2^{-\left(\mathbb{E}[R]+\mathbb{E}[G^{>t}]+\mathbb{E}[V^{<t}]-(k-1)\mathbb{E}[\sum_{\mathbf{v}} K_t^{\mathbf{v}}]-\sum_{\mathbf{v}}\log a_{\mathbf{v}}\mathbb{E}[K_t^{\mathbf{v}}]\right)}.$$

Since $\mathbb{E}[K_t^{\mathbf{v}}] = a_{\mathbf{v}}K_t$, where

$$K_t = \left(kt^{k-1}(1-t)+t^k\right)|\mathcal{D}| \;=\; \left(kt^{k-1}-(k-1)t^k\right)|\mathcal{D}|\,,$$

the above can be written as

$$2^{-\left(\mathbb{E}[R]+\mathbb{E}[G^{>t}]+\mathbb{E}[V^{<t}]-\left((k-1)-H(\{a_{\mathbf{v}}\}_{\mathbf{v}\in\{0,1\}^{k-1}})\right)K_t\right)}$$

$$\geq 2^{-\left(\mathbb{E}[R]+\mathbb{E}[G^{>t}]+\mathbb{E}[V^{<t}]-\varepsilon K_t\right)}.$$

We also have $\mathbb{E}[R] \leq (n-k|\mathcal{D}|)\cdot S_k$, $\mathbb{E}[V^{<t}] = kt|\mathcal{D}|$ and $\mathbb{E}[G^{>t}] \leq k|\mathcal{D}|\cdot\int_t^1(1-q^{(k)}(p))dp$. Substituting these in the above expression gives

$$2^{-\left((n-k|\mathcal{D}|)S_k+k|\mathcal{D}|\cdot\int_t^1(1-q^{(k)}(p))dp+kt|\mathcal{D}|-\varepsilon K_t\right)}$$

As

$$\int_t^1(1-q^{(k)}(p))dp \;=\; \int_0^1(1-q^{(k)}(p))dp-\int_0^t(1-q^{(k)}(p))dp$$

$$\leq S_k - t + tq^{(k)}(t) \;\leq\; S_k - t + t\left(\tfrac{t}{1-t}\right)^{k-1}$$

we have

$$(n-k|\mathcal{D}|)S_k + k|\mathcal{D}|\cdot\int_t^1(1-q^{(k)}(p))dp + kt|\mathcal{D}| - \varepsilon K_t$$

$$\leq\; (n-k|\mathcal{D}|)S_k + k|\mathcal{D}|\cdot\left(S_k - t + t\left(\frac{t}{1-t}\right)^{k-1}\right)$$

$$+\; kt|\mathcal{D}| - \varepsilon\left(kt^{k-1}-(k-1)t^k\right)|\mathcal{D}|$$

$$=\; S_k n - |\mathcal{D}|\left(\varepsilon\left(kt^{k-1}-(k-1)t^k\right) - kt\left(\frac{t}{1-t}\right)^{k-1}\right).$$

For a small enough choice of t, we have that

$$\delta := \varepsilon\left(kt^{k-1}-(k-1)t^k\right) - kt\left(\frac{t}{1-t}\right)^{k-1} > 0\,.$$

□

We are thus left to deal with the case that $H(\{a_{\mathbf{v}}\}_{\mathbf{v}\in\{0,1\}^{k-1}})$ is very close to $k-1$.

For a k-SAT formula φ with a unique satisfying assignment α, and a maximal collection $\mathcal{D}$ of disjoint clauses, let $\mathbf{w} = (w_1, w_2, \ldots, w_k)$ be the fractions of clauses in $\mathcal{D}$ of weight $1, 2, \ldots, k$, respectively. (Recall that the weight of a clause is the number of literals in it that evaluate to 1 under α.) The values $a_{\mathbf{v}}$ are, of course, functions of $\mathbf{w}$. By symmetry, $a_{\mathbf{v}_1} = a_{\mathbf{v}_2}$ for every $\mathbf{v}_1, \mathbf{v}_2$ of the same weight. Thus, we let $a_i = a_{\mathbf{v}}$, for $\mathbf{v}$ of weight i, for $i = 0, \ldots, k-1$.

Lemma 7.3. *$a_0 = \frac{1}{k}w_1$ and $a_i = \frac{1}{\binom{k}{i}}w_i + \frac{1}{\binom{k}{i+1}}w_{i+1}$, $1 \leq < k$.*

Proof. By definition a_i is the probability of obtaining a specific $(k-1)$-vector of weight i, e.g., the vector $\mathbf{v} = 1^i0^{k-1-i}$ of i 1's followed by $k-1-i$ 0's. If the clause chosen from $\mathcal{D}$ is of weight i, which happens with probability w_i, the probability of getting $\mathbf{v}$ is $1/\binom{k}{i}$. If the clause chosen from $\mathcal{D}$ is of weight $i+1$, which happens with probability w_{i+1}, the probability of getting $\mathbf{v}$ is $1/\binom{k}{i+1}$. If the clause chosen is not of weight i or $i+1$, then $\mathbf{v}$ cannot be obtained. (Note that $w_0 = 0$, as all clauses are satisfied.) □

If we let $\mathbf{a} = (a_0, a_1, \ldots, a_{k-1})$, it follows that $\mathbf{a} = M\mathbf{w}$, where

$$M \;=\; \begin{bmatrix} \frac{1}{\binom{k}{1}} & 0 & & \cdots & 0 \\ \frac{1}{\binom{k}{1}} & \frac{1}{\binom{k}{2}} & 0 & & \vdots \\ & \frac{1}{\binom{k}{2}} & \frac{1}{\binom{k}{3}} & \ddots & \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & & \frac{1}{\binom{k}{k-1}} & \frac{1}{\binom{k}{k}} \end{bmatrix}.$$

The matrix M is clearly invertible. If we let $\mathbf{a}_0 = (\frac{1}{2^{k-1}}, \frac{1}{2^{k-1}}, \ldots, \frac{1}{2^{k-1}})$, the vector of probabilities corresponding to the uniform distribution over $\{0,1\}^{k-1}$, there is a unique vector $\mathbf{w}_0$ such that $\mathbf{a}_0 = M\mathbf{w}_0$. It is easy to check that the vector $\mathbf{w}_0$ is given by:

$$(\mathbf{w}_0)_i = \begin{cases} \frac{1}{2^{k-1}}\binom{k}{i} & ,\ i \text{ is odd} \\ 0 & ,\ i \text{ is even} \end{cases}$$

Note that $H(\{a_{\mathbf{v}}\}) = k-1$ only if $\mathbf{a} = \mathbf{a}_0$. This happens only if the weight distribution over the disjoint clauses is $\mathbf{w}_0$. A remarkable property of $\mathbf{w}_0$ is that it assigns non-zero probabilities only to *odd* weights. (When $k = 3$, for example, the weight distribution is $(w_1, w_2, w_3) = (\frac{3}{4}, 0, \frac{1}{4})$, consistent with a NAE-3-SAT instance.) Thus, we immediately get

Corollary 7.4. *For every k and $\varepsilon > 0$, there exists $\varepsilon' > 0$, such that if $||\mathbf{w} - \mathbf{w}_0||_1 > \varepsilon$ then $H(\{a_{\mathbf{v}}\}) \leq (k-1) - \varepsilon'$.*

Thus, we are left to prove that we get an improvement when $\mathbf{w}$ is very close to $\mathbf{w}_0$. In this case, the value of the last variable in every disjoint clause is almost uniquely determined by the values of the first $k-1$ variables of the clause. Thus, if we need to guess the last variable of a clause we have a probability close to 1 of succeeding. However, it might be that somehow the last variable of each clause in $\mathcal{D}$ is always forced.

We next show that in certain cases, a variable appearing in a disjoint clause is either forced with a probability higher than $1-S_k$, or has a positive probability of being *last* in its clause and *not* forced.

Let $F(x)$ be the event that variable x is *forced*, and let $G(x)$ be the event that x is *guessed*. Let $C_1(x_1;x_2,\ldots,x_k)$ be the clause composed of literals of the variables $x_1,\ldots,x_k$ such that the literal of x_1 evaluates to 1 and the literals of $x_2,\ldots,x_k$ evaluate to 0 under the unique satisfying assignment of the input formula. The clause $C(x_1;x_2,\ldots,x_k)$ may or may not appear in the input formula.

Lemma 7.5. *There exists a constant $\varepsilon_3>0$ such that if the clause $C_1(x_1;x_2,\ldots,x_k)$ does* not *appear in the input formula, then*

$$\Pr[F(x_1) \text{ or } (G(x_1) \text{ and } \sigma(x_2),\ldots,\sigma(x_k)<\sigma(x_1)] \geq (1-S_k)+\varepsilon_3 .$$

Proof. Let $C_1(x_1;y_2,\ldots,y_k)$ be a critical clause of x_1 that appears in the input formula φ. (Such a clause must exist.) As this clause is different from $C_1(x_1;x_2,\ldots,x_k)$, that does not appear in the formula, we get that $|\{x_2,\ldots,x_k,y_2,\ldots,y_k\}|\geq k$. Now

$$\begin{aligned}
&\Pr[F(x_1) \mid \sigma(x_1)=p] \\
&\geq \Pr[\sigma(y_2),\ldots,\sigma(y_k)<p \mid \sigma(x_1)=p] \\
&\quad + \Pr[\sigma(x_2),\ldots,\sigma(x_k)<p \mid \sigma(x_1)=p] \\
&\quad \cdot \Pr[F(x_1) \mid \sigma(x_2),\ldots,\sigma(x_k)<p,\ \sigma(x_1)=p] \\
&\quad - \Pr[\sigma(x_2),\ldots,\sigma(x_k),\sigma(y_2),\ldots,\sigma(y_k)<p \mid \sigma(x_1)=p] \\
&\geq p^{k-1}(1+\Pr[F(x_1) \mid \sigma(x_2),\ldots,\sigma(x_k)<p,\ \sigma(x_1)=p]) - p^k
\end{aligned}$$

and

$$\begin{aligned}
&\Pr[G(x_1) \text{ and } \sigma(x_2),\ldots,\sigma(x_k)<\sigma(x_1) \mid \sigma(x_1)=p] \\
&= p^{k-1}(1-\Pr[F(x_1) \mid \sigma(x_2),\ldots,\sigma(x_k)<p,\ \sigma(x_1)=p]) .
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\Pr[F(x_1) \mid \sigma(x_1)=p] \ + \\
&\Pr[G(x_1) \text{ and } \sigma(x_2),\ldots,\sigma(x_k)<\sigma(x_1) \mid \sigma(x_1)=p] \\
&\geq 2p^{k-1}-p^k .
\end{aligned}$$

Thus,

$$\begin{aligned}
&\Pr[F(x_1) \text{ or } (G(x_1) \text{ and } \sigma(x_2),\ldots,\sigma(x_k)<\sigma(x_1)] \\
&\geq \int_0^1 \max(2p^{k-1}-p^k, q^{(k)}(p))dp \ \geq\ (1-S_k)+\varepsilon_3 .
\end{aligned}$$

for some $\varepsilon_3>0$. (Recall that $p^{k-1}\leq q^{(k)}(p)\leq\left(\frac{p}{1-p}\right)^{k-1}$ and thus $q^{(k)}(p)$ is strictly smaller than $2p^{k-1}-p^k$ for a small enough p.) □

We now have:

Theorem 7.6. *For every* small enough *$\varepsilon>0$, there exists $\delta>0$, such that if $||\mathbf{w}-\mathbf{w}_0||_1<\varepsilon$, then biased-PPSZ runs in time $2^{S_k n-\delta|\mathcal{D}|}$.*

Proof. Let C be a clause of $\mathcal{D}$. The input formula φ may or may not contain other clauses on exactly the same variables appearing in C. We are especially interested in how many of these clauses are of weight 1, i.e., critical for one of the variables appearing in C.

We consider two cases. Either, (1) for at least half of the clauses of $\mathcal{D}$ there are at least two critical clauses that use the variables of the clause. Or, (2) for at least half of the clauses of $\mathcal{D}$ there is at most one critical clause using the variables of the clause. (The bound on δ can be improved by choosing a more balanced partition into cases.)

Consider case (1) first. Note that two critical clauses on the same set of variables are related, in the sense of Lemma 6.2. (Lemma 6.2 is stated for $k=3$, but as mentioned, it can be easily extended to any $k\geq 3$.) Thus, there are at least $\frac{1}{2}|\mathcal{D}|$ variables with a forcing probability of at least $\geq(1-S_k)+\varepsilon_2$ for some $\varepsilon_2>0$. Therefore, standard PPSZ, and thus also biased-PPSZ, runs in time $2^{S_k n-\varepsilon_2\cdot\frac{1}{2}|\mathcal{D}|}$.

Consider now case (2). For at least $\frac{1}{2}|\mathcal{D}|$ of the clauses of $\mathcal{D}$ there is at most one critical clause using the variables of the clause. Thus, at least $k-1$ of the variables in each one of these $\frac{1}{2}|\mathcal{D}|$ clauses satisfies the condition of Lemma 7.5. Let X be the set of variables that satisfies this condition. Then, $|X|\geq(k-1)\cdot\frac{1}{2}|\mathcal{D}|$.

Let $x_1,x_2,\ldots,x_k$ be the variable appearing in a clause $C\in\mathcal{D}$ and suppose that $x_1\in X$. By Lemma 7.5, we get that

$$\Pr[F(x_1) \text{ or } (G(x_1) \text{ and } \sigma(x_2),\ldots,\sigma(x_k)<\sigma(x_1)] \geq (1-S_k)+\varepsilon_3 .$$

Consider a variant of PPSZ that guesses variables without bias, except for variables that are last to appear in a clause of $\mathcal{D}$. If such a variable needs to be guessed, it is given with probability β the value that would make the weight of the clause odd, and the opposite value with probability $1-\beta$, for some $\beta>\frac{1}{2}$ to be chosen later. This is again a restricted version of the algorithm of Section 2.4.

The success probability of the algorithm is $\mathbb{E}[(\frac{1}{2})^R\beta^G(1-\beta)^B]$ where G is the number of guessed variables that are last in their clause $C\in\mathcal{D}$ which is of odd weight, B is the number of guessed variables that are last in their clause $C\in\mathcal{D}$ which is of even weight, and R is the number of other guessed variable. Jensen's inequality shows that the expected success probability is $(\frac{1}{2})^{\mathbb{E}[R]}\beta^{\mathbb{E}[G]}(1-\beta)^{\mathbb{E}[B]}$. We have $\mathbb{E}[B]\leq\varepsilon|\mathcal{D}|$ as there is at most one such variable in every even weight clause of $\mathcal{D}$ and there are at most $\varepsilon|\mathcal{D}|$ such clauses. We also have by Lemma 7.5 that $\mathbb{E}[n-R]\geq(1-S_k)n+\frac{1}{2}(k-1)|\mathcal{D}|\cdot\varepsilon_2$ as every variable of X has probability at least $(1-S_k)+\varepsilon_2$ to be either forced or in one of G and B. As usual, we also have $\mathbb{E}[R+G+B]\leq S_k n$. These inequalities together give that

$$\begin{aligned}
&\left(\frac{1}{2}\right)^{\mathbb{E}[R]}\beta^{\mathbb{E}[G]}(1-\beta)^{\mathbb{E}[B]} \\
&\geq \left(\frac{1}{2}\right)^{S_k n-\frac{1}{2}(k-1)|\mathcal{D}|\cdot\varepsilon_3}\beta^{\frac{1}{2}(k-1)|\mathcal{D}|\cdot\varepsilon_3-\varepsilon|\mathcal{D}|}(1-\beta)^{\varepsilon|\mathcal{D}|} \\
&= \left(\frac{1}{2}\right)^{S_k}(2\beta)^{\frac{1}{2}(k-1)|\mathcal{D}|\cdot\varepsilon_3}\left(\frac{1-\beta}{\beta}\right)^{\varepsilon|\mathcal{D}|} .
\end{aligned}$$

For a small enough ε there exists a choice of β for which the above is $(\frac{1}{2})^{S_k n-\delta|\mathcal{D}|}$ for some $\delta>0$. □

Combining the above theorems immediately implies

Lemma 7.7. *For every k there exists $\varepsilon'_k>0$ such that Biased-PPSZ solves an instance of Unique k-SAT with $|\mathcal{D}|$ disjoint clauses in time $2^{S_k n-\varepsilon'_k|\mathcal{D}|}$.*

We are now ready to prove Theorem 7.1.

Proof. Let $\gamma=\frac{1}{n}|\mathcal{D}|$ be the normalized size of $\mathcal{D}$. Enumerating over the values of the variables appearing in $\mathcal{D}$ and solving the remaining $(k-1)$-SAT instance using PPSZ (or biased-PPSZ) takes time $(2^k-1)^{\gamma n}\cdot 2^{S_{k-1}n}$. By Lemma 7.7, biased-PPSZ solves the problem in time $2^{(S_k-\gamma\varepsilon'_k)n}$. Thus, if γ is sufficiently small, we get an improvement over PPSZ by the enumeration. If γ is larger, we

get an improvement by using biased-PPSZ. The running times of these two algorithms is the same when $\gamma = \gamma_0$, where

$$\gamma_0 = \frac{S_k - S_{k-1}}{\log(2^k - 1) + \varepsilon'_k} .$$

The running time of the algorithm, for any γ, is at most $2^{(S_k - \varepsilon_k)n}$ where

$$\varepsilon_k = \gamma_0 \varepsilon'_k = \frac{S_k - S_{k-1}}{\log(2^k - 1) + \varepsilon'_k} \varepsilon'_k > 0 .$$

□

In the full version of this paper, we repeat the above analysis for $k = 3$ while paying more attention to the exact constants and trade-offs. We also perform some other optimizations. The running time of biased-PPSZ for a specific weight vector $\mathbf{w} = (w_1, w_2, w_3)$ is efficiently *computable*. We conduct a rigorous *computer assisted* search for the worst weight vector $\mathbf{w}$ and rigorously prove an upper bound on the running time of the algorithm for any weight vector $\mathbf{w}$. We obtain the following concrete upper bound:

Theorem 7.8. *Unique* 3*-SAT can be solved in time* $O(1.307^n)$.

8 CONCLUDING REMARKS AND OPEN PROBLEMS

The main contribution of this paper is the idea of adding bias into the PPSZ algorithm. We suggested a concrete way of adding such a bias and showed that it yields slightly improved algorithms for k-SAT for every $k \geq 3$.

The numerical bound we currently have for Unique 3-SAT is 1.306995^n. We believe that this can be further improved. For Unique NAE-3-SAT we get a bound of 1.30452^n. The best bound that we can hope to achieve using the concrete algorithms suggested for NAE-3-SAT and 3-SAT is 1.30331^n. (This correspond to the simple and natural case in which the critical clause trees of variables that appear in the same clause are disjoint.) We believe that this "lower bound" can be approached, both for NAE-3-SAT and 3-SAT.

Further improvements can possibly be obtained by *not* using a set of disjoint clauses. More variables can potentially be guessed with bias, but the analysis becomes more challenging.

We hope that the improved bounds we obtained for Unique 3-SAT also apply directly to 3-SAT, using essentially the arguments of Hertli [3], avoiding the 'reduction' of Scheder and Steinberger [15] and the associated loss that comes with it.

Our bounds improve on the current bound available for the PPSZ algorithm. It is not known, however, whether these bounds are tight. The current best lower bound on the running time of the PPSZ algorithm were obtained by Pudlák et al. [11].

The best k-SAT algorithms, including the ones presented here, have running times of the form $2^{(1-\frac{\Theta(1)}{k})n}$. Obtaining bound of the form $2^{(1-\frac{\omega(1)}{k})n}$, or providing evidence that such bounds cannot be obtained, would be a major breakthrough. Shedding more light on this problem is probably more important than improving the constant in the bound for 3-SAT.

ACKNOWLEDGMENTS

T. D. Hansen was partially supported by BARC, funded by the VILLUM Foundation grant 16582. Haim Kaplan was partially supported by the Israel Science Foundation grant 1841/14 by the German-Israeli Foundation for Scientific Research and Developmentrant 1367/2017, and by the Blavatnik Research Foundation. Part of this research was carried out while Or Zamir and Uri Zwick were visiting BARC, funded by the VILLUM Foundation grant 16582. This work was carried out in partial fulfillment of the requirements for the Ph.D. degree of Or Zamir.

REFERENCES

[1] Noga Alon and Joel H. Spencer. 2016. *The probabilistic method* (4nd ed.). Wiley-Interscience.
[2] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *Proc. of 3rd STOC*. 151–158. https://doi.org/10.1145/800157.805047
[3] Timon Hertli. 2014. 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General. *SIAM J. Comput.* 43, 2 (2014), 718–729. Announced at FOCS'11.
[4] Timon Hertli. 2014. Breaking the PPSZ Barrier for Unique 3-SAT. In *Proc. of 41st ICALP I.* 600–611. https://doi.org/10.1007/978-3-662-43948-7_50
[5] Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe. 2007. Randomized Algorithms for 3-SAT. *Theory Comput. Syst.* 40, 3 (2007), 249–262. https://doi.org/10.1007/s00224-005-1275-6
[6] Russell Impagliazzo and Ramamohan Paturi. 2001. On the Complexity of k-SAT. *J. Comput. Syst. Sci.* 62, 2 (2001), 367–375. https://doi.org/10.1006/jcss.2000.1727
[7] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.
[8] Burkhard Monien and Ewald Speckenmeyer. 1985. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics* 10, 3 (1985), 287–295. https://doi.org/10.1016/0166-218X(85)90050-2
[9] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. 2005. An improved exponential-time algorithm for k-SAT. *J. ACM* 52, 3 (2005), 337–364. https://doi.org/10.1145/1066100.1066101 Announced at FOCS'98.
[10] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. 1999. Satisfiability Coding Lemma. *Chicago J. Theor. Comput. Sci.* (1999). http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html
[11] Pavel Pudlák, Dominik Scheder, and Navid Talebanfard. 2017. Tighter Hard Instances for PPSZ. In *Proc. of 44th ICALP (Leibniz International Proceedings in Informatics (LIPIcs))*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.), Vol. 80. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 85:1–85:13. https://doi.org/10.4230/LIPIcs.ICALP.2017.85
[12] Tong Qin and Osamu Watanabe. 2018. An Improvement of the Algorithm of Hertli for the Unique 3SAT Problem. In *International Workshop on Algorithms and Computation*. Springer, 93–105.
[13] Robert Rodošek. 1996. A New Approach on Solving 3-Satisfiability. In *Artificial Intelligence and Symbolic Mathematical Computation, International Conference AISMC-3, Steyr, Austria, September 23-25, 1996, Proceedings*. 197–212. https://doi.org/10.1007/3-540-61732-9_59
[14] Daniel Rolf. 2005. Derandomization of PPSZ for Unique-k-SAT. In *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*. 216–225. https://doi.org/10.1007/11499107_16
[15] Dominik Scheder and John P. Steinberger. 2017. PPSZ for General k-SAT - Making Hertli's Analysis Simpler and 3-SAT Faster. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*. 9:1–9:15. https://doi.org/10.4230/LIPIcs.CCC.2017.9
[16] Uwe Schöning. 2002. A Probabilistic Algorithm for k-SAT Based on Limited Local Search and Restart. *Algorithmica* 32, 4 (2002), 615–623. https://doi.org/10.1007/s00453-001-0094-7
[17] Magnus Wahlström. 2005. An Algorithm for the SAT Problem for Formulae of Linear Length. In *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*. 107–118. https://doi.org/10.1007/11561071_12