Exploiting Independent Subformulas: A Faster Approximation Scheme for #k-SAT

Manuel Schmitt Rolf Wanka

Department of Computer Science
University of Erlangen-Nuremberg, Germany
{manuel.schmitt, rolf.wanka}@cs.fau.de

Abstract

We present an improvement on Thurley's recent randomized approximation scheme for #k-SAT where the task is to count the number of satisfying truth assignments of a Boolean function Φ given as an n-variable k-CNF. We introduce a novel way to identify independent substructures of Φ and can therefore reduce the size of the search space considerably. Our randomized algorithm works for any k. For #3-SAT, it runs in time $O(\varepsilon^{-2} \cdot 1.51426^n)$, for #4-SAT, it runs in time $O(\varepsilon^{-2} \cdot 1.60816^n)$, with error bound ε.

Keywords: Algorithms; analysis of algorithms; randomized algorithms; #k-SAT; satisfiability.

1 Introduction

Background. The satisfiability problem (SAT) is one of the classical and central problems in algorithm theory. Its prominent role in Computer Science has even been compared [15] to the one that Drosophila (the fruit fly) has in Genetics. Given a Boolean formula Φ in conjunctive normal form (CNF) on n variables with m clauses, it has to be determined whether there is a satisfying assignment for Φ (and in this case, to determine one) or not. If every clause of Φ has length at most k, Φ is called a k-CNF and the problem is dubbed k-SAT. It is well known (for a comprehensive overview, see [3]) that k-SAT is NP-complete for any $k \geq 3$, and that it can be solved in time linear in the input length for $k = 2$ [1]. So it is generally assumed that there is no polynomial time algorithm solving k-SAT for $k \geq 3$. In particular, 3-SAT has attracted much attention because of its "borderline" status.

There is a rich history of developing both deterministic and randomized algorithms with running time $o(2^n)$ solving k-SAT. The currently fastest deterministic algorithm for 3-SAT runs in time[1] $O^*(1.3303^n)$ [11], the fastest randomized algorithm has a running time of $O^*(\log(\delta^{-1}) \cdot 1.30704^n)$ [4]. In the randomized setting, the use of δ means the following: If Φ is not satisfiable, the algorithm returns the correct answer. If Φ is satisfiable, it returns with probability $1-\delta$ a satisfying assignment. Table 1 presents all best running times currently known to solve k-SAT.

For many combinatorial problems including k-SAT, it is often not only important to determine one solution (if it exists), but also to determine the number of all different solutions. A famous

[1] In this context, the notion $O^*(.)$ is commonly used to suppress factors that are of size $2^{o(n)}$.

Table 1: Previous and new results for k-SAT and #k-SAT where the input k-CNF has n variables and m clauses. The times are given in $O^*(.)$ notation. β_k is the base-2 logarithm of the base of the running time in column "k-SAT rand". For definition of μ_k, see Sec. 2.2; ψ_k is the largest root of $1 - 2z^k + z^{k+1} = 0$; $2^{1/(2-\beta_k)} > \alpha_k$.

	k-SAT deterministic		k-SAT rand		β_k	#k-SAT exact		#k-SAT rand, prev.		#k-SAT this paper
$k=2$	$n+m$	[1]	—		—	1.2377^n	[18]	—		—
$k=3$	1.3303^n	[11]	1.30704^n	[4]	0.3864	1.6423^n	[10]	1.5366^n	[16]	1.51426^n
$k=4$	1.5^n	[12]	1.46899^n	[4]	0.5548	1.9275^n	[2, 19]	1.6155^n	[16]	1.60816^n
$k \geq 5$	$\left(\frac{2\cdot(k-1)}{k}\right)^n$	[12]	$2^{(1-\mu_k/(k-1))\cdot n}$	[14]	$1-\frac{\mu_k}{k-1}$	ψ_k^n	[19]	$2^{1/(2-\beta_k)\cdot n}$	[16]	α_k^n (Sec. 5.3)

example from statistical physics is the computation of the number of configurations in monomer-dimer systems (for an overview, see [8]). The complexity class that corresponds to these *counting problems* is #P, and #SAT, the problem to determine the number of satisfying assignments, is well-known to be #P-complete. More exactly, let #k-SAT denote the problem to determine $\#\Phi$, i. e., for input Φ being a k-CNF, the number of satisfying assignments. Then, it is known [17] that #k-SAT is #P-complete for $k \geq 2$.

Topic of this work. In the area of combinatorial counting problems, there is also the problem of approximating the wanted number. In particular, there is the task to develop so-called *randomized approximation schemes* that receive as input Φ and an arbitrarily small bound ε on the maximum admissible error and that compute with some fixed probability greater 1/2 an ε-estimate of $\#\Phi$ (for exact definitions, see Sec. 2). In a recent paper, Thurley [16] presents such a randomized approximation scheme for #k-SAT that has, for $k = 3$, running time $O^*(\varepsilon^{-2} \cdot 1.5366^n)$, and for $k = 4$, $O^*(\varepsilon^{-2} \cdot 1.6155^n)$. A detailed description of Thurley's algorithm is presented in Sec. 2. Table 1 also presents all best running times currently known to solve #k-SAT.

A different approach by Impagliazzo et al. [7] leads to a randomized Las Vegas algorithm for #k-SAT that always returns the exact solution and has expected running time $O^*(2^{(1-1/(30k))n})$. Note that for any k, Thurley's algorithm is faster than this method.

New Results. We present a randomized approximation scheme for #k-SAT that takes the input k-CNF much more into account than Thurley's algorithm. In particular, we present a method that determines a large set of maximal independent subformulas of Φ. I. e., the subformulas have no variables in common and can therefore be treated independently. As they are maximal, they convert the remaining clauses into clauses of length $k - 1$. Hence, the search space is substantially reduced. Our scheme, which works for any #k-SAT instance, has for #3-SAT running time $O(\varepsilon^{-2} \cdot 1.51426^n)$, and for #4-SAT, it works in time $O(\varepsilon^{-2} \cdot 1.60816^n)$. Note that our scheme is for all k faster than Thurley's scheme.

Organization of Paper. In the next section, we define the necessary terms, and we give a comprehensive description of Thurley's randomized approximation scheme. In Sec. 3, we present a first improvement that exploits single clauses. Generalizing this approach and building upon each other, we present further improvements based on large sets of maximal independent clauses (Sec. 4), and on large sets of maximal independent subformulas (Sec. 5).

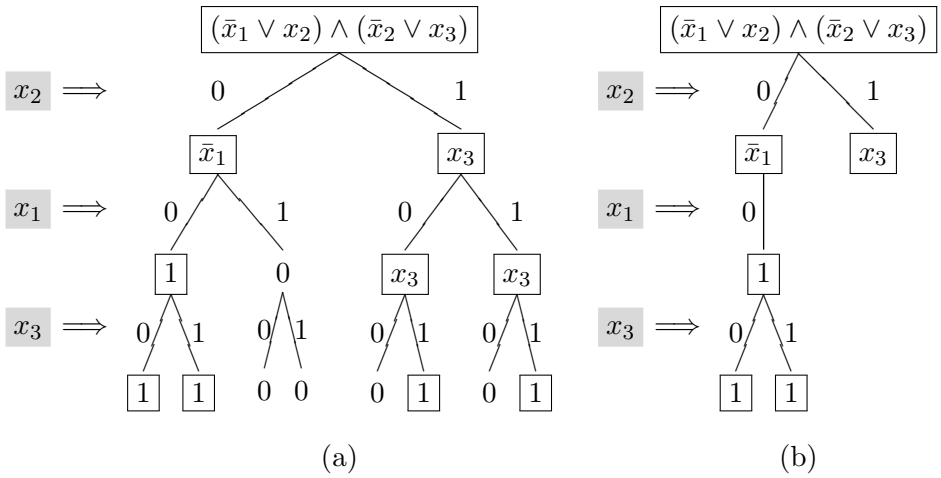

Figure 1: (a) Elimination tree and (b) a 3-cut for $\Phi = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$, due to the elimination order (x_2, x_1, x_3). The sum of the leaves is $\#\Phi = 4$. Satisfiable nodes are boxed. Note that $\#\Phi$ can already be computed from the nodes on level 1.

2 Elimination Trees, Monte Carlo Counting, and Thurley's Algorithm

Let Φ be a k-CNF, i. e., a Boolean function given in conjunctive normal form with n different variables $x_1, \ldots, x_n$ on m different clauses such that every clause has length at most k. For an arbitrary Boolean formula ϕ, let $\mathrm{Var}(\phi)$ denote the variables that occur in ϕ. Let $b : \mathrm{Var}(\phi) \to \{0, 1\}$ be a *partial* assignment of truth values to the variables in ϕ. By ϕ_b we denote the formula we obtain from ϕ by fixing in ϕ the variables according to b.

There is a nice interpretation of $\#k$-SAT in terms of complete binary trees of height n (i. e., having levels $0, \ldots, n$) that is sometimes used in the context of counting. An *elimination tree* for a k-CNF Φ can be defined as follows. Fix an *elimination order* $(y_1, \ldots, y_n)$ of the variables. Every node ϕ of the tree corresponds to a Boolean formula. The root (on level 0) of the tree is Φ. Every node ϕ on level i, $0 \leq i < n$, has two children: One child is $\phi_{y_i=0}$, the other one is $\phi_{y_i=1}$. So a path from the root to a leaf corresponds to an assignment to the variables, and the formula at a leaf is either 0 or 1. $\#\Phi$ is the number of leaves marked 1. The mark 1 is additionally broadcast to all internal nodes on a path from a 1-leaf to the root. I. e., it is visible on every node ϕ whether ϕ is satisfiable or not. For a small example, see Fig. 1(a).

Let ℓ be a positive integer. An ℓ-cut of an elimination tree is an arbitrary connected subtree that contains the root, only 1-nodes, and has ℓ leaves (w. r. t. the subtree). For an example, see Fig. 1(b). An ℓ-cut contains at most $n \cdot \ell$ nodes. From determining an ℓ-cut, immediately $\#\Phi \geq \ell$ follows. Note that the elimination order significantly influences the moment when in the elimination tree the number of satisfying assignments can be determined.

Let $\varepsilon > 0$ be an arbitrarily small number. ε is the upper bound on the admissible relative error. A number L is called an ε-approximation of $\#\Phi$ if $(1 - \varepsilon) \cdot \#\Phi \leq L \leq (1 + \varepsilon) \cdot \#\Phi$. A *randomized approximation scheme* (RAS) A is a randomized algorithm that computes on inputs Φ and error ε a number $A(\Phi, \varepsilon)$ such that $\Pr[A(\Phi, \varepsilon)$ is an ε-approximation$] \geq \frac{3}{4}$. Note that by the *median of means* method, the probability can be boosted to any number $1 - \delta$, for δ being arbitrarily close to

0. Algorithm A which usually outputs a mean is repeated $R = \Theta(\log \delta^{-1})$ times, and the median of the R values computed is returned.

2.1 Monte Carlo Counting

A very simple, general Monte Carlo approach for counting works as follows. Let $\mathcal{S}$ be the set whose cardinality has to be computed, and let $\mathcal{U} \supseteq \mathcal{S}$ be a superset of $\mathcal{S}$ such that $|\mathcal{U}|$ can be computed exactly and, preferably, fast. Sample independently and uniformly at random T elements from $\mathcal{U}$, and let L be the number of elements from $\mathcal{S}$ among these T samples. Return the number $\frac{L}{T} \cdot |\mathcal{U}|$ as an approximation of $|\mathcal{S}|$. Standard probability theory (e. g., see [13, p. 311]) gives that a sample size of $T = \Theta(\varepsilon^{-2} \cdot |\mathcal{U}|/|\mathcal{S}|)$ suffices to ensure the RAS property.

For #k-SAT, $\mathcal{U} = \{0,1\}^n$ may be chosen. If somehow a lower bound ℓ on #Φ is known, the Monte Carlo approach immediately gives a RAS with running time $T_{\mathrm{MC}} = O(\varepsilon^{-2} \cdot 2^n/\ell \cdot (n+km)) = O^*(\varepsilon^{-2} \cdot 2^n/\ell)$. If ℓ (or #Φ) is small, this is unfortunately an unsatisfactory upper bound. We refer to this algorithm as $\mathrm{MC}(\Phi, \ell, \varepsilon)$. If the reliability is amplified to $1-\delta$ by the median of means method, the running time is $O^*(\log(\delta^{-1}) \cdot \varepsilon^{-2} \cdot 2^n/\ell)$, and we write $\mathrm{MC}(\Phi, \ell, \varepsilon, \delta)$.

Note that for the similar problem #DNF where a Boolean formula Φ in disjunctive normal form is given, a set $\mathcal{U}$ can be devised [9] with $|\mathcal{U}| \leq m \cdot$ #Φ yielding a RAS with polynomial running time $O(\varepsilon^{-2} \cdot m \cdot (n+km)) = O^*(\varepsilon^{-2})$.

2.2 Thurley's RAS

The running time of MC is decreasing in ℓ. Therefore, Thurley presented an algorithm that, for input Φ and ℓ, determines whether there are at least ℓ satisfying assignments. This time the running time is increasing in ℓ. In a last step, ℓ is chosen such that it balances the running times of MC and Thurley's approach. In the following, we explain this method in detail because it is also the starting point for our improvements.

Let β_k denote the smallest known constant such that there is a randomized algorithm solving k-SAT in time $O^*(2^{\beta_k \cdot n})$. Hence, $\beta_3 \approx 0.3864$, $\beta_4 \approx 0.5548$, and, for $k \geq 5$, $\beta_k = 1 - \mu_k/(k-1)$, where

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j \cdot (j + \frac{1}{k-1})}$$

is the constant involved in the PPSZ algorithm [14].

Thurley's RAS works as follows: It has as input a k-CNF Φ and a bound ℓ. It computes an ℓ-cut of the elimination tree (if it exists). Whether a node is a 1-node can be checked fast with the known randomized k-SAT decision algorithms mentioned above where $\delta^{-1} = 2^{\Theta(n^c)}$, for some constant $c \geq 2$. We call this the ℓ-cut phase.

If it cannot find an ℓ-cut, it reports the number of subtree leaves it actually has found in the cut as estimation on #Φ (in fact, this number is even the correct value, w. h. p.). If, on the other hand, ℓ subtree leaves have been determined, this means that a lower bound of ℓ on #Φ has been determined. In this case, the Monte Carlo algorithm $\mathrm{MC}(\Phi, \ell, \varepsilon)$ is executed to approximate #Φ with error ε.

Since in the worst case 2^i nodes (formulas) are on level i and i variables have already been fixed,

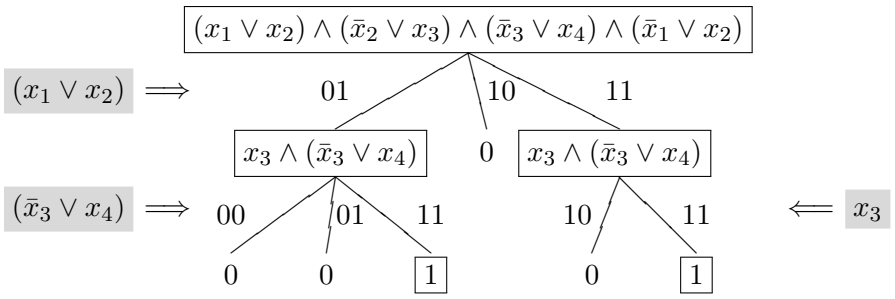

Figure 2: Possible pruned elimination tree obtained by elimination w. r. t. clause $(x_1 \vee x_2)$ at the root and, therefore, chosen variables x_1, x_2, and then on level 1, w. r. t. clause $(\bar{x}_3 \vee x_4)$ on the left node, and clause x_3 on the right node. For both nodes, the chosen variables are x_3, x_4. Note that this tree has just 6 leaves (the upper bound is 9) rather than 16 leaves as in the binary case from Sec. 2. Also note that in general, on the same level different variables at different nodes may be chosen.

the running time of the ℓ-cut phase is

$$T_{\text{cut}} = O^*\left(\sum_{i=0}^{\log(n\cdot\ell)} 2^i \cdot (2^{\beta_k})^{n-i}\right) = O^*\left(2^{\beta_k\cdot n} \cdot \sum_{i=0}^{\log(n\cdot\ell)} (2^{1-\beta_k})^i\right) = O^*(2^{\beta_k\cdot n} \cdot \ell^{1-\beta_k}) \ .$$

Hence, the overall running time is at most $T_{\text{cut}} + T_{\text{MC}} = O^*(2^{\beta_k\cdot n} \cdot \ell^{1-\beta_k} + \varepsilon^{-2} \cdot 2^n/\ell)$ which becomes $O^*\big(\varepsilon^{-2} \cdot \big(2^{1/(2-\beta_k)}\big)^n\big)$ when $\ell = 2^{n\cdot(1-\beta_k)/(2-\beta_k)}$. For $k = 3$, this is $O(\varepsilon^{-2} \cdot 1.5366^n)$, and, for $k = 4$, it is $O(\varepsilon^{-2} \cdot 1.6155^n)$.

The whole algorithm in both phases does not exploit the actual structure of Φ which opens the possibility for improvements.

3 Pruning the Tree: Taking Single Clauses into Account

The first possibility to slightly improve Thurley's RAS considers the clauses that occur in the formulas ϕ that are associated with the nodes of the elimination tree. The following approach is also the basis for the further improvements described in the subsequent sections. If C is a clause of ϕ (in the following, we will write $C \in \phi$) and consists of κ ($\kappa \leq k$) literals, there is exactly one truth assignment to the κ variables of C such that C is not satisfied. So any assignment that incorporates this specific assignment does not contribute to $\#\Phi$.

With this observation, it is possible to modify the elimination tree as follows. The root is the input k-CNF Φ. For a node's formula ϕ, choose one of its clauses, C, having κ literals, then choose k variables including all variables from C, and plug into the formula only those $2^k - 2^{k-\kappa}$ ($< 2^k$) truth assignments that satisfy C. For every formula obtained in this way, a new node is generated. That means that we increase the degree of the elimination tree from 2 to up to $2^k - 1$ and reduce its height to $\lceil n/k \rceil$. This tree is substantially smaller than the binary elimination tree described in Sec. 2. The total number of nodes is at most $O((2^k - 1)^{n/k})$. For $k = 3$, this is $O(1.91294^n)$. For a 2-CNF example, see Fig. 2.

An ℓ-cut of such an elimination tree is now used in the cut phase of Thurley's RAS. Now, the

running time of the cut phase is

$$T_{\text{Cut}} = O^*\left(\sum_{i=0}^{\log_{(2^k-1)}(n\cdot\ell)} (2^k-1)^i \cdot (2^{\beta_k})^{n-k\cdot i}\right)$$
$$= O^*(2^{\beta_k\cdot n}\cdot \ell^{1-\beta_k\cdot k/\log(2^k-1)}) \ . \tag{1}$$

If we use this pruned elimination tree to determine an ℓ-cut, and then (if necessary) using the Monte Carlo approach for computing the estimate with error at most ε, we choose ℓ for balancing the two phases as follows:

$$\log \ell = \frac{1-\beta_k}{2-\beta_k\cdot\frac{k}{\log(2^k-1)}}\cdot n.$$

We can state the result of this section with

$$p_k = \frac{1-\beta_k\cdot\left(\frac{k}{\log(2^k-1)}-1\right)}{2-\beta_k\cdot\frac{k}{\log(2^k-1)}},$$

Theorem 1. *Our pruned elimination tree algorithm is a RAS for #k-SAT. Its running time is* $O(\varepsilon^{-2}\cdot(2^{p_k})^n)$.

So for $k=3$, the running time is $O(\varepsilon^{-2}\cdot 1.5298^n)$, and for $k=4$, it is $O(\varepsilon^{-2}\cdot 1.6122^n)$ which is already slightly better than the running time of Thurley's RAS (see Table 1).

From this first improvement approach we already learn that by inspecting Φ we can reduce the size of the elimination tree. But we can see more. If we can simultaneously find several clauses that do not share variables, we even can easily exclude from the elimination tree all assignments that do not satisfy these clauses.

As the tree computed in this section may be unbalanced and different variable choices on the same level are possible, to sample assignments uniformly with the help of the tree may be not possible. That means that this pruned tree approach presumably cannot be used to also speed up the Monte Carlo approach, except for the transition from level 0 to level 1, where the mentioned problem does not occur.

In the next sections, we construct larger structures that reduce the size of the elimination tree further *and* that allow for faster uniform sampling.

4 Taking Sets of Independent Clauses into Account

For a k-CNF Φ, two clauses of Φ are called independent if they have no variable in common. A subformula $\psi \subseteq \Phi$ is called independent if the clauses in ψ are pairwise independent. ψ is called maximal if every clause in Φ shares at least one variable with a clause in ψ. Let $|\psi|$ denote the number of clauses in ψ.

4.1 Speeding up the Monte Carlo counting

In the running time T_{MC} of the Monte Carlo algorithm from Sec. 2.1, the cardinality of $\mathcal{U}$ plays a very important role. Suppose that an independent subformula ψ has been somehow (see Sec. 4.2 below)

computed. Exploiting ψ, one can use a significantly smaller set $\mathcal{U}_\psi$, namely $\mathcal{U}_\psi = \{b \in \{0,1\}^n \mid b \text{ satisfies } \psi\}$ which is obviously a superset of $\mathcal{S}_\Phi = \{b \in \{0,1\}^n \mid b \text{ satisfies } \Phi\}$. As the clauses in ψ are independent, the size of $\mathcal{U}_\psi$ can be bounded as follows: $|\mathcal{U}_\psi| \leq (2^k-1)^{|\psi|} \cdot 2^{n-k\cdot|\psi|} = 2^n \cdot (1-2^{-k})^{|\psi|}$. Sampling assignments from $\mathcal{U}_\psi$ is simply done by choosing u. a. r. one of the at most 2^k-1 satisfying assignments to the variables of each $C \in \psi$, and assigning 0 or 1 to each of the remaining variables. Provided $\ell \leq \#\Phi = |\mathcal{S}_\Phi|$, the running time is $T_{\mathrm{MC}} = O^*(\frac{2^n}{\varepsilon^2 \cdot \ell} \cdot (1-2^{-k})^{|\psi|})$. In terms of elimination trees, ψ is used during the sampling for going from the root to the nodes on level 1. The remaining sampling is performed as in the binary elimination tree case described in Sec. 2.1.

4.2 Controlling the decision between elimination and recursion

Until now, we assumed ψ already available. Now we present a method for either obtaining a sufficiently large number of independent clauses or, if this is not possible, to transform the k-CNF to a not too large number of $(k-1)$-CNFs. This method has been introduced by Hofmeister et al. [5, 6] and works on inputs Φ and integer $\hat{m}$ as follows. In particular, it also controls whether the algorithm from Sec. 3 is applied or recursive calls on nodes of the elimination tree:

Starting with $\psi = \emptyset$, greedily increase ψ as much as possible. Note that ψ is now a maximal independent subformula.

If $|\psi| \geq \hat{m}$, return ψ.

Otherwise, use ψ to generate the at most $(2^k-1)^{\hat{m}}$ different $(k-1)$-CNFs on level 1 of the elimination tree, *recursively* solve $\#(k-1)$-SAT with these formulas as input, return the sum of the estimations, and report the overall counting task as finished.

We will refer to this method as $\textsc{Red}(\Phi, \hat{m}, \varepsilon, \delta)$. When $\textsc{Red}(\Phi, \hat{m}, \varepsilon, \delta)$ has been executed and returned ψ with $\psi \geq \hat{m}$, store ψ and if the pruned elimination tree algorithm from Sec. 3 reports the existence of at least ℓ satisfying assignments, use ψ in the Monte Carlo algorithm from Sec. 4.1. Since RED makes random decisions when solving the $\#(k-1)$-SAT instances, we specify in the call of RED also δ such that the probability that RED returns either an ε-approximation or a set ψ of at least $\hat{m}$ independent clauses is at least $1-\delta$.

Together with the modified algorithm to determine an ℓ-cut from the previous section, we obtain an improved RAS. Since the algorithm for solving $\#k$-SAT for $k \geq 4$ includes the algorithm itself for solving the occurring $\#(k-1)$-SAT instances, the runtime has no closed form, but can be calculated recursively. We terminate the recursion for $k = 2$ and use Wahlström's (even deterministic) algorithm [18], solving #2-SAT in time $O(1.2377^n)$. Although our method works for any k, we state our result for the cases $k \in \{3, 4\}$ only.

Theorem 2. *The algorithm described above is a RAS solving $\#k$-SAT. For $k = 3$, its running time is $O(\varepsilon^{-2} \cdot 1.5181^n)$, and for $k = 4$, its running time is $O(\varepsilon^{-2} \cdot (\log(\delta^{-1}) + n) \cdot 1.6105^n)$.*

Proof. (a) Let $k = 3$. For arbitrary $\hat{m}$, RED either finds a set ψ of $\hat{m}$ independent clauses or it has to solve at most $7^{\hat{m}}$ #2-SAT instances, each over the remaining $n - 3\hat{m}$ variables. This is done (even deterministically) with Wahlström's algorithm [18] in time $O(1.2377^{n-3\hat{m}})$. So the overall running time of RED is $T_{\mathrm{RED}} = O((7 \cdot 1.2377^{-3})^{\hat{m}} \cdot 1.2377^n) = O(3.6920^{\hat{m}} \cdot 1.2377^n)$.

As mentioned above, the running time of MC in case of $|\psi| \geq \hat{m}$ is at most $T_{\mathrm{MC}} = O^*(2^n/\ell \cdot (\frac{7}{8})^{\hat{m}})$ for $\ell \leq \#\Phi$, and the running time for finding an ℓ-cut (or being sure that none exists) is due to Eq. (1), $T_{\mathrm{cut}} = O^*(2^{\beta_3 \cdot n} \cdot \ell^{1-\beta_3 \cdot 3/\log 7})$. The break-even point for T_{RED}, T_{Cut} and T_{MC} and therefore the worst case occurs for choosing $\hat{m} = 0.1563\,n$ and $\ell = 1.2903^n$.

(b) The proof for the case $k \geq 4$ is analogous. The only difference is that RED is not deterministic anymore. In order to get an ε-approximation with probability at least $1-\delta$ in the end, we require from the nested $\#(k-1)$-SAT algorithm an ε-approximation with probability at least $1-\delta/7^{\hat{m}} \leq 1-\delta/2^n$, so the necessary number of repetitions for each recursive call of the RAS for $\#(k-1)$-SAT has to be increased by n (additively) to ensure the desired success probability.

For the time bound for #4-SAT, we now use the bound from (a) for #3-SAT and obtain the claimed result by the same straight-forward calculations as in the first case. In the worst case, the parameters are $\hat{m} = 0.0587\,n$ and $\ell = 1.2372^n$. □

5 Taking Large Independent Structures into Account

One can easily see that both the modified version of the Monte Carlo algorithm and the modified method for calculating an ℓ-cut do not require the elements of ψ to be clauses. Both can be generalized for ψ being a set of pairwise independent, arbitrary subformulas with constant size. E. g., in $\psi = \sigma_1 \wedge \sigma_2$ with $\sigma_1 = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$ and $\sigma_2 = (x_5 \vee \bar{x}_6) \wedge (x_5 \vee x_7)$, σ_1 and σ_2 are independent. In order to recognize the subformulas, we write (in general) $\psi = \{\sigma_1, \ldots, \sigma_{\hat{m}}\}$. We call a single subformula σ a STRUCT. If every clause shares at least one variable with a clause in ψ, we call ψ maximal. Due to their constant size, it is possible to compute the number of their satisfying assignments. We first show how STRUCTs can be used to improve the algorithms MC and CUT, then we show how to construct them, and finally we present the overall RAS.

5.1 Monte Carlo counting and cut phase if many independent subformulas are known

The generalized version of MC is presented below as Algorithm 1, the generalized version of the ℓ-cut phase as CUT (Algorithm 2). For a subformula σ, let n_σ denote the number of different variables in σ, and L_σ the number of satisfying assignments of σ. Note that CUT uses a global counter L (only set to 0 in the first call) and globally aborts as soon as L reaches ℓ. Otherwise, it returns L after finishing.

Our results from Sec. 4 lead to the following lemmas.

Lemma 1. *Let $\psi = \{\sigma_1, \ldots, \sigma_{\hat{m}}\}$ be a set of constant-sized, pairwise independent subformulas of Φ. If $\#\Phi \geq \ell$, MC(Φ, ψ, ℓ, ε, δ) returns with probability at least $1-\delta$ an ε-approximation for $\#\Phi$. The running time is*

$$T_{\mathrm{MC}} = O^*\Big(\log(\delta^{-1}) \cdot \frac{2^n}{\varepsilon^2 \,.\, \ell} \cdot \prod_{\sigma \in \psi} \frac{L_\sigma}{2^{n_\sigma}}\Big) \ .$$

Proof. We sample from the set of all assignments satisfying ψ. The size of this set is

$$2^{n - \sum_{\sigma \in \psi} n_\sigma} \cdot \prod_{\sigma \in \psi} L_\sigma = 2^n \cdot \prod_{\sigma \in \psi} \frac{L_\sigma}{2^{n_\sigma}} \ .$$

The result follows directly from Sec. 2.1. □

Lemma 1 generalizes the result of Sec. 4.1, where we used independent clauses, to the new situation, enabling the use of more complex, but still pairwise independent subformulas. The same generalization can be applied to the cut phase from Sec. 3.

Algorithm 1: MC$(\Phi, \psi, \ell, \varepsilon, \delta)$

input : k-SAT instance Φ, set $\psi = \{\sigma_1, \ldots, \sigma_{\hat{m}}\}$ of independent subformulas, parameters ℓ, ε, δ

output : w. p. $1 - \delta$: ε-approximation on $\#\Phi$

$U := 2^n \cdot \prod_{\sigma \in \psi} \frac{L_\sigma}{2^{n_\sigma}}$; $\quad T := \log(\delta^{-1}) \cdot \frac{U}{\varepsilon^2 \cdot \ell}$; $\quad L := 0$;

repeat T times

  forall the $\sigma \in \psi$ **do**

    assign u. a. r. one of the assignments satisfying σ to its variables;

  forall the $x \in \mathrm{Vars}(\Phi)$ *not already fixed* **do**

    assign u. a. r. 0 or 1 to x;

  if Φ *is satisfied* **then**

    $L := L + 1$;

return $L/T \cdot U$;

An important difference is that in Sec. 3, we could assume that for each node ϕ there always is a clause to generate new nodes since otherwise calculating $\#\phi$ would be trivial. Now it may happen that ψ becomes empty. In this case, we fall back to using single clauses, but due to the maximality of ψ, those clauses have length at most $k - 1$. The following lemma states the running time of Cut in both cases.

Lemma 2. *If $\#\Phi \leq \ell$, then, with probability at least $1 - \delta$,* Cut$(\Phi, \psi, \ell, \delta)$ *returns $\#\Phi$ exactly. If $\ell \leq \prod_{\sigma \in \psi} L_\sigma$, let $\psi' = \{\sigma_1, \ldots, \sigma_{\hat{m}'}\} \subset \psi$ denote a subset of ψ, where $\hat{m}'$ is chosen such that $\prod_{i=1}^{\hat{m}'-1} L_{\sigma_i} < \ell \leq \prod_{i=1}^{\hat{m}'} L_{\sigma_i}$. If $\ell \geq \prod_{\sigma \in \psi} L_\sigma$, set $\hat{m}' := \hat{m} + \log_{(2^{k-1}-1)}(\ell \cdot \prod_{\sigma \in \psi} L_\sigma^{-1})$.*[2] *Then the running time of* Cut *is*

$$T_{\mathrm{Cut}} = O^*\left(\log(\delta^{-1}) \cdot 2^{\beta_k \cdot n} \cdot \prod_{\sigma \in \psi'} \frac{L_\sigma}{2^{\beta_k \cdot n_\sigma}}\right),$$

if $\ell \leq \prod_{\sigma \in \psi} L_\sigma$, and

$$T_{\mathrm{Cut}} = O^*\left(\log(\delta^{-1}) \cdot 2^{\beta_k \cdot n} \cdot \left(\frac{2^{k-1}-1}{2^{\beta_k \cdot (k-1)}}\right)^{\hat{m}'-\hat{m}} \cdot \prod_{\sigma \in \psi} \frac{L_\sigma}{2^{\beta_k \cdot n_\sigma}}\right)$$

otherwise.

Proof. Assume $\#\Phi \leq \ell$, so the algorithm searches for a pruned elimination tree with ℓ leaves. Since 2^n is a rigorous bound on the number of formulas that have to be tested on satisfiability, the probability that the satisfiability check always gives the right answer is at least $1 - 2^n \cdot \delta / 2^n = 1 - \delta$.

Now we focus on the run-time analysis. Basically, for each node of the tree, at most $2^k - 1$ different k-SAT instances must be solved. So we have to analyze for each node ϕ, how many variables of ϕ are fixed. On the i-th level of the tree, for $i \leq \hat{m}$, the variables of the first i subformulas of ψ are fixed. If $i > \hat{m}$, all the variables of ψ and additionally the variables of $i - \hat{m}$ of the remaining

[2] Roughly speaking, $\hat{m}'$ is the minimum number of elements of ψ respectively additional clauses that have to be chosen by Cut until the total of ℓ leaves of the elimination tree can be achieved.

Algorithm 2: CUT(Φ, ψ, ℓ, δ) // Note: L is a global variable

input : k-SAT instance Φ, set of subformulas ψ, parameters ℓ, δ
output : w. p. $1-\delta$: either $\#\Phi$ exactly, or the message that $\#\Phi \geq \ell$

if Φ *is unsatisfiable (w. p. at least* $1-\delta/2^n$*)* **then**
  return L;
else
  $L := L+1$;
  if $L \geq \ell$ **then**
    global abort;
if $\psi \neq \emptyset$ **then**
  choose the first $\sigma \in \psi$;
else
  choose $\sigma \in \Phi$;
forall the *Assignment* b *to* $\mathrm{Vars}(\sigma)$ *satisfying* σ **do**
  CUT(Φ_b, $\psi \setminus \{\sigma\}$, ℓ, δ);
return L;

clauses are fixed. Note that due to the maximality of ψ, those clauses have size at most $k-1$. Let N_i denote the number of nodes in level i. For $i \leq \hat{m}$, the nodes on level i have $\sum_{j=1}^{i} n_{\sigma_j}$ fixed variables, so the time T_i required for processing only the nodes on level i is

$$T_i = O^*\left((n+\log(\delta^{-1}))\cdot N_i \cdot 2^{\beta_k \cdot n - \sum_{j=1}^{i} n_{\sigma_j}}\right) = O^*\left(\log(\delta^{-1})\cdot N_i \cdot 2^{\beta_k \cdot n} \cdot \prod_{j=1}^{i} 2^{-n_{\sigma_j}}\right).$$

For $i > \hat{m}$, the number of fixed variables is $\sum_{j=1}^{\hat{m}} n_{\sigma_j} + (k-1)\cdot(i-\hat{m})$. In this case, we obtain

$$T_i = O^*\left(\log(\delta^{-1})\cdot N_i \cdot 2^{\beta_k \cdot n} \cdot 2^{-(k-1)\cdot(i-\hat{m})} \cdot \prod_{j=1}^{\hat{m}} 2^{-n_{\sigma_j}}\right).$$

Since a node on level $i-1$ has out-degree at most L_{σ_i} for $i \leq \hat{m}$ and $2^{k-1}-1$ otherwise, on level i there are at most $N_i \leq \prod_{j=1}^{i} L_{\sigma_j}$ nodes if $i \leq \hat{m}$ and $N_i \leq \left(2^{k-1}-1\right)^{i-\hat{m}} \prod_{j=1}^{\hat{m}} L_{\sigma_j}$ nodes if $i \leq \hat{m}$. Of course, $N_i \leq \ell$ is also a bound we have for every N_i. Let h be the maximum length of a path from the root to a leaf. Then the overall running time T_{Cut} of CUT is (where we omit the $O^*(.)$ and

the factor $\log(\delta^{-1})$)

$$
\begin{aligned}
T_{\text{Cut}} &= \sum_{i=0}^{h} T_i = \sum_{i=0}^{\hat{m}} T_i + \sum_{i=\hat{m}+1}^{h} T_i \\
&\leq \sum_{i=0}^{\hat{m}} \min\left\{ \ell, \prod_{j=1}^{i} L_{\sigma_j} \right\} \cdot 2^{\beta_k \cdot n} \cdot \prod_{j=1}^{i} 2^{-\beta_k \cdot n_{\sigma_j}} \\
&\quad + \sum_{i=\hat{m}+1}^{h} \min\left\{ \ell, \left(2^{k-1}-1\right)^{i-\hat{m}} \prod_{j=1}^{\hat{m}} L_{\sigma_j} \right\} \cdot 2^{\beta_k \cdot n} \cdot 2^{-(k-1)\cdot(i-\hat{m})} \cdot \prod_{j=1}^{\hat{m}} 2^{-n_{\sigma_j}} \\
&= 2^{\beta_k \cdot n} \cdot \sum_{i=0}^{\hat{m}} \min\left\{ \ell \cdot \prod_{j=1}^{i} 2^{-\beta_k \cdot n_{\sigma_j}}, \prod_{j=1}^{i} \frac{L_{\sigma_j}}{2^{\beta_k \cdot n_{\sigma_j}}} \right\} \\
&\quad + 2^{\beta_k \cdot n} \cdot \sum_{i=\hat{m}+1}^{h} \min\left\{ \ell \cdot 2^{-(k-1)\cdot(i-\hat{m})} \cdot \prod_{j=1}^{\hat{m}} 2^{-n_{\sigma_j}}, \left(\frac{2^{k-1}-1}{2^{k-1}}\right)^{i-\hat{m}} \prod_{j=1}^{\hat{m}} \frac{L_{\sigma_j}}{2^{n_{\sigma_j}}} \right\}
\end{aligned}
$$

One can easily see that in both sums, the first term inside the min is decreasing in i. Since for any σ we are going to use, we have $L_\sigma / 2^{\beta_k \cdot n_\sigma} > 1$ (and since otherwise the runtime of the algorithm would be even better), the second term inside each min is increasing in i. So, up to a constant factor, the sum is equal to the summand where both terms inside the minimum have the same value, which is the case for $i = \hat{m}'$. This finishes the proof. □

Note that, in the case of a node on level $i > \hat{m}$, CUT would not have to solve a k-SAT instance anymore but only a $(k-1)$-SAT instance. However, such considerations would lead to only a small improvement of the running time, so we sacrificed the benefit from this observation in order to simplify our analysis.

5.2 Finding large sets of independent subformulas

It remains to provide some method that collects the set ψ of subformulas. We call these subformulas STRUCTs and, in order to compute them, we generalize algorithm RED from Sec. 4.2 as follows. We start with the $\hat{m}$ independent clauses as initial set of STRUCTs. Now we search iteratively for further clauses in Φ that extend in a controlled way the STRUCTs we already have such that they remain independent. Since the STRUCTs must have constant size, there are STRUCTs that we do not want to extend anymore. We call a variable x occurring in STRUCT σ closed, if we do not want to add further clauses to σ that contain x. σ is called closed if it has only closed variables. For example, assume we already found the two STRUCTs $\sigma_1 = \{(x_1 \vee x_2), (\bar{x}_2 \vee x_3)\}$ and $\sigma_2 = \{(x_4 \vee x_5)\}$ where x_2 and x_5 are closed variables. The clause $(x_2 \vee x_6) \in \Phi$ would not be considered because it contains the closed variable x_2. But the clause $(x_3 \vee x_4) \in \Phi$ can be used to extend and connect the two STRUCTs to the single STRUCT $\sigma_3 = \{(x_1 \vee x_2), (\bar{x}_2 \vee x_3), (x_3 \vee x_4), (x_4 \vee x_5)\}$ which is, in our example, a closed STRUCT.

Of course, there is no guarantee that the extension phase runs until every STRUCT is closed. In the above example, without the clause $(x_3 \vee x_4)$ only the two non-closed STRUCTs can be obtained.

But if we ensure that each STRUCT has at least one closed variable in every clause, the set ψ is maximal, meaning that every clause in Φ contains at least one variable that is a closed variable of some STRUCT $\sigma \in \psi$. So, in order to reduce Φ to a $\#(k-1)$-SAT instance, one only has to fix the closed variables. If RED finds a large number of STRUCTs, then MC (Algorithm 1) runs faster. If there are only a few, then fixing their closed variables leads to only a few $\#(k-1)$-SAT instances to be solved. For a given STRUCT σ, let f_σ denote the number of closed variables and w_σ the number of assignments to the closed variables that do not already violate a clause of σ, i. e., $f_\sigma = n_\sigma$ and $w_\sigma = L_\sigma$ for every closed STRUCT.

The algorithm for grouping the STRUCTs is given in Algorithm 3. α_k refers to a constant determined in the proof of Theorem 3 below. One can think of it as the value such that for the $\#k$-SAT problem an ε-approximation can be obtained in time $O^*(\varepsilon^{-2} \cdot \log(\delta^{-1}) \cdot \alpha_k^n)$ with probability at least $1-\delta$. The algorithm accesses a library, containing for every occurring STRUCT σ the closed variables. This library assures that the size of the STRUCTs is constant.

Algorithm 3: RED$(\Phi, \ell, \varepsilon, \delta)$

input : k-SAT instance Φ, parameters ℓ, ε, δ
output : either a set ψ of STRUCTs or, w. p. $1-\delta$: ε-approximation L of $\#\Phi$

$\psi := \emptyset$;
while *there is a clause* $C \in \Phi$ *that contains no closed variable of a* STRUCT *in* ψ **do**
  $\chi :=$ all STRUCTs of ψ that have a variable with C in common;
  $\sigma :=$ new STRUCT created from C and the STRUCTs from χ;
  $\psi := (\psi - \chi) \cup \{\sigma\}$;
/* α_k: const. determined in Thm. 3 */
if $\alpha_{k-1}^n \cdot \prod_{\sigma \in \psi} w_\sigma \cdot \alpha_{k-1}^{-f_\sigma} < \alpha_k^n$ **then**
  $L := 0$;
  forall the *assignments* b *to the variables in* ψ *that satisfy* ψ **do**
    $L := L +$ INDEPSUBFORM_RAS$(\Phi_b, \varepsilon, \delta/2^n)$;
  return L;
else
  return ψ;

Lemma 3. *Assuming $\#(k-1)$-SAT can be approximated in time $O^*(\alpha_{k-1}^n)$, RED$(\Phi, \ell, \varepsilon, \delta)$ (Algorithm 3) returns either w. p. at least $1-\delta$ an ε-approximation of $\#\Phi$ or a set ψ of pairwise independent STRUCTs such that $\alpha_{k-1}^n \cdot \prod_{\sigma \in \psi} w_\sigma / \alpha_{k-1}^{f_\sigma} \geq \alpha_k^n$. It runs in time*

$$O^*\left(\varepsilon^{-2} \cdot \log(\delta^{-1}) \cdot \alpha_{k-1}^n \cdot \prod_{\sigma \in \psi} \frac{w_\sigma}{\alpha_{k-1}^{f_\sigma}}\right).$$

Proof. If $\alpha_{k-1}^n \cdot \prod_{\sigma \in \psi} \frac{w_\sigma}{\alpha_{k-1}^{f_\sigma}} > t$ after the **while**-loop, which obviously runs in polynomial time, is finished, then the algorithm returns the set ψ with the claimed property. Otherwise it enumerates all assignments for the closed variables of the STRUCTs of ψ that do not already cause ψ to be evaluated to 0. Since the STRUCTs in ψ are pairwise independent, there are exactly $\prod_{\sigma \in \psi} w_\sigma$ such assignments.

For each of these assignments, RED starts an algorithm that w. p. at least $1 - \delta/2^n$ returns an ε-approximation of the resulting $\#(k-1)$-SAT instances. The probability that each of the $\prod_{\sigma\in\psi} w_\sigma$ runs actually returns an ε-approximation is therefore at least $1 - \delta/2^n \cdot \prod_{\sigma\in\psi} w_\sigma \geq 1 - \delta$. Since $\sum_{\sigma\in\psi} f_\sigma$ variables are fixed, each of the $\#(k-1)$-SAT instances has $n - \sum_{\sigma\in\psi} f_\sigma$ variables and to approximate it within the given parameters takes time

$$O^*\left(\varepsilon^{-2} \cdot (n + \log(\delta^{-1})) \cdot \alpha_{k-1}^{n-\sum_{\sigma\in\psi} f_\sigma}\right) = O^*\left(\varepsilon^{-2} \cdot \log(\delta^{-1}) \cdot \alpha_{k-1}^{n} \cdot \prod_{\sigma\in\psi} \alpha_{k-1}^{-f_\sigma}\right).$$

This finishes the proof. □

5.3 All things come together: The new randomized approximation scheme

Combining all results, we are now able to state our main algorithm that solves $\#k$-SAT. INDEPSUBFORM_RAS (Algorithm 4) shows how to combine the Algorithms 1, 2 and 3.

Algorithm 4: INDEPSUBFORM_RAS

input : k-CNF Φ, ε, δ
output : Approximated number L of satisfying assignments of Φ

```
if RED(Φ, ℓ, ε, δ) returns L then
    return L
else
    ψ := set of STRUCTs returned by RED;
    if CUT(Φ, ψ, ℓ, δ) returns value #Φ < ℓ then
        return #Φ;
    else
        return MC(Φ, ψ, ℓ, ε, δ);
```

Theorem 3. INDEPSUBFORM_RAS *(Algorithm 4) is a RAS running in time* $O(\varepsilon^{-2} \cdot \log(\delta^{-1}) \cdot 1.51426^n)$ *for #3-SAT and in time* $O(\varepsilon^{-2} \cdot \log(\delta^{-1}) \cdot 1.60816^n)$ *for #4-SAT.*

Proof. The results follow from calculating the break-even points of the time bounds in Lemma 1, Lemma 2 and Lemma 3. For our version of algorithm RED, we used the STRUCTs and made the decisions about which variable to declare closed as described in Table 2 and 3, resp. We declared every other STRUCT that is not listed as closed (by setting all its variables closed) and set in Algorithm 3 $\alpha_3 = 1.51426$ for $k = 3$ and $\alpha_4 = 1.60816$ for $k = 4$.

For $k = 3$, in the worst case, $\ell = 1.28794^n$ and there are $0.05252\,n$ closed STRUCTs of the form $\{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_4 \vee x_5), (x_2 \vee x_6 \vee x_7), (x_4 \vee x_8 \vee x_9)\}$. For $k = 4$, in the worst case, $\ell = 1.23823^n$ and there are $0.01785\,n$ closed STRUCTs of the form $\{(x_1 \vee x_2 \vee x_3 \vee x_4), (x_1 \vee x_5 \vee x_6 \vee x_7), (x_2 \vee x_8 \vee x_9 \vee x_{10}), (x_5 \vee x_{11} \vee x_{12} \vee x_{13})\}$. This leads to the claimed bounds. □

Note that, for any k, our method runs in time $O^*(\alpha_k^n)$ with α_k depending on β_k and α_{k-1}. For all k, $\alpha_k < 2^{1/(2-\beta_k)} =: \vartheta_k$ (Thurley's running time), even if we define STRUCTs consisting of just a single clause as closed. E. g., $\alpha_5 \approx 1.6694 < \vartheta_5 \approx 1.6712$.

Table 2: Non-closed STRUCTs for $k = 3$

Type σ	L_σ	closed Vars.
$\{(x_1 \vee x_2 \vee x_3)\}$	7	$\{x_3\}$
$\{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_4 \vee x_5)\}$	25	$\{x_1\}$
$\{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_2 \vee x_4)\}$	13	$\{x_1\}$
$\{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_4 \vee x_5), (x_2 \vee x_6 \vee x_7)\}$	89	$\{x_1, x_2\}$

Table 3: Non-closed STRUCTs for $k = 4$

Type σ	L_σ	closed Vars.
$\{(x_1 \vee x_2 \vee x_3 \vee x_4)\}$	15	$\{x_4\}$
$\{(x_1 \vee x_2 \vee x_3 \vee x_4), (x_1 \vee x_5 \vee x_6 \vee x_7)\}$	113	$\{x_1\}$
$\{(x_1 \vee x_2 \vee x_3 \vee x_4), (x_1 \vee x_5 \vee x_6 \vee x_7), (x_2 \vee x_8 \vee x_9 \vee x_{10})\}$	851	$\{x_1, x_2\}$

References

[1] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8:121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

[2] Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science*, 81:49–64, 1991. doi:10.1016/0304-3975(91)90315-S.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[4] Timon Hertli. 3-SAT faster and simpler – Unique-SAT bounds for PPSZ hold in general. In *Proc. 52nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 277–284, 2011. doi:10.1109/FOCS.2011.22.

[5] Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe. A probabilistic 3-SAT algorithm further improved. In *Proc. 19th Int. Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 193–202, 2002. doi:10.1007/3-540-45841-7_15.

[6] Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe. Randomized algorithms for 3-SAT. *Theory of Computing Systems*, 40:249–262, 2007. doi:10.1007/s00224-005-1275-6.

[7] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *Proc. 23rd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 961–972, 2012.

[8] Mark Jerrum and Alistair Sinclair. The Markov chain Monte-Carlo method: An approach to approximate counting and integration. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 482–520. PWS, 1996.

[9] Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10:429–448, 1989. doi:10.1016/0196-6774(89)90038-2.

[10] Konstantin Kutzkov. New upper bound for the #3-SAT problem. *Information Processing Letters*, 105:1–5, 2007. doi:10.1016/j.ipl.2007.06.017.

[11] Kazuhisa Makinoa, Suguru Tamaki, and Masaki Yamamoto. Derandomizing HSSW algorithm for 3-SAT. In *Proc. 17th Int. Computing and Combinatorics Conference (COCOON)*, pages 1–12, 2011. doi:10.1007/978-3-642-22685-4_1.

[12] Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's k-SAT algorithm. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, pages 245–252, 2011. doi:10.1145/1993636.1993670.

[13] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[14] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *Journal of the ACM*, 52:337–364, 2005. doi:10.1145/1066100.1066101.

[15] Yannis C. Stamatiou. Threshold phenomena: The computer scientist's viewpoint. In *Bulletin of the EATCS*, volume 80, pages 199–234, 2003.

[16] Marc Thurley. An approximation algorithm for #k-SAT. In *Proc. 29th Int. Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 78–87, 2012. doi:10.4230/LIPIcs.STACS.2012.78.

[17] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979. doi:10.1137/0208032.

[18] Magnus Wahlström. A tighter bound for counting max-weight solutions to 2SAT instances. In *Proc. 3rd Int. Workshop on Parameterized and Exact Computation (IWPEC)*, pages 202–213, 2008. doi:10.1007/978-3-540-79723-4_19.

[19] Wenhui Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 155:277–288, 1996. doi:10.1016/0304-3975(95)00144-1.