Solving and Sampling with Many Solutions: Satisfiability and Other Hard Problems*†

Jean Cardinal[1], Jerri Nummenpalo[2], and Emo Welzl[3]

1 **Université Libre de Bruxelles (ULB), Computer Science Department, Brussels, Belgium**
jcardin@ulb.ac.be
2 **ETH Zürich, Department of Computer Science, Zürich, Switzerland**
njerri@inf.ethz.ch
3 **ETH Zürich, Department of Computer Science, Zürich, Switzerland**
emo@inf.ethz.ch

Abstract

We investigate parameterizing hard combinatorial problems by the size of the solution set compared to all solution candidates. Our main result is a uniform sampling algorithm for satisfying assignments of 2-CNF formulas that runs in expected time $O^*(\varepsilon^{-0.617})$ where ε is the fraction of assignments that are satisfying. This improves significantly over the trivial sampling bound of expected $\Theta^*(\varepsilon^{-1})$, and on all previous algorithms whenever $\varepsilon = \Omega(0.708^n)$. We also consider algorithms for 3-SAT with an ε fraction of satisfying assignments, and prove that it can be solved in $O^*(\varepsilon^{-2.27})$ deterministic time, and in $O^*(\varepsilon^{-0.936})$ randomized time. Finally, to further demonstrate the applicability of this framework, we also explore how similar techniques can be used for vertex cover problems.

1998 ACM Subject Classification F.2.2 Theory of Computation, Nonnumerical Algorithms and Problems, Computations on discrete structures, G.2.1. Discrete Mathematics, Combinatorics, Combinatorial algorithms

Keywords and phrases Satisfiability, Sampling, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2017.11

1 Introduction

In order to cope with the computational complexity of combinatorial optimization and satisfiability problems without sacrificing correctness guarantees, one can consider a family of instances for which a certain parameter is bounded, and analyze the complexity of algorithms as a function of this parameter. While it is now commonplace in combinatorial optimization to define the parameter as the *size* of a solution, we here consider computationally hard problems parameterized by the *number* of solutions. More precisely, we will consider satisfiability problems in which we are promised that a fraction at least ε of all possible assignments are satisfying, and graph covering problems in which a fraction at least ε of all vertex subsets of a certain size are solutions.

Counting and sampling solutions to CNF formulas and more generally to CSP formulas has important practical applications. For example, in verification and artificial intelligence [12];

* This work started at the 2016 Gremo Workshop on Open Problems (GWOP), on June 6-10 at St. Niklausen, OW, Switzerland.
† A full version of this paper is available on arXiv [2], https://arxiv.org/abs/1708.01122.


© Jean Cardinal, Jerri Nummenpalo and Emo Welzl;
licensed under Creative Commons License CC-BY

and Bayesian inference [13]. Recent algorithmic developments have made possible practical algorithms that can tackle industrial scale problems [10].

In contrast to that line of work we focus on the exact complexity of sampling, in particular to sampling solutions for 2-CNF formulas, and show that we can significantly improve on the *trivial sampling algorithm* that repeatedly samples uniformly in the search space and terminates after ε^{-1} steps on average. A few previous works have also considered satisfiability problems under the promise that there are many solutions, most notably from Hirsch [6], and more recently from Kane and Watanabe [9]. Their focus has been on deterministic algorithms and we extend their work while also adding the consideration of randomized algorithms for k-SAT.

Before detailing our contributions more precisely, we briefly summarize the current state of knowledge regarding this family of questions.

1.1 Background and previous work on satisfiability

Hirsch [6] developed a deterministic algorithm that finds a satisfying assignment for a k-CNF formula F with an ε fraction of satisfying assignments in time $O^*(\varepsilon^{-\delta_k})$ where $(\delta_k)_{k=2}^{\infty}$ is a positive increasing sequence defined by the roots of the characteristic polynomials of certain recurrence relations. The constant obtained for $k = 3$ is $\delta_3 \approx 7.27$. The main idea in his algorithm is that such formulas F have *short implicants* which are satisfying assignments that need to fix only few variables – in this case only $O(\log \varepsilon^{-1})$ many – and such assignments can be found relatively fast with a branching algorithm. Trevisan [17] proposed a similar algorithm to that of Hirsch but with an explicit running time of $O^*(\varepsilon^{-(\ln 4)k2^k})$. Although his algorithm is slightly simpler, the performance guarantees, at least for small k, are worse.

Kane and Watanabe [9] looked at general CNF formulas in a similar setting. They assume that $\varepsilon \geq 2^{-n^{\delta}}$, that the number of clauses is bounded by $n^{1+\delta'}$ and that $\delta + \delta' < 1$. Under these conditions they show that the formula has a short implicant that only fixes a linear fraction of the variables and they provide a $O^*(2^{n^{\beta}})$ time algorithm for finding a solution with $\beta < 1$.

Classical derandomization tools naturally apply in this context. For arbitrary CNF formulas on n variables with $\varepsilon 2^n$ satisfying assignments, one can obtain a deterministic algorithm by using a pseudorandom generator that ε-fools depth-2 circuits. A result by De et al. [3] provides such pseudorandom generators with seed length $O\left(\log n + \log^2 \frac{m}{\varepsilon} \log\log \frac{m}{\varepsilon}\right)$. By enumerating over all seeds, we obtain a running time of $O^*\left(\left(\frac{n}{\varepsilon}\right)^{c \cdot \log \frac{n}{\varepsilon}}\right)$ for some constant c (assuming there are $\mathrm{poly}(n)$ clauses). A recent result of Servedio and Tan improves this running time to $n^{\tilde{O}(\log\log n)^2}$ for any $\varepsilon \geq 1/\mathrm{poly}\log(n)$ [16].

We let *Sample-2-SAT* denote the problem of sampling exactly and uniformly a satisfying assignment. Due to self-reducibility of satisfiability, any algorithm for the counting problem #2-SAT can be used to solve Sample-2-SAT with only a multiplicative polynomial loss in runtime. In fact, so far the best algorithm for Sample-2-SAT is Wahlström's #2-SAT algorithm [18] that runs in time $O(1.238^n)$. In contrast to the exponential time algorithms, 2-SAT can be solved in linear time with the classical algorithm of Aspvall et al. [1]. We note that while Sample-2-SAT is between 2-SAT and #2-SAT in complexity, under the assumption $RP \neq NP$ it is not possible to uniformly or even almost uniformly sample satisfying assignments in polynomial time. We can use a simple threefold reduction to prove this:

- The constraints for an independent set in a graph can be modeled as a 2-SAT formula. Therefore a polynomial time algorithm for Sample-2-SAT would give a polynomial time

algorithm for *Sample-IS*. (sampling uniformly among independent sets of any size). The same holds for approximate versions of the problems.
- Such sampling algorithms would yield a fully polynomial randomized approximation scheme (FPRAS) for #IS. See for example the article of Jerrum et al. [8].
- Lastly, such an FPRAS exists only if $RP = NP$. For details see for example the book by Jerrum [7, Chapter 7, Proposition 7.7].

Even when relaxing Sample-2-SAT to almost uniform sampling, the best algorithm is still the one based on Wahlström's counting algorithm. This is in contrast to k-CNF formulas with $k \geq 3$ which have an exponential gap between exact and almost uniform sampling. More precisely, the gap is between exact and approximate counting. See Schmitt and Wanka [14] for a table of the best algorithms.

1.2 Our results

In Section 2 we recall Hirsch's [6] algorithm for finding a satisfying assignment for a k-CNF F with a fraction ε of satisfying assignments. We slightly generalize his analysis to also cover improved branching rules for k-SAT. The resulting deterministic algorithms have running times of $O^*(\varepsilon^{-\lambda_k})$ for some positive increasing sequence $(\lambda_k)_{k=2}^{\infty}$, where for instance $\lambda_3 \leq 2.27$. We demonstrate how similar techniques can be used for finding vertex covers and we give a deterministic algorithm running in time sublinear in ε^{-1} for instances of k-vertex cover with at least $\varepsilon\binom{n}{k}$ solutions and k bounded by some fraction of n.

In Section 3 we prove our main result, Theorem 7, which describes an algorithm for Sample-2-SAT that runs in expected time $O^*(\varepsilon^{-0.617})$. It therefore improves on the algorithm based on Wahlström's algorithm [18] when $\varepsilon = \Omega(0.708^n)$, or equivalently when F has $\Omega(1.415^n)$ satisfying assignments. We leave it as an open problem to decide whether sampling solutions to 3-CNF formulas can be done in time $O^*(\varepsilon^{-\delta})$ with $\delta < 1$ and discuss why the 2-CNF case does not generalize. In Proposition 8 we show how to solve 3-SAT in time $O(\varepsilon^{-0.936}(m+n))$ using similar ideas.

1.3 Notation

For a Boolean variable x we denote its *negation* by $\bar{x}$ and for a set V of Boolean variables let $\overline{V}$ be the set of negated variables. A *literal* is either a Boolean variable or its negation and in the former case we call the literal *positive* and in the latter we call it *negative*. We think of a *CNF formula*, or simply a *formula*, F over a variable set V as a set $F = \{C_1, C_2, \ldots, C_m\}$ of *clauses* where each clause $C_i \subset V \cup \overline{V}$ is a set of literals without both x and $\bar{x}$ in the same clause for any variable $x \in V$. By a k-CNF formula and by a $(\leq k)$-CNF we denote CNF formulas in which every clause has cardinality exactly k or at most k, respectively. We let $\mathrm{vbl}(F) \subseteq V$ denote the set of variables that appear in F either as a positive or negative literal. The *empty formula* is denoted by $\{\}$ and the *empty clause* by $\square$. An *assignment* to the variables in the formula F is a function $\alpha : V \to \{0,1\}$ and it is said to *satisfy* F if every clause $C \in F$ is satisfied, namely, if the clause contains a literal whose value is set to 1 under the assignment. A satisfying assignment is also called a *solution*. The empty formula is satisfied by any assignment to the variables and the empty clause by none. The set of all satisfying assignments of a formula F over V is denoted $\mathrm{sat}_V(F)$, and we omit the subscript V when it is clear from the context. A *partial assignment* to F is a function $\beta : W \to \{0,1\}$ with $W \subseteq V$ and we let $F^{[\beta]}$ be the formula over the variables $V \setminus W$ which is attained from F by removing each clause of F that is satisfied under β and then removing all literals assigned to 0 from the remaining clauses. If $u \in V \cup \overline{V}$ is a literal and $i \in \{0,1\}$

we let $F^{[u \mapsto i]}$ denote $F^{[\beta]}$ where β is the partial assignment that maps only u to i. By *unit clause reduction* we refer to the process of repeatedly setting variables to satisfy the unit clauses until finishing the process by exhausting the unit clauses or finding the empty clause.

All the logarithms are in base 2 unless noted otherwise.

2 Deterministic algorithms and Hirsch's method

In this section we consider Hirsch's method [6] for finding a satisfying assignment to a k-CNF formula, and extend the analysis to accommodate any branching rule.

We first briefly recall basic definitions on branching algorithms. A *complexity measure* μ is a function that assigns a nonnegative value $\mu(F)$ to every instance F of some particular problem. Given a problem and a complexity measure μ for it, we say that an algorithm correctly solving the problem is a *branching algorithm* (with respect to μ) if for every instance F the algorithm computes a list $(F_1, \ldots, F_t)$ of instances of the same problem, recursively solves the F_i's, and finally combines the results to solve F. Finding the list $(F_1, \ldots, F_t)$ and recursively solving each of them is called a *branching*. Letting $b_i = \mu(F) - \mu(F_i)$ we call the vector $(b_1, \ldots, b_t)$ the *branching vector* associated to the branching. Lastly, the *branching number* $\tau(b_1, \ldots, b_t)$ is defined as the smallest positive solution of the equation $\sum_{i=1}^{t} x^{-b_i} = 1$. If λ is the largest branching number of any possible branching in the algorithm and $T(F)$ is the time used to find the branching and to combine the results after the recursive calls, then the running time of the algorithm can be bounded by $O(T(F)\lambda^{\mu(F)})$.

Following Hirsch [6], we consider a *breadth-first* version of such a branching algorithm, taking a k-CNF Boolean formula F as input. We use the number of variables as a measure, and branch on partial assignments β_i, each fixing exactly b_i variables. The set Φ_ℓ in the algorithm below eventually contains the formulas constructed from input F after fixing exactly ℓ variables.

1. set $\ell \leftarrow 0$, $\Phi_0 \leftarrow \{F\}$, and $\Phi_\ell \leftarrow \emptyset$ for all $\ell > 0$.
2. if $\{\} \in \Phi_\ell$, then stop and return the so far fixed variables
3. for each $F \in \Phi_\ell$ such that $\square \notin F$:
   a. find a collection of t partial assignments of the form $\beta_i : W_i \to \{0,1\}$, where $W_i \subseteq \mathrm{vbl}(F)$
   b. for each $i \in [t]$:
      i. $\Phi_{\ell+b_i} \leftarrow \Phi_{\ell+b_i} \cup \{F^{[\beta_i]}\}$
4. $\ell \leftarrow \ell + 1$; if $\ell \leq n$ then go to step 2

For this algorithm to be correct, the partial assignments in 3a have to of course be chosen according to a correct branching rule. The complete collection Φ_ℓ can be seen as a collection of nodes of the search tree of the recursive algorithm, and is referred to as the ℓth *floor* of the tree. The following lemma holds [6].

▶ **Lemma 1.** *$|\Phi_\ell| \leq \lambda^\ell$ where λ is the maximum branching number of the recursion tree.*

The following result was proved by Hirsch in the special case of the simple Monien-Speckenmeyer algorithm [11], in which the branching vector was $(1, 2, \ldots, k)$. We generalize it to arbitrary branching vectors. The proof is left for the full version of this paper [2].

▶ **Theorem 2.** *Consider a k-CNF formula F with n variables and m clauses, and suppose it has at least $\varepsilon 2^n$ satisfying assignments. Then any breadth-first branching algorithm for k-SAT with maximum branching number $\lambda_k < 2$ runs in time $O^*(\varepsilon^{-B})$ on this instance, where $B := 1/(\log_{\lambda_k} 2 - 1)$.*

To get concrete bounds from Theorem 2 it remains to find good branching rules for k-SAT. The improved algorithm by Monien and Speckenmeyer [11] for k-SAT uses the notion of autarkies and the branching vectors appearing in the algorithm are (1) and $(1, 2, \ldots, k-1)$ of which the latter has the worse branching number. This directly yields the following result for $k = 3$.

▶ **Theorem 3.** *Given a 3-CNF formula F on n variables and an $\varepsilon > 0$ with the guarantee that $|\mathrm{sat}(F)| \geq \varepsilon 2^n$, one can find a satisfying assignment for F in deterministic time $O^*\left(\varepsilon^{-2.27}\right)$.*

2.1 Vertex cover

The technique we have seen is not unique to satisfiability but extend easily to known graph problems. As an example, we now consider the *vertex cover problem*: given a graph G and an integer k, does there exist a subset $S \in \binom{V(G)}{k}$ such that $\forall e \in E(G), e \cap S \neq \emptyset$? The optimization version consists of finding a smallest subset S satisfying the condition. We consider exact algorithms, hence the problem is equivalent to the maximum independent set problem (consider $V(G) \setminus S$). This is naturally related to the previous results on 2-SAT: the vertex cover problem can be cast as finding a minimum-weight satisfying assignment for a monotone 2-CNF formula.

We first briefly recall a standard algorithm for finding a minimum vertex cover in a graph G on n vertices, if one exists, in time $O^*(1.3803^n)$. First note that if the maximum degree of the graph is 2, then the problem can be solved in polynomial time. Otherwise, pick a vertex v of degree at least 3, and return the minimum of $1 + VC(G - v)$ and $VC(G - v - N(v))$, where VC are recursive calls, and $N(v)$ is the set of neighbors of v in G. The running time $T(n)$ obeys the recurrence $T(n) = T(n-1) + T(n-4)$, solving to the claimed bound. We can also analyze it with respect to the size k of the sought cover, yielding $T(k) = T(k-1) + T(k-3)$, solving to 1.4656^k. In the latter, we do not count the total number of vertices that are processed, but only those that are part of the solution. Hence we can distinguish the branching number λ related to the number of vertices processed and the branching number ρ related to the number of vertices included in the vertex cover (equivalently, the weight of the current partial assignment). In our case, we have $\rho < 1.4656$.

We now consider instances of the vertex cover problem in which we are promised that there are at least $\varepsilon\binom{n}{k}$ vertex covers. Given a branching algorithm, we can parse its search tree in breadth-first order, by associating with each node the number of vertices included in S so far (that is, the weight of the partial assignment). We define Φ_ℓ as the set of nodes with such value ℓ, and call it the ℓth floor. The following lemma is similar to Lemma 1.

▶ **Lemma 4.** $|\Phi_\ell| \leq \rho^\ell$.

After generating the ℓth floor Φ_ℓ, there are at most $\rho^\ell\binom{n-\ell}{k-\ell}$ remaining covers to check. If this is less than the total number of solutions of size k, we are done. The following statement gives an upper bound on the number of levels of the tree we need to parse. We leave the proof to the full version of this paper [2].

▶ **Lemma 5.** *Let $\ell^* := \ln(\frac{1}{\varepsilon})/\ln(\frac{n}{\rho k})$. Then for $k, n >> \ell^*$ and $k \leq n/\rho$, we have*

$$\rho^\ell \binom{n-\ell}{k-\ell} \geq \varepsilon \binom{n}{k} \Rightarrow \ell \leq \ell^*.$$

For n large enough, Lemma 5 implies that if $\ell > \ell^*$ then the number of remaining solutions is smaller than the promised number $\varepsilon\binom{n}{k}$, and either we have found one already, or greedily

completing any partial solution leads to a solution. Hence the running time is within a linear factor of ρ^{ℓ^*}, which simplifies as follows.

▶ **Theorem 6.** *Given a Vertex Cover instance composed of a graph G on n vertices, a number $k < n/\rho$, and an $\varepsilon > 0$ with the guarantee that G has at least $\varepsilon\binom{n}{k}$ vertex covers of size k, one can find such a vertex cover in deterministic time*

$$O^*\left(\varepsilon^{-\frac{\log\rho}{\log(\frac{n}{\rho k})}}\right),$$

where ρ is the branching number of an exact branching algorithm for k-vertex cover. In particular, this holds for $\rho = 1.4656$.

Note that the running time remains sublinear in $1/\varepsilon$ for all values of k such that $\frac{\log\rho}{\log(\frac{n}{\rho k})} < 1 \Leftrightarrow k < n/\rho^2$. Hence for those values of k, and in particular when $k = o(n)$, we have a deterministic algorithm for k-vertex cover whose complexity improves on the trivial sampling algorithm.

3 Randomized algorithms for Sample-2-SAT and for 3-SAT

In this section we present our algorithm for Sample-2-SAT with an expected running time of $O\left(\varepsilon^{-0.617}(m+n)\right)$ on 2-CNF formulas with more than ε fraction of satisfying assignments. The parameter ε does not need to be a constant and the algorithms can be easily modified so that they do not need to know ε in advance. Before stating and proving our main result we consider a warm-up algorithm that gives a weaker bound but already highlights some of the main ideas. In the end we discuss the complications of generalizing our method to Sample-3-SAT and see how to solve 3-SAT in expected time $O\left(\varepsilon^{-0.940}(m+n)\right)$ using similar techniques as for Sample-2-SAT.

Schmitt and Wanka [14] have used analogous ideas to approximately count the number of solutions in k-CNF formulas.

3.1 A warm-up algorithm for Sample-2-SAT

We will start with a warm-up algorithm that we then improve. Let F be a 2-CNF formula over the variable set V with $n := |V|$ and with m clauses. Let $S \subseteq F$ be a greedily chosen maximal set of variable disjoint clauses. We make the following remarks.

- Any satisfying full assignment for F must in particular satisfy S and is therefore an extension of one of the $3^{|S|}$ partial assignments to $\mathrm{vbl}(S)$ that satisfy all clauses in S.
- Because of maximality any partial assignment of the form $\alpha : \mathrm{vbl}(S) \to \{0,1\}$ has the property that $F^{[\alpha]}$ is a (≤ 1)-CNF.
- Counting and sampling of solutions of a (≤ 1)-CNF is easily done in linear time.

The set S allows us on one hand to do improved rejection sampling and on the other hand to device a branching based sampling. More concretely, consider the following two algorithms that use S.

1. Sample uniformly among all full assignments for F that satisfy all the clauses in S until finding one that satisfies F.
2. Go through all $3^{|S|}$ partial assignments $\alpha : \mathrm{vbl}(S) \to \{0,1\}$ that satisfy S and for each α compute $A_\alpha := |\mathrm{sat}_{V\setminus\mathrm{vbl}(S)}(F^{[\alpha]})|$, i.e., the number of satisfying assignments in $F^{[\alpha]}$. Then $A := \sum_\alpha A_\alpha$ is the number of satisfying assignments in F. Draw one partial

assignment α^* at random so that $\Pr(\alpha^* = \alpha) = A_\alpha/A$. For the remaining variables choose an assignment $\beta^* : V \setminus \mathrm{vbl}(S) \to \{0,1\}$ uniformly among all assignments satisfying $F^{[\alpha^*]}$. Output the full assignment which when restricted to $\mathrm{vbl}(S)$ is α^* and when restricted to $V \setminus \mathrm{vbl}(S)$ is β^*.

The correctness of the first algorithm is clear since any assignment satisfying F must also satisfy S. One sample can also be drawn in linear time. Because the clauses of S are variable disjoint, the pool of assignments we are sampling from has $(\frac{3}{4})^{|S|}2^n$ assignments and it contains all the at least $\varepsilon 2^n$ satisfying assignments. Therefore the probability of one sample being satisfying is at least $(\frac{4}{3})^{|S|}\varepsilon$, implying an expected runtime of $O\left(\varepsilon^{-1}(\frac{3}{4})^{|S|}(m+n)\right)$ for the first algorithm.

We need the second algorithm to balance the first one when $|S|$ is small. For the correctness we observe that the partial assignments α partition the solution space in the sense that $A = \sum_\alpha A_\alpha = |\mathrm{sat}_V(F)|$ and a simple calculation shows that the output distribution is uniform over $\mathrm{sat}_V(F)$. With the remarks made before the algorithm description we conclude that the runtime of the second algorithm is $O(3^{|S|}(m+n))$. If space is a concern, the sampling of α^* can be done in linear space without storing the numbers A_α as follows: Sample a uniform number r from $\{1,\dots,A\}$ and go through the partial assignments α again in the same order and output the first α for which the total number of assignments counted up to that point reaches at least r.

For any given S we can choose the better of the two algorithms which gives an expected runtime guarantee of

$$O\left(\max_{|S|}\left\{3^{|S|}, \varepsilon^{-1}\left(\frac{3}{4}\right)^{|S|}\right\}\cdot(m+n)\right) = O\left(\varepsilon^{-\log_4 3}(m+n)\right) \tag{1}$$

where $\log_4 3 < 0.793$. Note that we do not need to know ε in advance to get the same runtime guarantee as we can simulate running both of the algorithms in parallel until one finishes.

3.2 A faster algorithm for Sample-2-SAT

In the warm-up algorithm we used the set S on the one hand to reduce the size of the set of assignments we are sampling from and on the other hand we used it as a small size *hitting set* for the clauses in F: every clause in F contained at least one variable from $\mathrm{vbl}(S)$. To improve we will do two things. Firstly, we will consider more complicated independent structures that improve on both aspects above, giving us both a smaller size sampling pool and a better hitting set. Secondly, we notice that it is not necessary to always use an exact hitting set in the counting procedure but an "almost hitting set" is enough. Namely, if some small set of variables hits almost all clauses we can count the number of solutions to the remaining relatively small (≤ 2)-SAT with a good exponential time algorithm for #2-SAT.

We introduce first some notation. For $i \in \mathbb{N}$ we call a set of clauses S an *i-star* if $|S| = i$ and if there exists a variable x such that for any pair of distinct clauses $C, D \in S$ we have $\{x\} = \mathrm{vbl}(C) \cap \mathrm{vbl}(D)$. A *star* is an i-star for some i. For $i \geq 2$ we call the variable x the *center* of the star and any other variable is called a *leaf*. For 1-stars we consider both of the variables as centers and neither of them as leaves. A star is called *monotone* if the center appears as the same literal in every clause of the star. We call a set T of exactly three clauses a *triangle* if every 2-element subset of T is a star and T is not itself a star. Finally, we call a family $\mathcal{M}$ of CNF formulas *independent* if no two formulas in $\mathcal{M}$ share common variables.

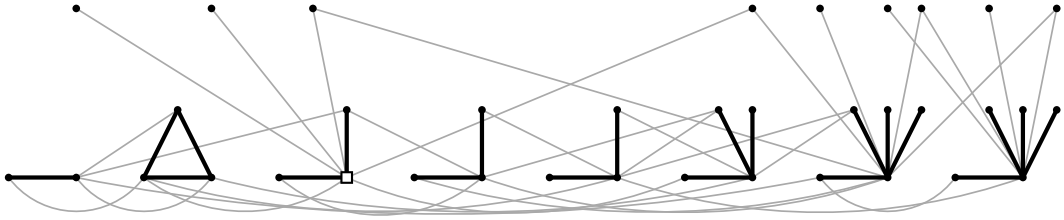

Figure 1 A possible construction of $\mathcal{M}_4$ for a formula F that is displayed as a graph with the variables as vertices and edges between variables appearing in the same clause. The subformulas of F that make up $\mathcal{M}_4$ are given by the components defined by the black bold edges. The edges that form up $\mathcal{M}_0$ are the horizontal black bold edges. There is one non-monotone 2-star in $\mathcal{M}_4$ and it is denoted by the square center vertex.

▶ **Theorem 7.** *Let F be a 2-CNF formula on n variables and m clauses and let $\varepsilon > 0$ be such that $|\mathrm{sat}(F)| \geq \varepsilon 2^n$. A uniformly random satisfying assignment for F can be found in expected time $O\left(\varepsilon^{-\delta}(m+n)\right)$ where $\delta < 0.617$.*

Proof. Let V be the variable set of F and let $k \geq 2$ be a constant independent of ε that we fix later. We start by constructing a sequence $(\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k)$ of $k+1$ independent families of formulas where every family consists of subformulas of F.

Let $\mathcal{M}_0$ be any independent 1-maximal family of 1-stars (clauses) in F. That is, in addition to maximality we require further that there is no clause in the family whose removal would allow the addition of two clauses in its place. We can find $\mathcal{M}_0$ with a greedy algorithm in linear time[1].

To construct $\mathcal{M}_1$ from $\mathcal{M}_0$, we add clauses of F to the 1-stars of $\mathcal{M}_0$ greedily to update them into non-monotone 2-stars or triangles while maintaining independence. As a result $\mathcal{M}_1$ is an independent family of subformulas of F that consists of 1-stars, non-monotone 2-stars, and triangles and no 1-star can be turned into the other two types by adding clauses of F to it without revoking independence.

For $i = 2, \ldots, k$ we construct $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ by greedily adding clauses of F to the monotone $(i-1)$-stars to turn them into monotone i-stars while ensuring independence. Since k is a constant, and all since greedily adding clauses can be done in linear time, the total time taken to construct the families is $O(m+n)$. An example of $\mathcal{M}_4$ can be seen in Figure 1. We describe the structural properties of the families later in the proof.

Analogously to the warm-up algorithm in the previous section we describe two different algorithms that both make use of the independent families we have constructed and that complement each other in terms of their running times. The second algorithm describes in fact k different algorithms, determined by the choice of a parameter $\ell \in \{1, \ldots, k\}$. For each $i = 1, \ldots, k$ we let s_i denote the number of monotone i-stars in $\mathcal{M}_k$. By construction the parameter $r_i := \sum_{j=i}^{k} s_j$ then denotes the number of monotone i-stars in M_i. We further let t be the number of triangles and q be the number of non-monotone 2-stars in $\mathcal{M}_k$, and therefore in every $\mathcal{M}_i$ with $i = 1, \ldots, k$. The two algorithms we consider are:

1. Sample uniformly among all full assignments for F that satisfy all the clauses in $\mathcal{M}_k$ until finding one that satisfies F.

[1] This is equivalent to finding a 1-maximal matching in a graph: first find a maximal matching and then find a maximal set of independent augmenting paths of length 3 and augment them.

2. Fix $\ell \in \{1, \dots k\}$. Define further the variable set $W := \mathrm{vbl}(\mathcal{M}_\ell)$ and let $W' \subseteq W$ be the set of variables of $\mathcal{M}_\ell$ that appear in a clause of F that has exactly one variable of $\mathcal{M}_\ell$ in them. Go through all $2^{|W'|}$ partial assignments $\alpha : W' \to \{0,1\}$ and compute $A_\alpha := |\mathrm{sat}_{V \setminus W'}(F^{[\alpha]})|$ by using Wahlström's #2-SAT algorithm [18]. Let $A := \sum_\alpha A_\alpha$ and choose one partial assignment α^* at random so that $\Pr(\alpha^* = \alpha) = A_\alpha / A$. For the remaining variables choose an assignment $\beta^* : V \setminus W' \to \{0,1\}$ uniformly among all assignments satisfying $F^{[\alpha^*]}$. This can be done by branching on a variable, using Wahlström's algorithm to count the number of assignments in the two branches, flipping a biased coin weighed by the counts to decide on the branch and repeating the same on the resulting formula until all variables have been set. Output the full assignment which when restricted to W' is α^* and when restricted to $V \setminus W'$ is β^*.

The correctness analysis for both of these two algorithms is essentially the same as in our warm-up in Section 3.1 and it remains to discuss the running times.

Starting with the first algorithm we note that the stars and triangles in $\mathcal{M}_k$ have constant size so the sampling of an assignment can be done in linear time in each iteration. Out of the 2^{i+1} possible assignments to the variables in any monotone i-star it can be easily checked that $2^i + 1$ satisfy all the clauses in the star. Both for a triangle or for a non-monotone 2-star there are 8 possible assignments out of which at most 4 are satisfying. Therefore from the independence of $\mathcal{M}_k$ we know that there are at most

$$2^{-t-q} \prod_{i=1}^{k} \left(\frac{2^i + 1}{2^{i+1}} \right)^{s_i} 2^n \tag{2}$$

full assignments to the variables in F that satisfy everything in $\mathcal{M}_k$. Since F has at least $\varepsilon 2^n$ satisfying assignments and the size of the universe we are sampling from is given by (2) we conclude that the first algorithm takes expected time

$$O\left(\varepsilon^{-1} 2^{-t-q} \prod_{i=1}^{k} \left(\frac{2^i + 1}{2^{i+1}} \right)^{s_i} (m+n) \right) \tag{3}$$

until returning a uniform satisfying assignment.

Consider now the runtime of the second algorithm. This is the more intricate part of the analysis and we will make use of the structure of the families that we have set up. It may be helpful to consider Figure 1. Let $F' \in \mathcal{M}_\ell$ be one of the subformulas in the family $\mathcal{M}_\ell$. We claim that $|\mathrm{vbl}(F') \cap W'| \leq 1$ and that if $\mathrm{vbl}(F') \cap W' = \{x\}$, then F' is either an ℓ-star or a non-monotone 2-star and x is the center of the star. Towards showing the claim let $\{u, v\}$ be a clause with $\mathrm{vbl}(u) \in W$ and $\mathrm{vbl}(v) \in V \setminus W$ so that $\{u, v\}$ is a witness for $\mathrm{vbl}(u) \in W'$. If $\mathrm{vbl}(u)$ was a leaf of a star of $\mathcal{M}_\ell$, then we could have made $\mathcal{M}_0$ larger which would contradict the 1-maximality when $\mathrm{vbl}(u) \in \mathrm{vbl}(\mathcal{M}_0)$ or just maximality in the case of $\mathrm{vbl}(u) \notin \mathrm{vbl}(\mathcal{M}_0)$. For the same reasons the variable $\mathrm{vbl}(u)$ can not appear in any triangle. For any $j < \ell$ the variable $\mathrm{vbl}(u)$ can also not be the center of a j-star as otherwise we would have updated that star into a monotone $(j+1)$-star when constructing $\mathcal{M}_{j+1}$ or we would have created a non-monotone 2-star already in the beginning while constructing $\mathcal{M}_1$. The options for $\mathrm{vbl}(u)$ that remain are the centers of ℓ-stars and the centers of the non-monotone 2-stars. In the case of $\ell = 1$ we still have to argue that at most one center may appear in W'. If both of the centers appeared in W', it would either violate the 1-maximality of $\mathcal{M}_0$ or we could have turned the 1-star into a triangle which proves the claim. Therefore we have the bound $|W'| \leq r_\ell + q$.

We can observe from the argumentation above that if $\alpha : W' \to \{0,1\}$ is a partial assignment for F, then doing unit clause reduction on the formula $F^{[\alpha]}$ results in a 2-CNF formula over some variable set $W_\alpha \subseteq W \setminus W'$. Computing A_α with Wahlström's algorithm takes time $O(c^{|W_\alpha|})$ [18]. Therefore we want to bound $|W_\alpha|$ as tightly as possible. If the assignment α sets the center literal of a monotone ℓ-star to 0, then the values of the ℓ remaining variables in the star are determined and will be set to their required values with unit clause reduction. For a non-monotone 2-star either assignment of the center will force the value of one of the leaves and one leaf stays undetermined. If α sets i of the r_ℓ literals in the centers of the monotone ℓ-stars to 0 we get the bound

$$|W_\alpha| \leq q + 3t + \ell(r_\ell - i) + \sum_{j=1}^{\ell-1}(j+1)s_j. \tag{4}$$

Among the assignments α that we consider there are $\binom{r_\ell}{i}2^q$ different ones that set i of the central literals of the monotone ℓ-stars to 0. Using formula (4) we conclude that the runtime cost of going over the assignments α and computing the numbers A_α is

$$\begin{aligned} &O\left(\sum_{i=0}^{r_\ell}\binom{r_\ell}{i}2^q \cdot c^{q+3t+\ell(r_\ell-i)+\sum_{j=1}^{\ell-1}(j+1)s_j} \cdot (m+n)\right) \\ &= O\left(c^{3t}(2c)^q\left(1+c^\ell\right)^{r_\ell}\left[\prod_{j=1}^{\ell-1}c^{(j+1)s_j}\right]\cdot(m+n)\right) \end{aligned} \tag{5}$$

where we used the binomial theorem. We can again use the same trick as in the warm-up algorithm to sample α^* without storing all the values of A_α to keep the space requirement linear. The running time of finding β^* with the branching procedure takes time $O(c^{|W_{\alpha^*}|}|W_{\alpha^*}| + (m+n))$ which is subsumed by (5).

We have now one algorithm with running time given by (3) and for any $\ell \in \{1,\ldots,k\}$ we have an algorithm with running time given by (5). Given the sequence $(\mathcal{M}_1,\ldots,\mathcal{M}_k)$ we choose the algorithm with the best runtime. To find a worst case upper bound on the runtime we look for the runtime in the form

$$O\left(\varepsilon^{-\delta}(m+n)\right) \tag{6}$$

and compute the nonnegative parameters $s_1,\ldots,s_k;t$ and q that maximize the minimum of the different runtimes. Write $\sigma_i := s_i/\log\frac{1}{\varepsilon}, \tau := t/\log\frac{1}{\varepsilon}, \rho := q/\log\frac{1}{\varepsilon}$. By taking logarithms of the runtimes (3), (5) and (6) we can write the problem of finding δ and the worst case parameters σ_i, τ, ρ as the linear program

$$\begin{array}{lrcl} & \max\limits_{\delta,\sigma_i,\tau,\rho} \quad \delta & & \\ \text{s.t.} \quad -\tau-\rho+\sum_{i=1}^{k}\sigma_i\log\left(\frac{2^i+1}{2^{i+1}}\right) & \geq & \delta-1 & \\ 3\tau\log c+\rho\log(2c)+\sum_{i=1}^{\ell-1}\sigma_i\log\left(c^{i+1}\right)+\sum_{i=\ell}^{k}\sigma_i\log\left(1+c^\ell\right) & \geq & \delta & \text{for all } \ell=1,\ldots,k \\ \delta,\sigma_i,\tau,\rho & \geq & 0 & \text{for all } i=1,\ldots,k\ . \end{array}$$

It turns out that we only need to consider $k = 7$ due to the fact that $c^{j+1} > 1 + c^j$ in the integers when $j \geq 7$ which implies that the running time for higher values of k no longer improves. For $k = 7$ the linear program has in the optimum $\delta < 0.61618$. The approximate values of the other variables in the optimum are $\sigma_1 \approx 0.131, \sigma_2 \approx 0.127, , \sigma_3 \approx 0.111, \sigma_4 \approx 0.084, \sigma_5 \approx 0.051, \sigma_6 \approx 0.022, \sigma_7 \approx 0.004$ and exact values of $\tau = 0$ and $\rho = 0$. This finishes the proof. ◀

We attempted to improve the analysis by constructing families that do not consist only of stars and triangles but the runtimes we achieved were not better. In some sense stars seem particularly good for the efficient use of Wahlström's #2-SAT algorithm as a subroutine because the set W' is not too big. We also note that while we could consider adding the option of choosing $\ell = 0$ in the second algorithm, it is easily verified that choosing $\ell = 1$ instead gives a better performance.

3.3 A randomized algorithm for 3-SAT

One could say that our Sample-2-SAT algorithm works because counting and sampling solutions for a (≤ 1)-CNF is trivial. Direct generalizations of our method to Sample-3-SAT do not work because the same is not true for (≤ 2)-CNF formulas. Instead of solving Sample-3-SAT we apply our method for 3-SAT.

▶ **Proposition 8.** *Let F be a 3-CNF formula on n variables and m clauses and let $\varepsilon > 0$ be such that $|\mathrm{sat}(F)| \geq \varepsilon 2^n$. A satisfying assignment for F can be found in expected time $O\left(\varepsilon^{-\log_8 7}(m+n)\right)$.*

Proof. Let S be a maximal set of variable disjoint clauses in F. Either sample among those assignments that satisfy S until finding a satisfying assignment or go through all the $7^{|S|}$ partial assignments to $\mathrm{vbl}(S)$ and check the satisfiability of the resulting (≤ 2)-CNF.

Checking through the partial assignments takes time $O(7^{|S|} \cdot (m+n))$ because each of the $7^{|S|}$ instances of (≤ 2)-SAT can be solved in linear time [1]. The rejections sampling takes expected time $O\left(\varepsilon^{-1}\left(\frac{7}{8}\right)^{|S|}(m+n)\right)$ because we are sampling from a pool of $\left(\frac{7}{8}\right)^{|S|}2^n$ assignments that contain all the at least $\varepsilon 2^n$ many satisfying assignments. Choosing always the better of the two methods, depending on $|S|$, gives a worst case running time of $O\left(\varepsilon^{-\log_8 7}(m+n)\right)$. ◀

Proposition 8 gives an algorithm that works for any ε, but there exist better algorithms for certain ranges of ε. The PPSZ algorithm for 3-SAT runs in expected time $O(1.308^n)$ [5] which is faster in the case that $\varepsilon = O(0.750^n)$. It is also possible to analyze Schöning's algorithm [15] for 3-SAT to get a dependence on ε by using an isoperimetric inequality for the hypercube by Frankl and Füredi [4]. The runtime guarantee that results is $O\left(\left(\frac{4}{3} \cdot 2^{-H^{-1}(\delta)}\right)^n\right)$ in expectation where δ is the solution to $\varepsilon = 2^{(\delta-1)n}$ and where $H : (0, 1/2] \to (0, 1]$ is the bijective *binary entropy function* defined by $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$. We will include a proof in the full version of this paper [2]. The range where Schöning's algorithm is better than Proposition 8 is when $\varepsilon = O(0.929^n)$.

4 Conclusion

An interesting open problem is whether Sample-3-SAT can be solved time $O^*\left(\varepsilon^{-\delta}\right)$ for some $\delta < 1$. Similarly, can we achieve such a running time for 3-SAT with a deterministic algorithm?

We also believe that parameterizing by the number of solutions should be a fruitful approach to other problems besides satisfiability or vertex cover.

Acknowledgments. We would like to thank Noga Alon and József Solymosi for discussions on the problem. We also thank the reviewers of IPEC 2017 for valuable remarks that improved the exposition.

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(3):121–123, 1979.
2. Jean Cardinal, Jerri Nummenpalo, and Emo Welzl. Solving and Sampling with Many Solutions: Satisfiability and Other Hard Problems. *ArXiv e-prints*, 2017. `arXiv:1708.01122`.
3. Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. Improved pseudorandom generators for depth 2 circuits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX, and 14th International Workshop, RANDOM*, pages 504–517, 2010.
4. Peter Frankl and Zoltán Füredi. A short proof for a theorem of Harper about hamming-spheres. *Discrete Mathematics*, 34(3):311–313, 1981.
5. Timon Hertli. 3-SAT faster and simpler—unique-SAT bounds for PPSZ hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014.
6. Edward A Hirsch. A fast deterministic algorithm for formulas that have many satisfying assignments. *Logic Journal of IGPL*, 6(1):59–71, 1998.
7. Mark R Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Springer Science & Business Media, 2003.
8. Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.
9. Daniel M Kane and Osamu Watanabe. A short implicant of cnfs with relatively many satisfying assignments. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 20, page 176, 2013.
10. Kuldeep S Meel, Moshe Y Vardi, Supratik Chakraborty, Daniel J Fremont, Sanjit A Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. Constrained sampling and counting: Universal hashing meets sat solving. In *AAAI Workshop: Beyond NP*, 2016.
11. Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.
12. Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan s Marcu, and Gil Shurek. Constraint-based random stimuli generation for hardware verification. *AI magazine*, 28(3):13, 2007.
13. Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005.
14. Manuel Schmitt and Rolf Wanka. Exploiting independent subformulas: A faster approximation scheme for #k-SAT. *Information Processing Letters*, 113(9):337–344, 2013.
15. Uwe Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.
16. Rocco Servedio and Li-Yang Tan. Deterministic search for CNF satisfying assignments in almost polynomial time. Unpublished manuscript, 2016.
17. Luca Trevisan. A note on approximate counting for k-DNF. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 417–425. Springer, 2004.
18. Magnus Wahlström. A tighter bound for counting max-weight solutions to 2SAT instances. In *International Workshop on Parameterized and Exact Computation*, pages 202–213. Springer, 2008.