An Approximation Algorithm for #k-SAT

Marc Thurley

Centre de Recerca Matemàtica, Bellaterra, Spain. **Supported by Marie Curie Intra-European Fellowship 271959**

Abstract

We present a simple randomized algorithm that approximates the number of satisfying assignments of Boolean formulas in conjunctive normal form. To the best of our knowledge this is the first algorithm which approximates #k-SAT for any $k \geq 3$ within a running time that is not only non-trivial, but also significantly better than that of the currently fastest exact algorithms for the problem. More precisely, our algorithm is a randomized approximation scheme whose running time depends polynomially on the error tolerance and is mildly exponential in the number n of variables of the input formula. For example, even stipulating sub-exponentially small error tolerance, the number of solutions to 3-CNF input formulas can be approximated in time $O(1.5366^n)$. For 4-CNF input the bound increases to $O(1.6155^n)$.

We further show how to obtain upper and lower bounds on the number of solutions to a CNF formula in a controllable way. Relaxing the requirements on the quality of the approximation, on k-CNF input we obtain significantly reduced running times in comparison to the above bounds.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases #k-SAT, approximate counting, exponential algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.78

1 Introduction

The design and analysis of algorithms that determine the satisfiability or count the models of k-CNF formulas has quite some tradition. In the case of the satisfiability problem the earliest algorithm with a worst case running time which is significantly better than the trivial $\text{poly}(n)2^n$ bound dates back to at least 1985 [14]. The time bounds have improved gradually over the years with most recent results (only a few of which are [11, 18, 10, 9]) being analyses of randomized algorithms that have been obtained from either Schöning's algorithm [20], the algorithm of Paturi, Pudlák, Saks, and Zane [17], or a combination of both. The currently fastest algorithm for 3-SAT by Hertli [9] running in time $O(1.30704^n)$ falls roughly into the second category. The corresponding counting problems have seen similar improvements [3, 26, 2, 13, 4, 24] over the trivial time bound, with the current best worst case running time for #3-SAT being $O(1.6423^n)$ obtained by Kutzkov [13].

Quite surprisingly, however, the situation is completely different for the *approximation* of #k-SAT. To the best of my knowledge not even small improvements over the trivial worst case time bound are known.[1] This, however, does not seem to be due to a general lack of interest in the problem itself. From a complexity theoretical point of view, for example, several already classic papers [22, 23, 19] study questions closely related to direct algorithmic problems in #k-SAT approximation. In particular Valiant and Vazirani [23] bound the

[1] Disregarding, of course the pathological fact that exact methods can be interpreted as approximation algorithms, as well.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 78–87

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity of the approximation problem from above by reduction to SAT, and hence settle its complexity in a certain sense.

While theoretical results on the approximation of #k-SAT are rather old, there are several heuristic approaches to the problem, that have all appeared only fairly recently. Motivated by questions of practicability, these results focus on methods that can be shown empirically to work well, while sacrificing some (at least theoretically) desirable properties. That is, some of these approaches yield approximations without any guarantee on the quality of the approximation [25, 5]. Others yield reliable lower and upper bounds [8, 7, 12] which, in certain cases, are surprisingly good although generally their quality is rather low. In particular, this line of work does not provide rigorous bounds on running times and neither does it yield rigorous quality estimates of the approximation computed.

With regard to the above results, the lack of competitive worst case bounds for #k-SAT approximation algorithms seems to be due to several factors. First of all the exact algorithms found in the literature and their analyses do not seem to carry over easily to the approximation problem. Secondly, complexity theoretical insights are usually not considered applicable in the context of designing fast exponential algorithms. An example is the technique of Valiant and Vazirani which leads to a significant blow up in formula size. And thirdly, it is not clear which of the known algorithmic ideas used in the heuristic approaches could at least in principle show a good worst case behavior.

1.1 Contributions

In this paper we will see that one can indeed not only improve upon the trivial worst-case time bound mentioned above. But the algorithm we will present also provides arbitrarily good precision in significantly less time than known exact methods. To be more precise, the algorithm we present is a *randomized approximation scheme* for #k-SAT for every $k \geq 3$. Given a freely adjustable error tolerance $\epsilon > 0$, randomized approximation schemes produce an output that is within a factor of e^{ϵ} of the number $\#F$ of solutions of some input formula F.

We obtain the following main result, which we state here only for $k = 3$ and $k = 4$. The general result will be discussed in Section 3.

▶ Theorem 1.1. There is a randomized approximation scheme running in time $O(\epsilon^{-2} \cdot 1.5366^n)$ for #3-SAT and in time $O(\epsilon^{-2} \cdot 1.6155^n)$ for #4-SAT.

For #3-SAT this algorithm is already significantly faster than the currently fastest exact algorithm from [13] which runs in time $O(1.6423^n)$. For #4-SAT the benefit of approximation is even more impressive, as the best bound for exact methods is still the $O(1.9275^n)$ bound of the basically identical algorithms of Dubois [3] and Zhang [26].

We will see that the algorithm of Theorem 1.1 is not complicated and monolithic like the branching algorithms usually employed in exact counting results. But it is actually a combination of two very simple and very different algorithms. The main reason for considering this combination relies on two pieces of intuition. On the one hand, if a formula has few solutions, then it is not too bad an idea to compute their number by simply enumerating them. On the other hand, if a formula has many solutions, then a quite trivial sampling algorithm should yield good results.

Observe that the result of Theorem 1.1 can already be used to compute e.g. $\#F$ exactly in time $O(1.5366^n)$ for any 3-CNF formula which has only a sub-exponential number of solutions. To achieve this we only have to set ϵ appropriately. However, we shall see below, that this can also be achieved in significantly less time. Motivated by the heuristic results on the

approximation of #k-SAT described above, we also study the effect of weaker requirements on the approximation bounds. It seems, of course, perfectly reasonable to assume that weaker bounds should come at the benefit of dramatically improved running time bounds. We will therefore show that this is the case. With respect to lower bounds we obtain:

▶ Theorem 1.2 (Lower Bound Algorithm). There is a randomized algorithm which, on input a 3-CNF formula F on n variables and a natural number N, performs the following in time $O(N^{0.614} \cdot 1.30704^n)$:

- If $\#F > N$ it reports this with probability at least 3/4.
- If $\#F \leq N$ then with probability at least 3/4 it reports this and outputs the correct value $\#F$.

Furthermore, there is a deterministic algorithm solving this task in time $O(N^{0.585} \cdot 1.3334^n)$.

This lower bound algorithm will in fact be used in the proof of Theorem 1.1 and relies on the above observation that we can simply use a SAT algorithm for enumerating all solutions provided the input formula has only few. The time bounds mentioned thus arise from the SAT algorithms used – the randomized 3-SAT algorithm by Hertli [9] and the deterministic one of Moser and Scheder [15] (which is in fact a derandomized version of Schöning's algorithm).

To obtain upper bounds, on the other hand, we cannot use the high-solution part of Theorem 1.1. But, although it might seem unreasonable to expect that this would yield a competitive running time, we can use an algorithm based on the bisection technique of Valiant and Vazirani [23]. Interestingly an algorithm based on Valiant and Vazirani's technique has been used already in the heuristic result of [8]. Their approach, however, is quite different from ours and does not have a good worst-case behavior.

By systematically augmenting the input formula F with randomly chosen GF(2)-linear constraints, the bisection technique makes it possible to approximate $\#F$ by determining satisfiability of the augmented formulas. The main difference of our approach to this classical scheme lies in the observation that it is more reasonable for our purposes to work directly with the system of linear equations obtained, instead of encoding it into k-CNF. In this way we obtain the running time bounds which are valid even for general CNF input formulas.

▶ Theorem 1.3 (Upper Bound Algorithm). There is an algorithm, which on input a CNF formula F on n variables and an integer $\mu \leq n$ takes time $O^*(2^{n-\mu})$ and performs the following with probability at least 2/3:

It outputs a number $u \geq \mu$ such that $U := 2^{u+3} \geq \#F$. If furthermore $u > \mu$ then 2^u is a 16-approximation of $\#F$.

Remark

This algorithm will actually work for upper bounding $|S|$ for any set $S \subseteq \{0,1\}^n$ with a polynomial membership test. However, as this is a trivial consequence of the proof, we consider only the case that the input is a CNF formula.

Moreover, we do not particularly focus on improving the approximation ratio mentioned in Theorem 1.3. Such an improvement is in fact unnecessary if we want to use this algorithm to design a randomized approximation scheme: We can combine the above algorithm with that of Theorem 1.2 to obtain a 16-approximation algorithm for e.g. #3-SAT which runs (up to a polynomial factor) within the same time bound as that stated in Theorem 1.1. This algorithm can then be plugged into a Markov chain by Jerrum and Sinclair [21] to boost the quality of approximation. This yields a $(1 + \frac{1}{\mathrm{poly}(n)})$-approximation algorithm incurring only a polynomial overhead in the computation. Thus we have a second, although more complicated, algorithm that satisfies the claim of Theorem 1.1.

2 Preliminaries

For a CNF formula F, let $\mathrm{sat}(F)$ be the set of its solutions and $\#F = |\mathrm{sat}(F)|$. We shall always use n to denote the number of variables of a CNF formula under consideration. A *randomized α-approximation algorithm* $\mathbb{A}$ for $\#k$-SAT outputs, on input a k-CNF formula F, a number $\mathbb{A}(F)$ such that

$$\Pr\left[\alpha^{-1}\#F \leq \mathbb{A}(F) \leq \alpha\#F\right] \geq p. \tag{1}$$

Here p is some constant, independent of the input and strictly larger than[2] $1/2$. A *randomized approximation scheme* for $\#k$-SAT, is then an algorithm which on input F and a natural number ϵ^{-1} behaves like a randomized e^{ϵ}-approximation algorithm.

We use the notation $x^1 = x$ and $x^0 = \bar{x}$. For a clause C, a variable x, and a truth value $a \in \{0,1\}$, the *restriction* of C on $x = a$ is the constant $\mathbf{1}$ if the literal x^a belongs to C, and $C \setminus \{x^{1-a}\}$ otherwise. We write $C|_{x=a}$ for the restriction of C on $x = a$. A *partial assignment* is a sequence of assignments $(x_1 = a_1, \ldots, x_r = a_r)$ with all variables distinct. Let α be a partial assignment. We will use the notation $\alpha \cup (x = a)$ to denote the assignment $(x_1 = a_1, \ldots, x_r = a_r, x = a)$. If C is a clause, we let $C|_\alpha$ be the result of applying the restrictions $x_1 = a_1, \ldots, x_r = a_r$ to C. Clearly the order of application does not matter. If F is a CNF formula, we let $F|_\alpha$ denote the result of applying the restriction α to each clause in F, and removing the resulting $\mathbf{1}$'s. We call $F|_\alpha$ the *residual* formula.

As we will use the algorithm of Paturi, Pudlák, Saks, and Zane [17] and a very recent paper by Hertli [9], we need the constant

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

3 The Algorithm

We are now able to state the main result in full detail.

▶ Theorem 3.1. For $k \geq 3$, $\#k$-SAT has a randomized approximation scheme running in time[3]

$$O^*\left(\epsilon^{-2} \cdot 2^{n\left(\frac{k-1}{k-1+\mu_k}\right)}\right).$$

As already outlined, the randomized approximation scheme of Theorem 3.1 is a combination of two different algorithms. We will discuss the algorithm for the case of few solutions now. The case of many solutions will be treated afterwards in Section 3.2.

3.1 Formulas with few solutions

For formulas with few solutions we will now present an algorithm relying on a simple enumeration of solutions by using a k-SAT algorithm as a subroutine. This will also prove Theorem 1.2.

[2] In the literature, usually either the value $p = 3/4$ or a further parameter δ such that $p = 1 - \delta$ seems to be common. However, it is well-known that all of these can be translated into each other with only polynomial overhead.

[3] We use the O^* notation to suppress factors sub-exponential in n.

▶ **Lemma 1.** *Let F be a k-CNF formula on n variables and let $\mathbb{A}$ be an algorithm performing the following task in time $O^*(2^{\beta_k n})$. If F is satisfiable, with probability at least $3/4$ it outputs a solution to F. If F is unsatisfiable, it reports this correctly.*

Then, there is a algorithm, which on input F and a natural number N, takes time $O^*(2^{\beta_k n} N^{(1-\beta_k)})$, and performs the following:

- *If $\#F > N$ it reports this with probability at least $3/4$.*
- *If $\#F \leq N$ then with probability at least $3/4$ it reports this and outputs the correct value $\#F$.*

Furthermore, if the algorithm reports $\#F > N$ then this holds with certainty.

Theorem 1.2 follows directly from this lemma by using the randomized 3-SAT algorithm of Hertli [9] which has $\beta_3 = 0.3864$. For the claim about the deterministic algorithm we use the result of Moser and Scheder [15], with $\beta_3 = 0.4151$.

In the proof of the above lemma, we will use the following fact which is very easily proven.

▶ **Lemma 2.** *A rooted tree with N leaves and depth (i.e. max root to leaf-distance) n has at most $n \cdot N$ vertices in total.*

Proof of Lemma 1. Note first, that by a standard trick we can boost the success probability of $\mathbb{A}$. Assume that, as provided by the statement of the lemma, we have error probability at most $1 - p \leq 1/4$. Then the probability of erring in M independent repetitions is at most $(1-p)^M \leq e^{-pM}$. Call the boosted version of this algorithm $\mathbb{A}^*$.

We shall fix a good value for M. Below we will see that algorithm $\mathbb{A}^*$ will be queried a number $O(nN)$ of times, for some $N \leq 2^n$, each time on a formula of at most n variables. The probability that the algorithm errs in any of these queries is at most $nN \cdot e^{-pM}$. So choosing M within a constant factor of $\log nN$ (which is polynomial in n) allows us to condition on the SAT algorithm not erring in any of the $O(nN)$ queries. The probability of that latter event is close to 1. And as this is the only possible source of failure of the algorithm, we will easily achieve a success probability of $3/4$ in the end.

The algorithm

Check if F is satisfiable, using $\mathbb{A}^*$, and if so, perform the following. Inductively, construct a search tree associated with partial assignments α, such that $F|_\alpha$ is satisfiable. For a leaf in the current search tree associated with some assignment α, choose a variable x from $F|_\alpha$ and check $F|_{\alpha \cup (x=0)}$ and $F|_{\alpha \cup (x=1)}$ for satisfiability using the algorithm $\mathbb{A}^*$. For each of the satisfiable restrictions add a new child to the current leaf in the search tree. We stop the algorithm, if it has N leaves, or if it has found all of the solutions of F. Traversing this tree, e.g. in a depth first manner we can implement this procedure in polynomial space (not taking the space needs of $\mathbb{A}^*$ into account).

Time

Consider the search tree this algorithm produces. As it has at most N leaves, and depth at most n, we have (recall Lemma 2) at most nN nodes overall in the tree.

Observe first, that at each node we perform at most 2 queries to the SAT algorithm $\mathbb{A}^*$. A node of level ℓ in the tree incurs queries taking time at most $O^*(b^{n-\ell})$ for $b = 2^{\beta_k}$. Therefore, we can give an upper bound of the overall time spent in answering all queries by bounding the time spent on a completely balanced binary search tree of depth $d = \log nN$.

Let $T(d, n)$ denote the overall time spend to run the algorithm on a balanced binary search tree with d levels with an n variable formula. Then, up to a sub-exponential factor

for the time spent at each node in the tree, $T(d,n) = b^n + 2T(d-1,n-1)$. Note that $T(0,n) = 1$, and thus

$$T(d,n) = \sum_{\nu=0}^{d} 2^\nu b^{n-\nu}$$

which yields the claimed bound. ◀

3.2 Formulas with many Solutions

We use the following simple folklore algorithm which can be found e.g. in Motwani and Raghavan's book [16]. Given a CNF formula F on n variables, choose an assignment from $\{0,1\}^n$ uniformly at random. Repeat this process a number N of times and let X be the number of solutions of F among these N trials. By a simple argument (see e.g. Theorem 11.1. in [16]), if

$$N = \Omega\left(\frac{2^n}{\epsilon^2 \# F}\right)$$

then with probability at least $3/4$, the value $X \cdot \frac{2^n}{N}$ is an e^ϵ-approximation of $\#F$. Hence, we have the following

▶ **Lemma 3.** *Let F be an n variable CNF formula with at least N solutions and ϵ^{-1} a natural number. Then there is an algorithm which, in time $O^*\left(\frac{2^n}{\epsilon^2 N}\right)$ yields a randomized e^ϵ-approximation of $\#F$.*

3.3 Combining the algorithms

We shall now prove Theorem 3.1 by combining both of the above algorithms. Let F be a k-CNF formula on n variables and ϵ^{-1} a natural number. We run the algorithm of Lemma 1 with a parameter N. The exact value of N will be determined later. Note that if the algorithm reports $\#F \leq N$, it also computes $\#F$ exactly with probability at least $3/4$. Otherwise, if the algorithm reports that $\#F > N$, we know with certainty that this is the case. Hence, given that the algorithm reports the latter, the algorithm of Lemma 3 will take time $O^*\left(\frac{2^n}{\epsilon^2 N}\right)$ to yield an e^ϵ-approximation of $\#F$.

It remains to bound the running time which amounts to optimizing the cutoff parameter N. For every choice of N, the combined algorithm works in time within a sub-exponential factor of

$$\max\{2^{\beta_k n} N^{(1-\beta_k)}, \frac{2^n}{N}\}.$$

Let f be such that $\log_2 N = f \cdot n$. Then this maximum translates into $\max\{\beta_k + f(1-\beta_k), 1-f\}$. Since $(\beta_k + f(1-\beta_k)$ is increasing and $1-f$ is decreasing in f, the minimum over all f of the maximum of the two is obtained when f is chosen so as to make them equal, that is, at

$$f = \frac{1-\beta_k}{2-\beta_k}.$$

This translates into an overall running time of

$$O^*\left(2^{\frac{n}{2-\beta_k}}\right).$$

Recall that β_k determines the running time $O^*(2^{\beta_k n})$ of the subroutine consisting of a randomized k-SAT algorithm, used in the algorithm of Lemma 1. We shall have a look at these running times, now.

3.3.1 The case $k \geq 5$

The algorithm of Paturi, Pudlák, Saks and Zane [17], can be used as the subroutine randomized k-SAT algorithm, which has a running time of

$$O^*\left(2^{(1-\frac{\mu_k}{k-1})n}\right). \tag{2}$$

Hence, we have here, $\beta_k = 1 - \frac{\mu_k}{k-1}$ which yields the claimed bound.

3.3.2 The cases $k = 3$ and $k = 4$

For these values of k, several improvements over the PPSZ algorithm have been presented. The currently fastest one is that by Hertli [9], whose bounds match those of the PPSZ algorithm in the unique-SAT case. We thus also have here the corresponding bound of equation (2).

4 Upper bounds

In this section we will prove Theorem 1.3 by presenting a simple algorithm producing upper bounds on $\#F$. We will use Valiant and Vazirani's bisection technique [23] and its application to approximate counting. We will therefore consider random GF(2)-linear systems of equations of the form $Ax = b$. For some $m \leq n$ these consist of an $m \times n$ matrix A, an m dimensional vector b and a vector x representing the variables of F. The entries of A and b are chosen independently and uniformly at random from $\{0, 1\}$. As such systems give rise to a family of pairwise independent hash functions of the form $h(x) = Ax - b$, we will use the following well known

▶ Fact 4.1 (Hashing Lemma). Let F be a CNF formula, and $Ax = b$ an $m \times n$ random linear system of equations. Then

$$\Pr\left[|\{x \in \mathrm{sat}(F) \mid Ax = b\}| \notin (1-\epsilon, 1+\epsilon) \cdot 2^{-m} \cdot \#F\right] \leq \frac{(1-2^{-m})2^m}{\#F \cdot \epsilon^2}.$$

The formulation of this fact in terms of CNF formulas is just for convenience. In fact, in its general wording it can be applied to any finite set – its proof can be found e.g. in Goldreich's book [6]. Secondly, we need a standard fact about the rank of random matrices such as the matrices obtained in the above way. Consider a random $m \times n$ matrix A as above with $m \leq n$ and let r denote its rank. The proof of the following lemma then follows easily, for example, from a result of Blömer, Karp and Welzl [1]:

▶ **Lemma 4.** *There is a constant c such that $\mathbb{E}[r] \geq m - c$. Furthermore, $r \geq m - O(\log m)$ with probability at least $1 - O(m^{-1})$.*

A third ingredient of the algorithm is the following Lemma, which can be proved by simple linear algebra .

▶ **Lemma 5.** *Let $Ax = b$ be a system of GF(2)-linear equations with solution set $\mathcal{A}$. There is an algorithm listing all solutions in time within a polynomial factor of $|\mathcal{A}|$.*

We are now ready to prove the Theorem.

Proof of Theorem 1.3. We start with the description of the algorithm. Choose a random GF(2)-linear $n \times n$ system of equations $Ax = b$. Starting with a parameter $\nu = n$ and decreasing ν in each step, we build random linear systems $A_n x = b_n, A_{n-1} x = b_{n-1}, \ldots, A_\mu x = b_\mu$.

The system $A_\nu x = b_\nu$ is obtained from $Ax = b$ by deleting the last $n - \nu$ rows of A and entries of b. Then F_ν denotes the pair consisting of F and $A_\nu x = b_\nu$. We say that F_ν is *satisfiable* if there is a solution $x \in \mathrm{sat}(F)$ such that $A_\nu x = b_\nu$.

For each ν we rigorously determine whether F_ν is satisfiable by using the algorithm of Lemma 5 to list all solutions of the linear system and for each determine whether it satisfies F. We let u be the minimum $\nu \geq \mu$ such that F_ν is unsatisfiable. If all $F_n, \ldots, F_\mu$ are unsatisfiable, we set $u = \mu$.

To establish the time bound, note that the running time is dominated by the time the algorithm spends in determining satisfiability of F_μ. By Lemma 4 the matrix A_μ has with high probability rank at least $m - O(\log n)$ and we have $|\mathcal{A}| = O^*(2^{n-\mu})$ which yields the claimed time bound. Thus we shall in the following condition on the event that the rank of A_μ satisfies this rank criterion.

Correctness

The correctness follows from standard arguments also used in the classical approach [23]. We give a proof for completeness. Let $f = \lceil \log \#F \rceil$. Assume first, for simplicity, that $\mu = 0$. We will show that with the desired probability, u is a 16-approximation to $\#F$.

First, consider the event that $u = f - c$ for some $c \geq 4$. The probability p_c of this event is the probability that all of $F_n, \ldots, F_u$ are unsatisfiable. Furthermore, conditional on F_u being unsatisfiable, all F_ν for $\nu \geq u$ are unsatisfiable, as well. Hence, we have $p_c = \Pr[F_u \text{ is unsat }]$, and by the Hashing Lemma 4.1, we have thus

$$\Pr[F_u \text{ is unsat }] < \frac{2^u}{\#F} \leq 2^{1-c}.$$

By a union bound argument, we thus see that $f - 3 \leq u$ with probability at least $3/4$.

Next, consider $u = f + c$ which implies that F_{u-1} is satisfiable. Applying the Hashing Lemma 4.1 with parameter $\epsilon = \zeta - 1$ for $\zeta = 2^{u-1}/\#F$, we see that

$$\Pr[F_{u-1} \text{ is satisfiable}] < \frac{\zeta}{(\zeta - 1)^2}.$$

As $\zeta \geq 1$, this bound is decreasing in ζ, and hence in u, therefore, the probability that F_{u-1} is satisfiable is at most $2^{c-1}(2^{c-1} - 1)^{-2}$. Again, a union bound shows that $u \leq f + 3$ with probability at least $33/49$.

Next, note that if $\mu > 0$ then these findings do not change. Especially, if $\mu < f - 3$ then the above result does not change. And if $\mu \geq f - 3$, then by the above, with probability at least $3/4$ we have $u = \mu$.

Taking into account that we have conditioned on A_μ having rank $m - O(\log m)$ the above probabilities degenerate a bit. But the bounds claimed in the statement of the Theorem are still easily achieved. ◀

Remarks

Note that the use of listing algorithm of Lemma 5 can be avoided by using a uniform sampling algorithm for the solutions of $Ax = b$, this then yields essentially the same time bounds. Furthermore, uniform sampling is easily achieved by fixing a basis of the column space of A, choosing u.a.r. assignments to non-basis variables and extending these assignments to solutions of $Ax = b$.

5 Open Problems

It is a peculiar fact that our result falls short of yielding any reasonable time bound for the approximation of #2-SAT. A direct application of the algorithm of Theorem 3.1 to this case would yield (using a polynomial time 2-SAT subroutine) a bound of $O(1.4142^n)$ whereas the fastest exact method [24] takes time only $O(1.2377^n)$. It would therefore be interesting to develop an approximation algorithm which beats the bounds of these exact methods.

Secondly, the time bounds achieved in this paper are significantly better than those for known exact methods, but also, they are much worse than the bounds known for the corresponding satisfiability problems. Is it possible to close this gap, maybe even in terms of a purely algorithmic analog of Valiant and Vazirani's result?

6 Acknowledgments

I would like to thank Martin Grohe for bringing this problem to my attention and for several helpful discussions on the topic. For further discussions, I would also like to thank Holger Dell, Alistair Sinclair, and Piyush Srivastava.

References

1 Johannes Blömer, Richard M. Karp, and Emo Welzl. The rank of sparse random matrices over finite fields. *Random Struct. Algorithms*, 10(4):407–419, 1997.
2 Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2sat and 3sat formulae. *Theor. Comput. Sci.*, 332(1-3):265–291, 2005.
3 Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theor. Comput. Sci.*, 81(1):49–64, 1991.
4 Martin Fürer and Shiva Prasad Kasiviswanathan. Algorithms for counting 2-satsolutions and colorings with applications. In *AAIM*, pages 47–57, 2007.
5 Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. In *AAAI*, pages 198–203, 2007.
6 Oded Goldreich. *Computational complexity*. Cambridge University Press, Cambridge, 2008. A conceptual perspective.
7 Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. From sampling to model counting. In *IJCAI*, pages 2293–2299, 2007.
8 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *AAAI*, 2006.
9 Timon Hertli. 3-sat faster and simpler - unique-sat bounds for ppsz hold in general. In *FOCS*, 2011. To appear.
10 Timon Hertli, Robin A. Moser, and Dominik Scheder. Improving ppsz for 3-sat using critical variables. In *STACS*, pages 237–248, 2011.
11 Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-sat. In J. Ian Munro, editor, *SODA*, page 328. SIAM, 2004.
12 Lukas Kroc, Ashish Sabharwal, and Bart Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. *Annals OR*, 184(1):209–231, 2011.
13 Konstantin Kutzkov. New upper bound for the #3-sat problem. *Inf. Process. Lett.*, 105(1):1–5, 2007.
14 B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Appl. Math.*, 10(3):287–295, 1985.
15 Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's k-sat algorithm. In *STOC*, pages 245–252, 2011.

16 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

17 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for -sat. *J. ACM*, 52(3):337–364, 2005.

18 Daniel Rolf. Improved bound for the PPSZ/Schöning-algorithm for 3-sat. *JSAT*, 1(2):111–122, 2006.

19 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

20 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *FOCS*, pages 410–414, 1999.

21 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

22 Larry J. Stockmeyer. On approximation algorithms for #p. *SIAM J. Comput.*, 14(4):849–861, 1985.

23 Leslie G. Valiant and Vijay V. Vazirani. Np is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

24 Magnus Wahlström. A tighter bound for counting max-weight solutions to 2sat instances. In *IWPEC*, pages 202–213, 2008.

25 Wei Wei and Bart Selman. A new approach to model counting. In *SAT*, pages 324–339, 2005.

26 Wenhui Zhang. Number of models and satisfiability of sets of clauses. *Theor. Comput. Sci.*, 155(1):277–288, 1996.