Information Processing Letters 113 (2013) 337–344

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Exploiting independent subformulas: A faster approximation scheme for #k-SAT

Manuel Schmitt *, Rolf Wanka

Department of Computer Science, University of Erlangen-Nuremberg, Cauerstraße 11, 91058 Erlangen, Germany

ARTICLE INFO

Article history:
Received 10 September 2012
Received in revised form 22 February 2013
Accepted 25 February 2013
Available online 27 February 2013
Communicated by J. Torán

Keywords:
Algorithms
Analysis of algorithms
Randomized algorithms
#k-SAT
Satisfiability

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.

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 [1] 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 [2]) that k-SAT is NP-complete for any $k \geqslant 3$, and that it can be solved in time linear in the input length for $k = 2$ [3]. So it is generally assumed that there is no polynomial time algorithm solving k-SAT for $k \geqslant 3$. 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] $O^*(1.3303^n)$ [4], the fastest randomized algorithm has a running time of $O^*(\log(\delta^{-1}) \cdot 1.30704^n)$ [5]. 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 example from statistical physics is the computation of the number of configurations in monomer-dimer systems (for an overview, see [6]). The complexity class that corresponds to these *counting problems* is #P, and #SAT, the problem to determine the number of satisfying

* Corresponding author. Tel.: +49 9131 85 25387; fax: +49 9131 85 25149.
E-mail addresses: manuel.schmitt@cs.fau.de (M. Schmitt), rwanka@cs.fau.de (R. Wanka).

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

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2013.02.013

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 Section 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					#k-SAT				
	deterministic		rand		β_k	exact		rand, prev.		this paper
$k=2$	$n+m$	[3]	–		–	1.2377^n	[8]	–		–
$k=3$	1.3303^n	[4]	1.30704^n	[5]	0.3864	1.6423^n	[9]	1.5366^n	[10]	1.51426^n
$k=4$	1.5^n	[11]	1.46899^n	[5]	0.5548	1.9275^n	[12,13]	1.6155^n	[10]	1.60816^n
$k \geqslant 5$	$(\frac{2\cdot(k-1)}{k})^n$	[11]	$2^{(1-\mu_k/(k-1))\cdot n}$	[14]	$1-\frac{\mu_k}{k-1}$	ψ_k^n	[13]	$2^{1/(2-\beta_k)\cdot n}$	[10]	α_k^n (Section 5.3)

assignments, is well-known to be #P-complete. More exactly, let #k-SAT denote the problem to determine #Φ, i.e., for input Φ being a k-CNF, the number of satisfying assignments. Then, it is known [7] that #k-SAT is #P-complete for $k \geqslant 2$.

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 #Φ (for exact definitions, see Section 2). In a recent paper, Thurley [10] 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 Section 2. Table 1 also presents all best running times currently known to solve #k-SAT.

A different approach by Impagliazzo et al. [15] 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 Section 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 (Section 4), and on large sets of maximal independent subformulas (Section 5).

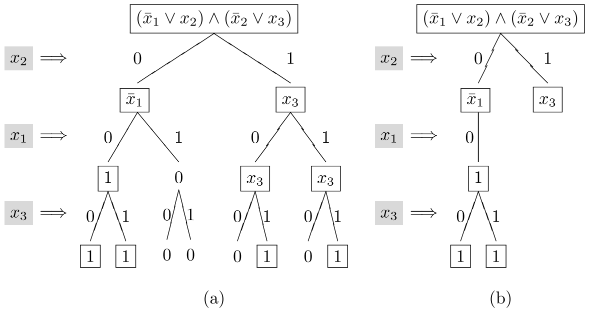

Fig. 1. (a) Elimination tree and (b) a 3-cut for $\Phi = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$, due to the elimination order (x_2, x_1, x_3). The sum of the leaves is $\#\Phi = 4$. Satisfiable nodes are boxed. Note that #Φ can already be computed from the nodes on level 1.

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 $x_1, \dots, x_n$ 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 $0, \dots, n$) 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* $(y_1, \dots, y_n)$ 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, $0 \leqslant i < n$, 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. #Φ 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 $\#\Phi \geqslant \ell$ follows. Note that the elimination order significantly influences the moment when in the