Available online at www.sciencedirect.com

SCIENCE DIRECT

Electronic Notes in Discrete Mathematics 17 (2004) 69–73

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

An improved local search algorithm for 3-SAT

Tobias Brueggemann[1] , Walter Kern[2]

Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

We slightly improve the pruning technique presented in Dantsin et. al. (2002) to obtain an $\mathcal{O}^*(1.473^n)$ algorithm for 3-SAT.

Keywords: exact algorithm, local search, 3-SAT
MSC2000: 68Q25

1 Introduction

An instance of 3-SAT is a boolean formula φ in n variables $x_1, \ldots, x_n$, defined as the conjunction of a set $\mathcal{C}$ of disjunctive clauses of length at most 3. Satisfiability of φ can be tested in a straightforward manner in time

$$\mathcal{O}\left(2^n \cdot n^3\right) = \mathcal{O}^*\left(2^n\right).$$

Here, as usual, we use the $\mathcal{O}^*$-notation to indicate that polynomial factors are suppressed.

During the last years so-called *exact algorithms* have been designed solving 3-SAT in time $\mathcal{O}^*(\alpha^n)$ with $\alpha < 2$, see Schoening [3] for an overview. The currently fastest randomized algorithms run in time $\mathcal{O}^*(1.3302^n)$ (see Hofmeister,

[1] Email: t.brueggemann@math.utwente.nl. Supported by the Netherlands Organization for Scientific Research (NWO) grant 613.000.225 (Local Search with Exponential Neighborhoods)

[2] Email: kern@math.utwente.nl. Corresponding author.

1571-0653/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.endm.2004.03.021

Schoening, Schuler and Watanabe [2]) and the fastest deterministic algorithm (see Dantsin et. al. [1]) takes $\mathcal{O}^*(1.481^n)$. We slightly improve the pruning technique used in Dantsin et. al. [1] to obtain a running time of $\mathcal{O}^*(1.473^n)$.

2 Local search

Let φ be an instance of 3-SAT given by a set $\mathcal{C}$ of clauses in variables $x_1, \ldots, x_n$. For $a \in \{0,1\}^n$ let $B_r(a) \subseteq \{0,1\}^n$ denote the set of 0-1 vectors with Hamming distance at most r from a. The currently fastest algorithms for 3-SAT are based on *local search*: First, a *covering code* of suitable *radius* $r \leq n$ is constructed, i.e. a set $A \subseteq \{0,1\}^n$ such that

$$\{0,1\}^n = \bigcup_{a \in A} B_r(a)$$

holds. Next we search for a truth assignment for φ in each $B_r(a)$, $a \in A$, separately. To make our paper self-contained, we briefly describe the basic idea for constructing a covering code and (to some extent) the local search within a given $B_r(a)$ as presented in Dantsin et. al. [1].

Covering codes.

As $B_r := B_r(0)$ contains exactly

$$V(n,r) = \sum_{i=1}^{r} \binom{n}{i}$$

elements, a covering code $A \subseteq \{0,1\}^n$ of radius $r \leq n$ must necessarily satisfy

$$|A| \geq \frac{2^n}{V(n,r)}.$$

Covering codes of approximately this size indeed exist and can be constructed randomly: Choose

$$t = \frac{n 2^n}{V(n,r)}$$

elements from $\{0,1\}^n$ uniformly at random, resulting in a set $A \subseteq \{0,1\}^n$ of size $|A| \leq t$. The probability that a particular $a^* \in \{0,1\}^n$ is *not* covered by any $B_r(a)$, $a \in A$ is at most

$$P[a^* \text{ not covered}] = \left(1 - \frac{V(n,r)}{2^n}\right)^t \leq \mathrm{e}^{-n},$$

using $1 + x \leq e^x$ for $x \in \mathbb{R}$. So the probability that A is *not* a covering code is at most $2^n e^{-n}$, which tends to 0 as $n \to \infty$.

This procedure can be de-randomized by taking in each step a new code word $a \in \{0,1\}^n$ that is best possible in the sense that it covers as many as possible of the yet uncovered elements in $\{0,1\}^n$. Note, however, that this *greedy construction* takes $\mathcal{O}^*(2^n)$ per step and thus almost $\mathcal{O}(2^{2n}) = \mathcal{O}^*(4^n)$ in total (which is far too slow). Dantsin et. al. [1] therefore propose the following. Let $K \in \mathbb{N}$ be a constant and assume w.l.o.g. that $n = Kn_0$ and $r = Kr$. Then construct a covering code $A_0 \subseteq \{0,1\}^{n_0}$ in time $\mathcal{O}(4^{n_0}) = \mathcal{O}^*\left(\sqrt[K]{4}^n\right)$ and take

$$A = \underbrace{A_0 \times \ldots \times A_0}_{K \text{ times}}$$

as a covering code for $\{0,1\}^n$. Proceeding this way, the time needed for constructing the covering code becomes negligible.

Local search.

Assume we want to search for a truth assignment for φ in $B_r(a) \subseteq \{0,1\}^n$. We may assume w.l.o.g. that $a = 0$, i.e., we search in $B_r = B_r(0)$. (Interchange x_i with $\overline{x}_i$ if necessary.) If $a = 0$ is not a truth assignment for φ, there must exist a *false clause*, i.e. a clause $C \in \mathcal{C}$ that is false under $a = 0$, say $C = (x_i \vee x_{i'} \vee x_{i''})$. It then suffices to search for a truth assignment in $B_{r-1} \subseteq \{0,1\}^{n-1}$ w.r.t. each of the formulae

$$\varphi_1 = \varphi[x_i = 1], \; \varphi_2 = \varphi[x_{i'} = 1] \text{ and } \varphi_3 = \varphi[x_{i''} = 1],$$

obtained by fixing a variable as indicated in brackets. If necessary, we may even fix in addition some variables to zero, e.g., define $\varphi_1 := \varphi[x_i = 1], \varphi_2 := \varphi[x_{i'} = 1, x_i = 0]$ and $\varphi_3 := \varphi[x_{i''} = 1, x_i = 0, x_{i'} = 0]$.

Continuing this way, our search can be described by a *search tree* T_r, constructed by *branching on false clauses* (one false clause per node), as indicated in figure 1.

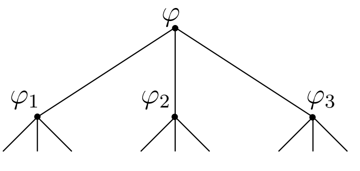

Fig. 1. The search tree T_r

Needless to say that we never branch to formulas $\varphi' = \varphi[x_i = 1, \ldots]$ that are obviously non-satisfiable because they contain an empty (non-satisfiable) clause. (For example, if $(\overline{x}_i) \in \mathcal{C}$, we would only branch to φ_2 and φ_3 in figure 1.) We denote the number of leaves of T_r by $|T_r|$ and refer to it as the *size* of T_r. Clearly,

$$|T_r| \leq 3^r \tag{1}$$

holds, an immediate consequence of the recursion $|T_r| \leq 3|T_{r-1}|$ (see figure 1). In case φ contains a false 2-clause $C \in \mathcal{C}$, then branching on C would yield $|T_r| \leq 2|T_{r-1}|$.

As pointed out in Dantsin et. al. [1], this simple argument already gives an $\mathcal{O}^*\left(\sqrt[2]{3}^n\right) \approx \mathcal{O}^*(1.7321^n)$ algorithm: Take $r = \frac{n}{2}$ and search $B_r(0)$ and $B_r(1)$ separately in time $\mathcal{O}^*(3^r) = \mathcal{O}^*\left(\sqrt[2]{3}^n\right)$ each.

Smaller search trees.

The trivial bound (1) on the size of the search tree can be improved by a clever branching technique, as shown in Dantsin et. al. [1]: Assume that φ contains three pairwise disjoint false clauses $C = (x_i \vee x_{i'} \vee x_{i''}), C_1 = (x_j \vee x_{j'} \vee x_{j''})$ and $C_1' = (x_k \vee x_{k'} \vee x_{k''})$ and a (true) clause $(\overline{x}_i \vee \overline{x}_j \vee \overline{x}_k)$. We may then *branch along* $(\overline{x}_i \vee \overline{x}_j \vee \overline{x}_k)$, i.e. first branch on C at the root node φ, then branch on C_1 at $\varphi_1 = \varphi[x_i = 1]$ and finally branch on C_1' at $\varphi_1' = \varphi_1[x_j = 1] = \varphi[x_i = 1, x_j = 1]$. The resulting search tree is indicated in figure 2.

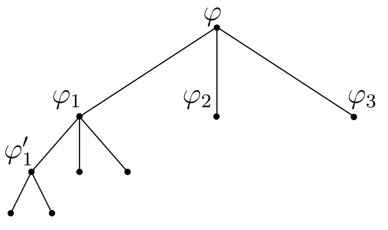

Fig. 2. Branching along $(\overline{x}_i \vee \overline{x}_j \vee \overline{x}_k)$

Note that the node corresponding to φ_1' has only two descendants because $\varphi[x_i = 1, x_j = 1, x_k = 1]$ is ruled out by the clause $(\overline{x}_i \vee \overline{x}_j \vee \overline{x}_k)$.

If a similar branching was possible also at φ_2 and φ_3, we would get a search tree satisfying a recursion

$$|T_r| \leq 6|T_{r-2}| + 6|T_{r-3}|. \tag{2}$$

Indeed, this is what Dantsin et. al. [1] show. Assuming inductively that $|T_k| \leq c\alpha^k$ holds for some constant $c > 0$, (2) implies that

$$|T_r| \leq \mathcal{O}(\alpha^r), \tag{3}$$

where $\alpha = \sqrt[3]{4} + \sqrt[3]{2} \approx 2.848$ is the largest root of $\alpha^3 - 6\alpha - 6 = 0$.

The main result of our paper slightly improves this bound as follows.

Theorem 2.1 *By branching on false clauses we can ensure that*

$$|T_r| \leq c\beta^r,$$

where $\beta = \frac{1+\sqrt{21}}{2} \approx 2.792$ *is the largest root of* $\beta^3 - 6\beta - 5 = 0$.

Running time.

Let $\varrho < \frac{1}{2}$ and $r = \varrho n$. By Stirling's formula, the size of a covering code we construct is (up to a polynomial factor) bounded by

$$|A| = \mathcal{O}^* \left(\left[2\varrho^\varrho (1-\varrho)^{1-\varrho} \right]^n \right).$$

According to (3), the number of nodes in T_r is bounded by $n|T_r| = \mathcal{O}^*(|T_r|)$ and hence the total running time is thus bounded by

$$\mathcal{O}^*(|A||T_r|) = \mathcal{O}^* \left(\left[2(\alpha\varrho)^\varrho (1-\varrho)^{1-\varrho} \right]^n \right).$$

This expression is minimal for $\varrho \approx 0.26$, yielding the bound of $\mathcal{O}^*(1.481^n)$ in Dantsin et. al. [1].

Similarly, replacing α by β from Theorem 2.1, we obtain for $\varrho \approx 0.264$ an exact algorithm that runs in $\mathcal{O}^*(1.473^n)$.

References

[1] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, O. Raghavan, U. Schoening [2002]: *A deterministic* $(2-2/(k+1))^n$ *algorithm for k-SAT based on local search.* In: Theoretical Computer Science 289 (2002), 69-83. Elsevier Science B.V.

[2] T. Hofmeister, U. Schoening, R. Schuler, O. Watanabe [2002]: *A Probabilistic 3-SAT Algorithm Further Improved.* In: H. Alt, A. Ferreira (Eds.): STACS 2002, LNCS 2285, 192-202. Springer-Verlag Berlin Heidelberg.

[3] U. Schoening [2002]: *A Probabilistic Algorithm for k-SAT Based on Limited Local Search and Restart.* In: Algorithmica 32 (2002), 615-623. Springer-Verlag New York Inc.