Worst-case bounds on k-SAT

Uwe Schöning

Universität Ulm
Abteilung Theoretische Informatik

schoenin@informatik.uni-ulm.de

Series of deterministic and probabilistic algorithms for k-SAT based on local search principle.

We specialize on 3-SAT.

For $n =$ number of variables, we have the straightforward worst case upper bound:
$poly(n) \cdot 2^n$.

Given a set of 3-clauses like

$$(x \vee \overline{y} \vee z), (\overline{x} \vee u \vee v), (\overline{y} \vee \overline{u}), (\overline{z} \vee v)$$

Determine satisfiability as follows.

Procedure Test (assignment a, integer m) :

If a is satisfying **Then** STOP (success)

If $m = 0$ **Then Return**

Else:

Let $C = (x_1 \vee x_2 \vee x_3)$ be some clause not satisfied by a, i.e. $a(x_1) = a(x_2) = a(x_3) = 0$

For $i := 1$ **To** 3 **Do** Test(modify(a, x_i) , $m - 1$)

Finds any satisfying assignment a^* within Hamming distance m from the initial assignment a, i.e. $d(a, a^*) \leq m$.

Complexity is $poly(n) \cdot 3^m$.

First variant (deterministic):

Run **Test**$(0^n, n/2)$ and **Test**$(1^n, n/2)$.

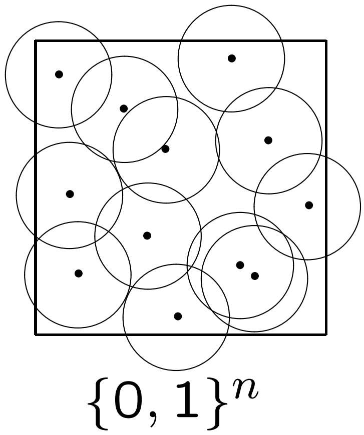
Complexity is $poly(n) \cdot 3^{n/2} = poly(n) \cdot 1.73^n$.

Second variant (probabilistic):

Repeat $t(n)$-times:

  Choose $a \in \{0,1\}^n$ at random;

  Run **Test**$(a, n/4)$.



Complexity is $poly(n) \cdot t(n) \cdot 3^{n/4} = poly(n) \cdot 1.5^n$.
This holds for

$$t(n) = 100 \cdot \frac{2^n}{\sum_{i=0}^{n/4} \binom{n}{i}}$$

Error probability is e^{-100}.

Remark on probabilistic algorithms

Given algorithm A that on satisfiable 3-SAT instance finds a satisfying assignment with probability at least q^{-n}, where $q > 1$.

Then consider A^*:

Repeat $100 \cdot q^n$ times:

Start A. If satisfying assignment found:
Output "satisfiable" and STOP

End-Repeat

Output "not satisfiable".

Complexity of A^* is $poly(n) \cdot q^n$.

A^* has one-sided error probability which is $(1 - q^{-n})^{100 \cdot q^n} \leq e^{-100}$.

Replace Procedure **Test** by some random walk on the set of assignments.

Procedure RandomWalk (assignment a):

Repeat $3n$ times:

  If a satisfying **Then** STOP (success)

  Else:

  Let $C = (x_1 \vee x_2 \vee x_3)$ be some clause not satisfied by a, i.e. $a(x_1) = a(x_2) = a(x_3) = 0$

  Choose $k \in \{1, 2, 3\}$ at random;

  $a :=$ modify(a, x_k);

End-Repeat

With probability (at least) $1/3$ the Hamming distance $d(a, a^*)$ decreases by 1. ("right direction")

With probability (at most) $2/3$ the Hamming distance $d(a, a^*)$ increases by 1. ("wrong direction")

Like gambler's ruin problem (ruin = sat. assignment found)

If we start the random walk at Hamming distance j, what is the probability of reaching a satisfying assignment a^* (or even stopping earlier)?

We are successful if during the first $3j \leq 3n$ steps we make $2j$ steps in the right direction and j steps in the wrong direction. Probability is:

$$\begin{aligned} & \binom{3j}{j} \cdot (1/3)^{2j} \cdot (2/3)^j \\ \geq \;& \frac{1}{O(\sqrt{j})} \cdot 2^{3j \cdot H(1/3)} \cdot (1/3)^{2j} \cdot (2/3)^j \\ = \;& \frac{1}{O(\sqrt{j})} \cdot (1/2)^j \end{aligned}$$

We used the entropy function H and an estimation

$$\binom{n}{\alpha n} \geq \frac{1}{\sqrt{8n\alpha(1-\alpha)}} \cdot 2^{n \cdot H(\alpha)}$$

Easiest way to exploit this: start random walk in Hamming distance $\leq n/2$, then success probability is $(1/2)^{n/2}/O(\sqrt{n})$.

First variant:

Run **RandomWalk**(0^n) for $100 \cdot 2^{n/2}$ times
and run **RandomWalk**(1^n) for $100 \cdot 2^{n/2}$ times.

Complexity is $poly(n) \cdot 2^{n/2} = poly(n) \cdot 1.414^n$.

Error probability is e^{-100}.

Second variant:

Repeat $t(n)$-times:

Choose $a \in \{0, 1\}^n$ at random;

Run **RandomWalk**(a).

This time the expected Hamming distance between a and a^* still is $n/2$.

Seems like same result as above, but ... let's look more closely.

Define the following 0-1-valued random variables:

$$X_i = \begin{cases} 1, & \text{if } a_i \neq a_i^*, \\ 0, & \text{otherwise} \end{cases}$$

Then $d(a, a^*) = X_1 + \ldots + X_n$ and

$$\begin{aligned} P(success) &\geq \frac{1}{O(\sqrt{n})} \cdot \sum_a P(a \text{ chosen}) \cdot (1/2)^{d(a,a^*)} \\ &= \frac{1}{O(\sqrt{n})} \cdot E[X] \end{aligned}$$

where

$$X = (1/2)^{d(a,a^*)} = (1/2)^{X_1 + \ldots + X_n}.$$

We have

$$\begin{aligned} E[X] &= E[(1/2)^{X_1 + \ldots + X_n}] \\ &= E[(1/2)^{X_1}] \cdots E[(1/2)^{X_n}] \end{aligned}$$

because of independence.

X_i is either 0 or 1, hence
$E[(1/2)^{X_i}] = (1/2) \cdot (1/2)^1 + (1/2) \cdot (1/2)^0 = 3/4.$

Thus, the above algorithm has running time $poly(n) \cdot (4/3)^n$ and error probability e^{-100}. (Choosing $t(n) = 100 \cdot (4/3)^n$.)

Improvement

Up to now, the random choice of the initial assignment is independent of the given 3-SAT instance.

Example: Even if $C = (x_1 \vee x_2 \vee x_3)$, then still with probability 1/8 all 3 variables are set to zero.

Can we avoid this?

Basic Idea: Assign variable values in groups of 3, avoid to set them all to zero.

Problem: Clauses share variables.

Definition: Two clauses are called **independent**, if they have no variable in common.

First part of a **New Algorithm**:

In a greedy fashion, collect a **maximally independent** set of clauses M (each consisting of 3 literals.) Each remaining clause has at least one variable in common with some clause in M.

Let these clauses in M be $C_1, \ldots, C_m$.

Now, either m is "small" ($m \leq 0.14665 \cdot n$), then use **Algorithm 1**,

or m is not "small", then use **Algorithm 2**.

Algorithm 1: Cycle through all 7^m (partial) assignments which make the clauses in M true.
After plugging in such a partial assignment, the remaining formula becomes a 2-SAT instance, and is solvable in polynomial time.

Complexity in this case: $poly(n) \cdot 7^{0.14665n} = poly(n) \cdot 1.33026^n$.

Algorithm 2:

Repeat $t(n)$ times the following:

Assign random values to the independent clauses $C = (x \vee y \vee z)$ in M with the following probabilities:

p_1 to $(0,0,1)$ $\qquad$ p_2 to $(0,1,1)$

p_1 to $(0,1,0)$ $\qquad$ p_2 to $(1,1,0)$

p_1 to $(1,0,0)$ $\qquad$ p_2 to $(1,0,1)$

p_3 to $(1,1,1)$

using probabilities satisfying $3p_1 + 3p_2 + p_3 = 1$.
(Choose $p_1 = 4/21,\ p_2 = 2/21,\ p_3 = 3/21$.)

Assign random, independent values to the remaining variables, as before.

Run **RandomWalk**(this assignment).

Analysis of Algorithm 2:

As before, we have to analyze $E[(1/2)^{d(a,a^*)}]$. This time, some of the bits are chosen in groups of 3. W.l.o.g. assume M consists of

$$\begin{aligned} x_1 \vee x_2 \vee x_3, \\ x_4 \vee x_5 \vee x_6, \\ \ldots, \\ x_{3m-2} \vee x_{3m-1} \vee x_{3m} \end{aligned}$$

Define

$$\begin{aligned} X_{1,2,3} &= d(\, a_1 a_2 a_3 \,,\, a_1^* a_2^* a_3^* \,) \\ X_{4,5,6} &= d(\, a_4 a_5 a_6 \,,\, a_4^* a_5^* a_6^* \,) \\ &\text{etc.} \end{aligned}$$

Then $P(success) = (1/O(\sqrt{n}))$ times

$E[(1/2)^{X_{1,2,3}}] \cdots E[(1/2)^{X_{3m-2,3m-1,3m}}] \cdot (3/4)^{n-3m}$

By the above choice of the probabilities p_1, p_2, p_3 all expressions $E[(1/2)^{X_{i,i+1,i+2}}]$ obtain the value $3/7$.

Altogether we get the success probability

$$(1/O(\sqrt{n})) \cdot (3/7)^m \cdot (3/4)^{n-3m}$$

This gives the complexity of Algorithm 2:

$$poly(n) \cdot (7/3)^m \cdot (4/3)^{n-3m}$$

which is (for $m > 0.14665 \cdot n$) at most

$$poly(n) \cdot 1.33026^n$$

Latest improvement (by Sven Baumer and Rainer Schuler, Ulm):

Find in a Greedy fashion **pairs** of clauses which share at least one variable. Collect a **maximally independent** set of such pairs, say P.

Further collect among the remaining clauses those which are independent (of each other and of the clauses in P). Call this set M.

A similar case distinction as before, resulting in (analogues of) Algorithm 1 and Algorithm 2.

For Algorithm 1, one needs to cycle through 25 assignments for the clause-pairs in P (instead 32).

For Algorithm 2, the analogue of p_1, p_2, p_3 consists of 9 probabilities $q_1, \ldots, q_9$ which need to be fixed in a suitable way.

The resulting complexity bound for 3-SAT is $poly(n) \cdot 1.329^n$.

Last Slide

(open questions, further developments, etc.)

Not all mentioned results are easily generalizable to k-SAT for $k > 3$.

Is it possible to reduce, instead of 2-SAT, to other easy SAT cases, like Horn-SAT?

Is it possible to exploit the fact that sometimes the probability for moving toward the ”right direction” in the random walk algorithm is 1/2 or 2/3, etc.

Very Last Slide

(References.)

U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. *Proc. 40th Sympos. on Foundations of Computer Science* 1999, New York, IEEE, 410–414.

U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica* 32,4 (2002) 615–623.

T. Hofmeister, U. Schöning, R. Schuler, O. Watanabe. A probabilistic 3-SAT algorithm further improved. *Sympoium on Theoretical Aspects of Computer Science 2002*. Lecture Notes in Computer Science 2285, Springer 2002, 192–202.

S. Baumer, R. Schuler: yet unpublished, 2002.