A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems

Uwe Schöning
Universität Ulm, Abteilung Theoretische Informatik
James-Franck-Ring, D-89069 Ulm, Germany
e-mail: schoenin@informatik.uni-ulm.de

Abstract

We present a simple probabilistic algorithm for solving k-SAT, and more generally, for solving constraint satisfaction problems (CSP). The algorithm follows a simple local-search paradigm (cf. [9]): randomly guess an initial assignment and then, guided by those clauses (constraints) that are not satisfied, by successively choosing a random literal from such a clause and flipping the corresponding bit, try to find a satisfying assignment. If no satisfying assignment is found after $O(n)$ steps, start over again. Our analysis shows that for any satisfiable k-CNF formula with n variables this process has to be repeated only t times, on the average, to find a satisfying assignment, where t is within a polynomial factor of $(2(1-1/k))^n$. This is the fastest (and also the simplest) algorithm for 3-SAT known up to date.

We consider also the more general case of a CSP with n variables, each variable taking at most d values, and constraints of order l, and analyze the complexity of the corresponding (generalized) algorithm. It turns out that any CSP can be solved with complexity at most $(d \cdot (1-1/l)+\varepsilon)^n$.

1. Algorithms for k-SAT

Several algorithms have been designed for k-SAT, and some in particular for the special case 3-SAT which beat the naive 2^n bound that is obtained by trying all potential 2^n many assignments for the n variables in the input formula.

The following list summarizes the known results for k-SAT and adds our new one, indicated by [*]. A constant c in the list means that there is an algorithm of the given type (deterministic or probabilistic) with complexity within a polynomial factor of c^n.

3-SAT	4-SAT	5-SAT	6-SAT	type	ref.
1.849	-	-	-	det.	[15]
1.782	1.835	1.867	1.888	det.	[13]
1.618	1.839	1.928	1.966	det.	[10]
1.588	1.682	1.742	1.782	prob.	[13]
1.579	-	-	-	det.	[17]
1.505	-	-	-	det.	[8]
1.5	1.6	1.667	1.715	prob.	[20]
1.497	-	-	-	det.	[19]
1.476	-	-	-	det.	[16]
1.447	1.496	1.569	1.637	prob.	[14]
1.362	1.476	-	-	prob.	[14]
1.334	1.5	1.6	1.667	prob.	[*]

2. The Algorithm

In the following we describe and analyze our algorithm. First consider the following probabilistic procedure:

input: a formula in k-CNF with n variables
Guess an initial assignment $a \in \{0,1\}^n$,
uniformly at random
Repeat $3n$ times:
  If the formula is satisfied by the actual assignment: stop and accept
  Let C be some clause not being satisfied by the actual assignment
  Pick one of the $\leq k$ literals in the clause at random and flip its value in the current assignment

Suppose we have a satisfiable formula and fix some satisfying assignment a^*. We want to estimate the probability that the algorithm finds a^* (or some other satisfying assignment.) Once we have found this "success probability" p, it is clear that the expected number of independent repetitions of the procedure until we find a satisfying assignment is $\frac{1}{p}$. The probability that we don't get a satisfying assignment

after t repetitions is at most $(1-p)^t \leq e^{-pt}$. Therefore, to achieve an acceptable error probability of, say e^{-20}, one needs to choose $t = \frac{20}{p}$. In any case, the complexity of the algorithm is within a polynomial factor of $\frac{1}{p}$.

Now we calculate p. It is clear that the random variable X that counts the number of bits in which the random assignment a and the fixed assignment a^* disagree (i.e. the Hamming distance between a and a^*) is binomially distributed (with parameters n and $\frac{1}{2}$.) That is, $Pr(X = j) = \binom{n}{j}2^{-n}$ for $j = 0, 1, \ldots, n$. Under the condition that $X = j$, the number of bits that have to be flipped to get from a to a^* is j (provided those bits are correctly selected.) We can imagine the process as a Markov chain. There is an initial state *start*, and there are the states $0, 1, \ldots, n$ which indicate the Hamming distance between a and a^*. The transfer from *start* to one of the state $0, 1, \ldots, n$ corresponds to the random selection of the initial assignment a. The transfer probability from *start* to state j is $\binom{n}{j}2^{-n}$. If the system is in state 0, this means, a satisfying assigment has been found, and the process stops. (Notice that the algorithm might even find *another* satisfying assignment in a state different from 0, if the formula has more than one satisfying assignment, but this situation can only increase the actual acceptance probability.)

If C is a clause that is not satisfied by the assignment a, then there must be at least one literal (out of k) in that clause whose value needs to be flipped so that the Hamming distance between a^* and a decreases by 1. Selecting a literal from such a k-CNF clause at random means that the current state j transfers to state $j-1$ with probability at least $\frac{1}{k}$, and transfers to state $j+1$ with probability at most $\frac{k-1}{k}$.

This Markov chain approach is very similar to the randomized algorithm by Papadimitriou [12] for 2-SAT. The difference here is that we do not allow the random walk to run for very long, just up to $3n$ steps, say, since the process tends to move into states that correspond to a large Hamming distance. The idea in our algorithm is that with (exponentially small, but) non-neglectible probability it can happen that the start state transfers to a state, say $n/4$, which is relatively close to the final state 0, and the probability to reach the final state is in this case at least $(\frac{1}{k})^{n/4}$ (but actually somewhat higher because the process runs for more than $n/4$ steps.)

Given that the process has initially transfered into state j we calculate the probability q_j that the process reaches the absorbing state 0. For this to happen the process needs at least j steps. We consider the case that the random walk takes $i \leq j$ steps in the "wrong" direction, then $i+j$ steps are required toward the "right" direction so that the process stops in state 0 after $j+2i$ steps. To calculate this probability requires to calculate the number of paths on a rectangular grid (which represents the possible movements on the Markov chain over the time scale) which transfers the process from state j to state 0 while using exactly i steps in the "wrong" direction. Using the ballot theorem from [5], page 73, it can be seen that this number is $\binom{j+2i}{i} \cdot \frac{j}{j+2i}$. Therefore, the probability can be estimated as follows.

$$\begin{aligned} q_j &\geq \sum_{i=0}^{j} \tbinom{j+2i}{i} \cdot \tfrac{j}{j+2i} \cdot (\tfrac{k-1}{k})^i \cdot (\tfrac{1}{k})^{i+j} \\ &\geq \tfrac{1}{3} \cdot \sum_{i=0}^{j} \tbinom{j+2i}{i} \cdot (\tfrac{k-1}{k})^i \cdot (\tfrac{1}{k})^{i+j} \end{aligned}$$

Further we can lower bound the above sum by its largest term as follows. We use the following fact (cf. [1])

$$\tbinom{n}{\alpha n} \sim 2^{h(\alpha)n} = (\tfrac{1}{\alpha})^{\alpha n}(\tfrac{1}{1-\alpha})^{(1-\alpha)n}$$

where $h(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$ is the binary entropy function. In particular, the two functions

$$\tbinom{(1+2\alpha)j}{\alpha j} \quad \text{and} \quad [(\tfrac{1+2\alpha}{\alpha})^{\alpha} \cdot (\tfrac{1+2\alpha}{1+\alpha})^{1+\alpha}]^j$$

are within polynomial factors of each other. We lower bound the above estimation for q_j by setting $\alpha = \frac{1}{k-2}$,

$$\begin{aligned} q_j &\geq \tfrac{1}{3} \cdot \sum_{i=0}^{j} \tbinom{j+2i}{i} \cdot (\tfrac{k-1}{k})^i \cdot (\tfrac{1}{k})^{i+j} \\ &\geq [(\tfrac{1+2\alpha}{\alpha})^{\alpha} \cdot (\tfrac{1+2\alpha}{1+\alpha})^{1+\alpha} \cdot (\tfrac{k-1}{k})^{\alpha} \cdot (\tfrac{1}{k})^{1+\alpha}]^j \\ &\quad \text{(where } \alpha = \tfrac{1}{k-2}\text{)} \\ &= (\tfrac{1}{k-1})^j \end{aligned}$$

where the last inequality holds up to some polynomial factor. Therefore, up to some polynomial factor, using the binomial theorem, we obtain the following estimate for the success probability p:

$$p \geq (\tfrac{1}{2})^n \sum_{j=0}^{n} \tbinom{n}{j} (\tfrac{1}{k-1})^j = (\tfrac{1}{2}(1 + \tfrac{1}{k-1}))^n$$

Therefore the complexity of k-SAT is within a polynomial factor of $(2(1-\frac{1}{k}))^n$.

Notice that we needed to consider random walks up to length $j + 2i \leq n + 2n = 3n$.

3. Constraint Satisfaction Problems

Notice that we did not use the fact that the clauses are disjunctions of their k literals, they could just as well be any Boolean formulas in k (out of the n) variables. The important point was just that by flipping the right literal value

the Hamming distance to the target assignment decreases by 1. This shows that the algorithm can be easily adapted to solve general constraint satisfaction problems.

A (discrete) *constraint satisfaction problem* consists of the following components:

- A set of n *variables* $x_1, \ldots, x_n$. The variables take values from some finite *domain* D where $d = |D|$. An *assignment* is a tuple of n values from D assigned to the variables.
- A set of *constraints* $C_1, \ldots, C_m$. A constraint is a 0-1-valued function on the domain D^n. For computational purposes a constraint can be *represented* as a formula, a circuit, a finite table, or an algorithm. If $C_j(a_1, \ldots, a_n) = 1$ we say that constraint C_j is *satisfied* by the assignment $(a_1, \ldots, a_n) \in D^n$. If a constraint C_j depends only on l arguments, then it is of *order* l.

The algorithmic task is, given a CSP (its representation), find an assignment that satisfies all constraints (if one exists.)

There is considerable interest in algorithms for constraint satisfaction problems since constraint satisfaction problems occur extremly common (see [7], [4], [2] Chapter 36.) Many NP-complete problems are (or can be formulated as) CSP's. Two popular examples are k-SAT and k-colorability (given a graph, find a coloring of the nodes with k colors such that no adjacent nodes get the same color.) In the case of k-SAT we have $D = \{0, 1\}$, i.e. $d = 2$, and the constraints are of order $l = k$ and are represented by CNF clauses consisting of at most k variables. In the case of k-colorability we have that n is the number of nodes in the graph, $|D| = d = k$, and each edge in the graph gives rise to a constraint C of order 2 where the constraint is satisfied iff the colors assigned to the two nodes of the edge are different.

Since k-SAT and k-colorability are NP-complete problems provided that $k \geq 3$ [6], these examples show that the CSP is NP-hard if either $(d \geq 3, l \geq 2)$ or $(d \geq 2, l \geq 3)$. The case $(d = 2, l = 2)$ is solvable in polynomial time. This case corresponds to (or can be formulated as) a 2-SAT problem and it is known that 2-SAT is in P.

The naive algorithm for a CSP with parameters n (number of variables), d (size of the domain), and l (the order of the constraints) is polynomially related to d^n since one can cycle through all potential assignments from D^n and check for each assignment whether it satisfies all constraints. Even a small improvement in the base value of this exponential function has a significant effect with respect to the size of CSP problems that can be solved within a given time. For example, if the base value d could be lowered to $\sqrt{d}$ then we could solve CSP's of about double the size within the same time.

Again, the algorithmic approach, shown for k-SAT can be carried out. Observe that for a CSP with parameters n, d, l as above, if some constraint of order l is not satisfied by the actual assignment from D^n, then there are $l \cdot (d-1)$ ways of changing the value of one of the variables involved in that constraint. That is, the role of "k" in the above discussion on k-SAT is replaced by "$l \cdot (d-1)$". (Furthermore, the role of "2^n" is replaced by "d^n".)

Following the scheme from above, the corresponding "success probability" can be estimated, up to some polynomial factor, to be at least

$$\begin{aligned} p &\geq \sum_{j=0}^{n} \tbinom{n}{j} (\tfrac{1}{d})^{n-j} (\tfrac{d-1}{d})^j (\tfrac{1}{k-1})^j \\ &= d^{-n} \sum_{j=0}^{n} \tbinom{n}{j} (\tfrac{d-1}{k-1})^j \\ &= d^{-n} (1 + \tfrac{d-1}{k-1})^n \quad (\text{where } k = l \cdot (d-1)\,) \end{aligned}$$

Indeed, for CSP's with $d > 2$ a further improvement is possible. Observe that it can happen that state j transfers to the same state j again in the next step. This means that the (wrong) value of a variable within a constraint that is not satisfied by the actual assignment is changed to another wrong value, thus the Hamming distance to the fixed satisfying assignment does not change. In the worst case, the probability that this happens is $\frac{d-2}{l(d-1)}$. That is, the state j can transfer in any of the states $j-1, j, j+1$ with probabilities $\frac{1}{l(d-1)}, \frac{d-2}{l(d-1)}, \frac{l-1}{l}$, respectively (where this is the worst-case situation.)

We analyze this modified Markov chain by reducing it to the above situation where the only transfers from state i are to state $i-1$ and $i+1$. We map all transitions of the form $i \to i \to \cdots \to i \to i-1$ (and $i \to i \to \cdots \to i \to i+1$) in the new model to a one-step transition $i \to i-1$ ($i \to i+1$, respectively) in the old model. A sequence of transitions of the form $i \to i \to \cdots \to i \to i-1$ occurs with probability

$$\begin{aligned} \tfrac{1}{l(d-1)} \cdot \sum_{k \geq 0} (\tfrac{d-2}{l(d-1)})^k &= \tfrac{1}{l(d-1)} \cdot \tfrac{1}{1 - \frac{d-2}{l(d-1)}} \\ &= \tfrac{1}{(l-1)(d-1)+1} \end{aligned}$$

Therefore, we can use the value $k = (l-1)(d-1)+1$ und plug it in the formula for the success probability calculated above:

$$p \geq d^{-n} \Big(1 + \frac{d-1}{k-1}\Big)^n = d^{-n} \Big(1 + \frac{1}{l-1}\Big)^n$$

The reciprocal value of this probability yields the complexity bound $(d(1-1/l))^n$. In the above calculation we have mapped arbitrarily long random walks in the new model to a single step in the old model. Actually, we can do this only for a bounded number c of steps which means that the number of steps the new Markov chain needs to be run is $3cn$ instead of $3n$. By choosing c large enough, one can get arbitrarily close to the value calculated above. Therefore, we can take this into account by adding a "$+\varepsilon$" to the base of the exponential function.

Theorem: For $d = 2$ there is an algorithm for the CSP with complexity polynomially related to

$$\Big(2(1-\frac{1}{l})\Big)^n$$

For $d > 2$ and any $\varepsilon > 0$ there is an algorithm for the CSP with complexity at most

$$\Big(d(1-\frac{1}{l})+\varepsilon\Big)^n$$

In the special case of $l = 2$ our bound $(\frac{d}{2}+\varepsilon)^n$ is beaten by Beigel and Eppstein's probabilistic approach [3] which essentially proceeds as follows: for each variable x_i randomly guess a 2-element subset $D_i \subseteq D$. Then, solve the restricted CSP that chooses x_i only from the domain D_i. This restricted CSP can be formulated as a 2-SAT problem and therefore can be solved in polynomial time. Given that the CSP has a solution, with probability $\frac{2}{d}$ the random guess for variable x_i is OK in the sense that D_i contains the correct value for variable x_i. Therefore, the complexity of this probabilistic algorithm is within a polynomial factor of $(\frac{d}{2})^n$.

The following table shows some numerical values for the CSP.

	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$
$d=2$	1	1.334	1.5	1.6	1.667
$d=3$	1.5	2.001	2.251	2.401	2.501
$d=4$	2	2.667	3.001	3.201	3.334
$d=5$	2.5	3.334	3.751	4.001	4.167
$d=6$	3	4.001	4.501	4.801	5.001

It was mentioned that the k-colorability problem is a special case of the CSP with $l = 2$. The best known algorithms for k-colorability are indeed somewhat better than our general CSP bounds, see [18] and [3].

4. Summary

We have presented a general algorithm to solve a CSP which improves upon the naive exponential-time algorithm considerably, and will therefore have some practical relevance. In the special case of 3-SAT the achieved complexity bound is the best known to date. The algorithm is another example with respect to Pudlak's proposal [15] to find algorithms for (k-)SAT which are not instantiations of the Davis-Putnam procedure, or more general, which are not algorithmic versions of previously known logical calculi.

Indeed, the algorithm is an example for a new paradigm in random algorithm design. Although the Markov chain approach is well known (cf. [11]) in random algorithm design, usually it is the stationary distribution and the rapid mixing property that is of interest. In our algorithm we do not intend to reach the stationary distribution, but rather it is intended that the algorithm reaches a relatively unlikely state rather quickly (if at all) which constitutes the solution of the problem. If the Markov chain does not reach this state whithin the first few steps, we start the process all over again.

5. Acknowledgements

For valuable remarks and discussions I want to thank V. Arvind, S. Baumer, M. Bossert, J. Köbler, P. Pudlák, and T. Thierauf.

6. References

[1] R.B. Ash: *Information Theory*. Dover 1965.

[2] M.J. Attallah (ed.): *Algorithms and Theory of Computation Handbook*. CRC Press 1999.

[3] R. Beigel, D. Eppstein: 3-coloring in time $O(1.33446^n)$: a no-mis algorithm. *Proceedings 36th Annual Sympos. on Foundations of Computer Science 1995*, pp. 444–452. ECCC Technical Report TR95-33 (1995).

[4] L. Bolc, J. Cytowski: *Search Methods for Artificial Intelligence*. Academic Press 1992.

[5] W. Feller: *An Introduction to Probability Theory and Its Applications*. Wiley, 1968.

[6] M.R. Garey, D.S. Johnson: *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman 1979.

[7] J. Gu, P. Purdom, J. Franco, B. Wah: *Algorithms for the Satisfiability Problem*. Cambridge University Press, to appear.

[8] O. Kullmann: A systematic approach to 3-SAT-decision, yielding 3-SAT-decision in less than 1.5045^n steps. *Theoretical Computer Science*, to appear.

[9] S. Minton, M.D. Johnston, A.B. Philips, P. Laird: Minimizing conflicts: a heuristic repair method for constraint

satisfaction and scheduling problems. *Artificial Intelligence* 58 (1992) 161–205.

[10] B. Monien, E. Speckenmeyer: Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics* 10 (1985) 287–295.

[11] R. Motwani, P. Raghavan: *Randomized Algorithms*. Cambridge University Press 1995.

[12] C.H. Papadimitriou: On selecting a satisfying truth assignment. *Proceedings of the 32nd Ann. IEEE Symp. on Foundations of Computing*, 163–169, 1991.

[13] R. Paturi, P. Pudlák, F. Zane: Satisfiability coding lemma. *Proceedings 38th IEEE Symposium on Foundations of Computing* 1997, 566–574.

[14] R. Paturi, P. Pudlák, M.E. Saks, F. Zane: An improved exponential-time algorithm for k-SAT. *Proceedings 39th IEEE Symposium on Foundations of Computing* 1998, 628–637.

[15] P. Pudlák: Satisfiability - Algorithms and Logic. *Proceedings Mathem. Foundations of Computer Science (MFCS)* 1998. Lecture Notes in Computer Science 1450, Springer-Verlag 1998, 129–141.

[16] R. Rodosek: A new approach on solving 3-satisfiability. *Proc. 3rd Intern. Conf. on AI and Symbolic Math. Computation*. 197–212, LNCS 1138, Springer-Verlag 1996.

[17] I. Schiermeyer: Solving 3-satisfiability in less than 1.579^n steps, *Selected papers from CSL 92*, LNCS 702, Springer-Verlag 1993, 379–394.

[18] I. Schiermeyer: Deciding 3-colourability in less than $O(1.415^n)$ steps. *19th Int. Workshop on Graph-Theoretical Concepts in Computer Science*. Springer-Verlag, 1994, 177-182.

[19] I. Schiermeyer: Pure literal look ahead: an $O(1.497^n)$ 3-satisfiability algorithm. *Proc. of the Workshop on Satisfiability*, Universita degli Studi, Siena, Italy, pages 63–72, 1996.

[20] U. Schöning: On the Complexity of Constraint Satisfaction Problems. Ulmer Informatik Berichte Nr. 99-03, Universität Ulm, 1999.