On the Complexity of 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 quite simple probabilistic algorithm for solving general constraint satisfaction problems (CSP). Given a CSP with n variables, each variable taking at most d values, and constraints of order l, the complexity of our algorithm is within a polynomial factor of $\left(\frac{d \cdot l}{l+1}\right)^n$.

As an application, this yields an $O\Big((\frac{2k}{k+1})^n \cdot poly(n)\Big)$ algorithm for k-SAT which is among the fastest known algorithms for k-SAT.

As another application, this yields an $O\Big((\frac{2k}{3})^n \cdot poly(n)\Big)$ algorithm for k-colorability.

1 Introduction and Notation

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.

*This work was supported by a DFG grant.

- 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], [3] Chapter 36). Many NP-complete problems are (or can be formulated as) CSP's. Two popular examples are k-SAT (find a satisfying assignment for a given formula in k-CNF) 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.

In the following we want to compare certain exponentially growing functions, and at the same time we want to ignore polynomial factors which do not influence the overall behavior of the function. Therefore we use the following definition. Say functions $f(n)$ and $g(n)$ are *polynomially related*, if there is a polynomial $p(n)$ such that for all but finitely many n,

$$\frac{1}{p(n)} \cdot g(n) \;\leq\; f(n) \;\leq\; p(n) \cdot g(n)$$

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.

In this paper we present a quite simple probabilistic algorithm that solves all CSP's in time which is polynomially related to $\left(\frac{d \cdot l}{l+1}\right)^n$. For CSP's with constraints of small order l, this is an significant improvement. Applied to the case of k-SAT this gives an algorithm for k-SAT of complexity $O\left((\frac{2k}{k+1})^n \cdot poly(n)\right)$ which is within the range of the best known algorithms for k-SAT (see [10, 12, 13]). Our approach is totally different than any of these cited papers. In the case of k-colorability this gives an algorithm of complexity $O\left((\frac{2k}{3})^n \cdot poly(n)\right)$.

The idea of our algorithm follows a simple and well known paradigm in heuristic search algorithm design (see e.g. [9]): produce some randomly generated initial assignments and guided by those constraints that are not yet satisfied try to "repair" the assignment by a certain number of modifications. In the following we present a theoretical analysis which shows how to choose the parameters of this approach (the number of initial assignments versus the number of modifications) optimally such that on the one hand "completeness" is achieved (i.e. the whole search space is covered) and on the other hand the complexity of the algorithm is minimized.

2 Searching for a satisfying assignment within a fixed Hamming distance

Suppose we are given an initial assignment $a \in D^n$ where n is the number of variables. Let $d = |D|$. There are

$$\sum_{i=0}^{\beta n} \binom{n}{i} (d-1)^i$$

many assignments within Hamming distance βn from a. (We assume $0 \leq \beta \leq 1$ and that βn is an integer). This is the set of assignments (called the Hamming sphere around a) which can be obtained from a by changing at most βn many values in $a = (a_1, \ldots, a_n)$.

This number can be bounded as follows (see [8], Chapter X, or [2], page 121)

$$[8n\beta(1-\beta)]^{-\frac{1}{2}} \cdot 2^{h(\beta)n}(d-1)^{\beta n} \;\leq\; \sum_{i=0}^{\beta n} \binom{n}{i}(d-1)^i \;\leq\; 2^{h(\beta)n}(d-1)^{\beta n}$$

where

$$h(\beta) = -\beta \log_2 \beta - (1-\beta)\log_2(1-\beta)$$

is the binary entropy function.

These estimations show that for constant β the value of the expression $\sum_{i=0}^{\beta n} \binom{n}{i}(d-1)^i$ is polynomially related to $\left(2^{h(\beta)}(d-1)^{\beta}\right)^n$.

An important feature of our algorithm is that, given an initial assignment a, it is not necessary to search through this number of assignments that we estimated above for to find some satisfying assignment within Hamming distance βn (if one exists). The given CSP can be used to prune the tree of assignments that has to be searched. Let C be a constraint that is not satisfied under the actual assignment a. The constraint C has order l, i.e. it depends only on l variables, say $x_{i_1}, x_{i_2}, \ldots, x_{i_l}$. At least one of the assignments to these variables has to be changed to (possibly) achieve an assignment that satisfies all constraints.

The following recursive backtracking procedure implements this idea:

procedure *test* (a , m): **boolean**
{ Here a is an assignment and $m \leq n$ is an integer.
The procedure returns true if and only if an
assignment exists that satisfies all constraints and that
can be obtained from a by changing at most m values}
if a satisfies all constraints **then return true**
  else if $m = 0$ **then return false**
let C be some constraint that is not satisfied by a
suppose C depends on the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_l}$
for $j := 1$ **to** l **do**
  for $p := 1$ to $d-1$ do
    if *test* ($a|_{i_j,p}$, $m-1$) **then return true**
return false

Here, $a|_{i_j,p}$ denotes the assignment that is obtained from $a = (a_1, \ldots, a_n)$ by changing the value a_{i_j} to $a_{i_j} + p \pmod d$. Here we suppose that $D = \{0, 1, \ldots, d-1\}$.

Suppose there exists at least one satisfying assignment a^* of the CSP within Hamming distance m of the initial assignment a. If $m = 0$, i.e. $a = a^*$, this is detected in the beginning of the procedure. If $m > 0$ and a does not yet satisfy all constraints of the CSP so that C is some constraint not being satisfied by a, then at least one of the assignments to the variables on which C depends has to be changed. Say, the value of a_{i_j} has to be changed such that the Hamming distance between a and a^* decreases by 1. Therefore, by induction hypothesis, at least one of the recursive calls $test\,(\,a|_{l_i,p}\,,\,m-1\,)$ for $p = 1, \ldots, d-1$ will return true. This proves the correctness of the procedure.

By the fact that the recursion depth is bounded by m, and the number of recursive calls is at most $l(d-1)$, the recursion tree has therefore at most $(l(d-1))^m$ many leaves, and the complexity of the procedure is polynomially related to $(l(d-1))^m$.

Observe that, depending on the choice of the value $m = \beta n$, the value $(l(d-1))^{\beta n}$ can be much smaller than $\left(2^{h(\beta)}(d-1)^\beta\right)^n$. (Example: let $\beta = 1/4$ and suppose $d = 3$ and $l = 3$. Then we get $(l(d-1))^{\beta n} \approx 1.57^n$ and $\left(2^{h(\beta)}(d-1)^\beta\right)^n \approx 2.09^n$.)

3 Algorithm Analysis

Now the entire algorithm consists of an outer loop which selects appropriate (or randomly chosen) assignments a such that the whole search space D^n is finally covered by the (βn)-Hamming spheres around the a's (with high probability). Within each loop, a call of $test(a, \beta n)$ is then performed. The complexity of the algorithm (up to a polynomial factor) is obtained by multiplying the necessary number of a's with the term $(l(d-1))^{\beta n}$ that we obtained above.

Ideally, the set of a's constitutes a perfect code (cf. [8]), i.e. all the βn-Hamming spheres are mutually disjoint and the whole space D^n is covered. (What

actually is sufficient here is a *covering code* [5]). In this case, the number of a's is

$$\frac{d^n}{\sum_{i=0}^{\beta n} \binom{n}{i}(d-1)^i}$$

This corresonds to the well known sphere bound from Coding Theory [5, 3, 8]. According to our above estimations this expression is polynomially related to

$$\Big(\frac{d}{2^{h(\beta)}(d-1)^\beta}\Big)^n$$

Therefore, the overall complexity of the algorithm is polynomially related to

$$\Big(\frac{d}{2^{h(\beta)}(d-1)^\beta}\Big)^n \Big(l(d-1)\Big)^{\beta n} \;=\; \Big(\frac{d\cdot l^\beta}{2^{h(\beta)}}\Big)^n$$

The base value of this exponential function is minimized by the choice $\beta = \frac{1}{l+1}$. Substituting this into the expression yields that the overall complexity, by this choice of β, can be bounded by

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

This corresponds to the complexity in the ideal case that the set of initial assignments forms a perfect code.

Next we analyze how often we need to guess a *random* assignment a such that with high probability, the βn-spheres around the a's cover all of D^n. (We keep the above calculated choice $\beta = \frac{1}{l+1}$). The probability that a single random assignment a lies within Hamming distance βn of a fixed satisfying assignment a^* is

$$\frac{\sum_{i=0}^{\beta n} \binom{n}{i}(d-1)^i}{d^n}$$

By the above estimations, this probability is at least

$$\sqrt{\frac{l+1}{8n}}\cdot\Big(\frac{d}{2^{h(\beta)}(d-1)^\beta}\Big)^{-n}$$

Suppose we randomly and independently choose t many starting assignments a. The probability that a^* is *not* within Hamming distance βn with any of the a's is at most

$$\Big(1-\sqrt{\frac{l+1}{8n}}\cdot\Big(\frac{d}{2^{h(\beta)}(d-1)^\beta}\Big)^{-n}\Big)^t$$

Using $1-x \leq e^{-x}$ it can be seen that for to achieve an acceptable error probability of, say e^{-20} or even e^{-n}, it is enough to choose

$$t = 20 \cdot \sqrt{\frac{8n}{l+1}} \cdot \left(\frac{d}{2^{h(\beta)}(d-1)^{\beta}}\right)^n \quad \text{or} \quad t = n \cdot \sqrt{\frac{8n}{l+1}} \cdot \left(\frac{d}{2^{h(\beta)}(d-1)^{\beta}}\right)^n,$$

respectively. That is, the number of random assignments that has to be chosen is within a polynomial factor of the ideal number of assignments in a perfect code.

Therefore, we have demonstrated that the complexity of the entire probabilistic algorithm is polynomially related to $\left(\frac{d \cdot l}{l+1}\right)^n$.

4 Discussion

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 k-SAT the achieved complexity bound is within the range of the best known algorithms for this problem [10, 12, 13]. The algorithm is another example with respect to Pudlak's proposal [14] to find algorithms for (k-)SAT which are not instantiations of the Davis-Putnam procedure.

The algorithm is probabilistic because the set of centers of the Hamming spheres that have to be searched (deterministically) are chosen randomly. By this we introduce a new paradigm for probabilistic algorithms. We have not seen this concept before in the context of probabilistic algorithms (see [11]) (although, somewhat related is Shannon's probabilistic construction of an error correcting code in the proof of the main theorem of Coding Theory [2, 8]). A somewhat similar philosophy is in the randomized "color-coding" algorithm from [1]: first a random structure is selected. The probability that the structure is "good", in some sense, should not be too low. Then, using this structure, under the condition that the structure is "good", the second (deterministic) part of the algorithm can prune the search for the solution.

It would be interesting to find a *deterministic* way of picking the centers of the Hamming spheres such that the whole search space is covered. The complexity of such a potential deterministic algorithm will asymptotically, within a polynomial factor, not be better than our probabilistic algorithms, as we have seen.

In some sense, half-way to a total derandomization of the algorithm is an approach that reduces the number of necessary random bits, e.g. from exponential to polynomial. Indeed this can be achieved. In [5] it is shown that almost every linear code asymptotically achieves the sphere bound. That is, it is enough to randomly guess a code matrix (of appropriate dimensions) and then use the linear code defined by this code matrix. With high probability the Hamming spheres cover the total search space. Apart from this probabilistc "precomputation" the rest of the algorithm is deterministic in this case. One just needs to cycle systematically through all codewords of this linear code.

Acknowledgements

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

References

[1] N. Alon, R. Yuster, U. Zwick: Color-coding. *26th Symp. on Theory of Computing*, ACM 1994.

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

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

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

[5] G. Cohen, I. Honkala, S. Litsyn, A. Lobstein: *Covering Codes*. North-Holland 1997.

[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] F.J. MacWilliams, N.J.A. Sloane: *The Theory of Error-Correcting Codes*. North-Holland 1983.

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

[13] 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.

[14] 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.