Solving Hard Combinatorial Problems with GSAT – a Case Study

Holger H. Hoos

Technische Hochschule Darmstadt, Fachgebiet Intellektik,
D-64289 Darmstadt, Germany
email: hoos@intellektik.informatik.th-darmstadt.de

Abstract. In this paper, we investigate whether hard combinatorial problems such as the Hamiltonian circuit problem HCP (an $\mathcal{NP}$-complete problem from graph theory) can be practically solved by transformation to the propositional satisfiability problem (SAT) and application of fast universal SAT-algorithms like GSAT to the transformed problem instances. By using the efficient transformation method proposed by Iwama and Miyazaki in 1994, one obtains a class of SAT-problems which are especially hard for SAT-algorithms based on local search such as GSAT. We identify structural properties of these problems which are responsible for GSAT's poor performance on this problem class. An empirical analysis indicates that several methods which have been designed to improve GSAT on structured problems are not effective for SAT-transformed HCP-instances.

1 Introduction

One of the basic paradigms of AI states that many problems from different domains can be formulated as logical problems and therefore general methods for solving logical problems can be used for their solution. Actually, this idea provides strong motivation for developping and studying algorithms for general logical problems like the satisfiability problem in propositional logic (SAT). This general approach seems particularly interesting for computationally hard (i.e. $\mathcal{NP}$-hard) problems for which no specialized, efficient algorithms are available.

In this paper, we examine whether GSAT, an incomplete, model-generating algorithm for SAT, can be practically used to solve hard instances of the *Hamiltonian circuit problem (HCP)*, a combinatorial problem from graph theory. For a given graph G, the HCP is the task of finding a cyclic tour (Hamiltonian circuit) in G that visits every vertex exactly once. This problem is $\mathcal{NP}$-complete (like SAT) and therefore, unless $\mathcal{P} = \mathcal{NP}$, not solvable by polynomial-time algorithms [10]. Any instance G of HCP can be transformed into a propositional CNF-formula α in such a way that the satisfiabilty of α is equivalent to the existence of a solution to the HCP, and the models of α correspond directly to the Hamiltonian circuits in G. Thus it is principally possible to use model-generating SAT-algorithms (like GSAT) for finding Hamiltonian circuits in graphs [7].

To practically realize this approach, one has to assure that the size of the original problem is not growing too much as a result of the SAT-transformation.

Furtheron, the time complexity of the transformation must be small wrt. the time complexity of the problem itself. Transformations satisfying these conditions shall be called "efficient transformations" in the sequel [7]. Efficient SAT-transformations are currently available for several graph problems, including HCP [7]. Principally the SAT-problems generated by these methods show an order of magnitude which seems to be managable by powerful SAT-algorithms like GSAT. Thus it makes sense to examine the idea of solving HCP-instances by the means of algorithms for the general SAT-problem in more detail.

The remainder of this paper is structured in the following way: In the next section we will present an efficient SAT-transformation for HCP due to Iwama and Miyazaki [7]. Furtheron we describe the GSAT algorithm [14] and some of its variants which have been successfully applied to various classes of SAT-problems. In Sect.3 we consider the application of GSAT to the HCP; in particular, we show that due to certain structural properties of the SAT-transformed HCP-instances these problems can be expected to be rather difficult for GSAT despite the efficient transformation method beeing used. The results of an experimental study of GSAT's performance on SAT-transformed HCP-instances presented in Sect.4 confirm this conjecture and show, which extensions of the basic GSAT algorithm can be used to improve its performance on the given problem class. Finally, in Sect.5 we sum up our results and discuss their implications on the underlying, general approach.

2 Preliminaries

In this section we first describe, how HCP-instances can be efficiently transformed into SAT-instances. Then we present the GSAT algorithm, a powerful, model-generating algorithm for the SAT-problem which has been applied successfully to many classes of SAT-problems.

2.1 Efficient SAT-Transformation for HCP

In 1994, efficient SAT-reductions of several graph problems including HCP were proposed by Iwama and Miyazaki [7]. The basic idea for transforming instances of HCP into CNF formulae is to focus on permutations of the given graph's vertices. This is done by assigning an integer value to each vertex, which defines its position in the permutation. The function which assigns such a *vertex position* to each vertex of the graph is called a *vertex assignment*. A vertex assignment encodes a permutation of the vertices, iff the following conditions hold:

- every vertex position is *valid* (i.e. no position is greater than the total number of vertices in the graph),
- the vertex positions are pairwise different (i.e. identical positions are never assigned to different vertices).

A vertex assignment codes for a Hamiltonian circuit of the given graph, iff

- it codes for a permutation of the vertices,

– nodes with successive positions in this permutation are always connected by an edge,
– there is an edge connecting the first vertex of the permutation to the last one.

This characterization can be used to transform any given instance of HCP into a constraint satisfaction problem (CSP) on integers (i.e. vertex positions). But now there is a straightforward SAT-transformation at hand: one simply has to code every vertex position by a number of propositional variables using the binary representation of integers. Thereby, every contraint on vertex positions can be translated into a CNF-clause which finally transforms the integer CSP into a SAT problem.

To give the formal description of the transformation algorithm, we first need some definitions: Let $G = (V, E)$ be a graph with n vertices, vertex set $V = \{v_0, \ldots, v_{n-1}\}$ and edge relation $E \subseteq V \times V$. To every vertex v_i $(0 \leq i \leq n-2)$ a set of $k = \lceil \log_2(n-1) \rceil$ propositional variables is assigned, which are denoted by $v_{i,1}, \ldots, v_{i,k}$. Since using this approach, Hamiltonian circuits are represented by cyclic permutations of the vertices, the position of vertex v_{n-1} can be fixed to $n-1$ (with 0 being the first position) without loss of generality. Furtheron we define $C(i,l)$ as the CNF-clause containing propositional variables $v_{i,1}, \ldots, v_{i,k}$ in such a way, that $C(i,l)$ evaluates to false iff position l is assigned to vertex i. The conjunction of two clauses c_1, c_2 is denoted by $c_1 \cup c_2$.

Efficient SAT-Reduction of HCP (Iwama and Miyazaki, 1994)

Step 1: For every vertex v_i $(0 \leq i \leq n-2)$ and every integer l $(n-1 \leq l \leq 2^k - 1)$ generate the clause $C(i,l)$. (The set of these clauses is satisfied iff a valid position is assigned to each vertex.)

Step 2: For every pair of vertices v_i, v_j $(0 \leq i, j \leq n-2)$ and every integer l $(0 \leq l \leq n-2)$ generate the clause $C(i,l) \cup C(j,l)$. (This set of clauses is satisfied iff all valid vertex positions are pairwise different.)

Step 3: For every pair of vertices v_i, v_j $(0 \leq i, j \leq n-2)$ with $(v_i, v_j) \notin E$ and every integer l $(0 \leq l \leq n-2)$ generate the clause $C(i, l-1) \cup C(j,l)$. (This set of clauses is satisfied iff any pair of vertices with successive positions are connected by an edge).

Step 4.1: For every vertex v_i $(0 \leq i \leq n-2)$ with $(v_i, v_{n-1}) \notin E$ generate the clause $C(i, n-2)$.

Step 4.2: For every vertex v_i $(0 \leq i \leq n-2)$ with $(v_{n-1}, v_i) \notin E$ generate the clause $C(i, 0)$. (Steps 4.1 and 4.2 are special cases of Step 3 for vertex v_{n-1}, to which implicitly always the position $n-1$ is assigned; cf. above).

For any given graph G with n vertices this algorithm generates a CNF-formula α with $(n-1)\lceil \log_2(n-1) \rceil$ propositional variables which is satisfiable iff G contains a Hamiltonian circuit. Furtheron the models of α directly correspond to the solutions of the HCP. The algorithm has polynomial time complexity (wrt. n) and can be applied to both directed and undirected graphs. Thus we get:

Result 1. *For the HCP, an efficient SAT-transformation is given by the Iwama-Miyazaki algorithm.*

A simple argument shows, that one cannot expect to find SAT-transformations of HCP using significantly fewer propositional variables (see [7] for details). The CNF-formulae which are obtained from HCP-instances using this algorithm will be referred to as *HC-formulae* in the sequel.

2.2 GSAT – a Fast, Incomplete SAT-Algorithm

GSAT is a heuristic SAT-algorithm proposed in 1992 by Selman, Levesque and Mitchell [14] which is based essentially on local search in the space of all variable assignments of the input CNF-formula. Starting with a randomly chosen assignment A, in every step of the procedure the value of one propositional variable is flipped, i.e. changed from true to false or vice versa. The selection of the variable to be flipped is based on the difference in the number of currently satisfied clauses the flip would cause. This value, which is positive for those flips that cause an increase in the number of satisfied clauses, is called the *score* of a variable. In every step, a variable with maximal score is flipped; if there are several variables with maximal score, one of these is randomly selected.

Thus, GSAT basically performs a kind of local hillclimbing on the number of satisfied clauses, which can be viewed as the global optimization of an objective function. Of course, the models of the input fomula are exactly those assignments satisfying all clauses simultanously. When such an assignment is reached, the GSAT algorithm terminates. If after a given number maxSteps of flips no solution is found, the algorithm restarts its local search by randomly choosing a new variable assignment. If after a given number maxTries of these tries still no model is found, GSAT terminates. The standard GSAT procedure can be formalized as follows:

```
procedure GSAT(α)
  for i := 1 to maxTries do
    A := randomly chosen assignment of the variables appearing in α;
    for j := 1 to maxSteps do
      if A satisfies α then return(B);
      else
        x := randomly chosen variable of α with maximal score
             under the current assignment A;
        flip value of x in A;
        end if;
      end for;
    end for;
  return (''no model found!'');
  end GSAT;
```

Note, that due to the random choices for the initial assignment and between equally good flips, GSAT is a non-deterministic algorithm. Another important property of GSAT is its incompleteness: However the parameters maxTries and maxSteps are chosen, if the input formula is satisfiable, there is no guaranty that GSAT finds a model. So in contrast to complete SAT-algorithms like the Davis-Putnam procedure (DP), GSAT can never determine the *unsatisfiability* of a given formula. However, it has been shown, that variants of the GSAT algorithm can outperform complete systematic search methods like DP on large classes of SAT-problems.

Another difficulty with local search methods like GSAT is that they can get "stuck" in local optima of the objective function which do not represent solutions. To cope with that, several techniques have been proposed, the most basic of these being the restart mechanism, which is realized in the basic algorithm as stated above.

An important extension of the basic algorithm is the "random walk" strategy [13]. Here, with a fixed probability walkProb-instead of a standard hillclimbing-step a so-called "walk-step" is performed; this is done by randomly selecting one of the variables appearing in currently unsatisfied clauses. Therefore, by each walk-step at least one formerly unsatisfied clause becomes satisfied. In contrast to ordinary hillclimb-steps, a walk-step may cause a decrease in the total number of satisfied clauses even if improving flips would have been possible. In consequence, by using random walk the local search becomes somewhat less directed; therefore the chances of escaping from local optima of the objective function are better than in the case of basic GSAT. Experimental analysis has shown, that for most problem classes the random-walk strategy significantly improves GSAT's performance [13].

Another GSAT variant called "clause weighting" is based on the idea of modifying the objective function in such a way that local optima are "levelled" [12]. This is done by assigning an integer value (weight) to each clause. When starting GSAT, all weights are initialized to the same value. Whenever an unsuccessful try is aborted, the weights of all currently unsatisfied clauses are increased by a constant value. The evaluation of the variable scores is modified such that a clause with weight w is treated like a set of w identical unweighted clauses in standard GSAT. This mechanism ensures that clauses which frequently have been found to be unsatisfied at the end of an unsuccessful try get a higher priority in succeeding tries. It has been shown that particularly for classes of structured problems (like SAT-encoded planning problems) this method causes significant improvement of GSAT's performance [11].

3 Applying GSAT to the HCP

From a theoretical point of view, the situation looks quite promising: By the Iwama-Miyazaki algorithm one can transform HCP-instances with n nodes into SAT problems with $n \log_2 n$ propositional variables. The GSAT algorithm has shown to be capable of solving SAT problems up to 500 variables in reasonable time, so one could expect to practically solve HCP-instances with approximately 50 variables by applying GSAT to the transformed problem instances. However, for several reasons this conjecture does not hold; as we will see, the transformed HCP-instances yield a class of SAT-problems which are very hard for local search algorithms like GSAT.

Since it is a well-known fact that the performance of SAT algorithms can depend heavily on the structure of a given class of problems, a careful characterization of the SAT-tranformed HCP-instances should be a good starting point. As we already know, the Iwama-Miyazaki method transforms any HCP-instance with n nodes into a CNF-formula with $(n-1)\lceil\log_2(n-1)\rceil$ propositional variables. But what about the number of clauses? Examination of the transformation algorithm shows that in steps 1, 4.1 and 4.2 at least $\Theta(n)$ and at most $\Theta(n^2)$ clauses with length $k = \lceil\log_2(n-1)\rceil$ are generated. Out of steps 2 and 3 one gets $\Theta(n^3)$ clauses of length $2k$, where the exact number of clauses depends on the number of vertices n and the edge-relation E of the given graph G.

Several authors claim that in general, a large ratio of clauses/variables might even improve GSAT's performance on a given class of formulae [3, 4]. Thus the situation doesn't look too bad for the HC-formulae under consideration. But in practice, the cubic length of the formula (wrt. n) may cause storage problems even for HCP-instances of modest size. This point is stressed by a simple example: Transforming a specific HCP-instance with 50 vertices by the given algorithm, one obtains a CNF-formula of 2,023,578 literals containing 294 variables and 169,047 clauses. Therefore, since the corresponding graph has a vertex degree of 1 (actually being one single, big Hamiltonian circuit) and seen under a practical point of view, we observe an almost dramatic increase in problem size caused by the transformation. As we will see, the situation is even worse, since in addition to this cubic blowup in problem size, the single steps of the GSAT-algorithm are slowed down significantly by another structural property of the HC-formulae.

To see why this happens, we have to look at the evaluation of variable scores in GSAT in more detail. For many problem classes (like Random-k-SAT [4]), a flip of one variable usually does *not* affect the scores of *all* other variables. On these problems the performance of GSAT can be significantly improved by using the following method of scoring (similar to the one described in [2]), which we will call "score-by-need": At the begin of each try, after selecting the initial assignment, the scores for all flips are evaluated and memorized. After each step of the algorithm (i.e. after each flip) now only the scores of those variables have to be updated, which share any clause with the flipped one [15]. Here a number of propositional variables is said to *share a clause* c, iff all of them appear in c. Variables which share at least one clause are said to *depend* on each other. So the actual number of scores which have to be evaluated in each step is determined by the variable-dependencies of the given formula.

Using this method, the average-case time complexity of each GSAT-step can be reduced from $\mathrm{O}(n^2)$ to $\mathrm{O}(1)$ for Random-3-CNF-formulae with constant clauses/variable-ratio [1]. Unfortunately, the score-by-need method is not effective for HC-formulae, because these show a property we call *complete variable-dependency*: in these formulae any variable depends on all other variables. In consequence, using the score-by-need method for problems with complete variable-dependency, after each flip one essentially has to evaluate the scores of all variables anew. Thus, as in contrast to problems with *sparse* variable-dependency, as Random-3-CNF with constant clause/variables-ratio [4], the score-by-need method cannot be used to improve the time efficiency of GSAT-steps for problem classes displaying this property.

Another structural property of HC-formulae is given by the fact, that any two variables assigned to the same vertex share much more clauses than variables belonging to different vertices. (Note, that all clauses encode constraints on vertex positions and therefore contain all the variables belonging to the respective vertices.) Consequently, when applying GSAT to this problem class, one should expect to obtain improved performance by using extensions of the basic algorithm which have been shown to be particularly effective for classes of structured problems (such as clause weights, [11]).

Several subclasses of the HCP greatly differ with regard to their time complexity. One important parameter is the vertex degree of the given graph (i.e. the number of edges entering resp. leaving each vertex). Since in the experimental part of this work, the HCP on directed graphs (digraphs) has been examined, we concentrate on the vertex out-degree here. Obviously, for digraphs with out-degree 1 the HCP has linear time complexity. To see this, consider the following algorithm: Start at an arbitrary vertex and repeatedly follow the single edge leaving the vertex you have visited last. Remember all the vertices you have visited; when you arrive at any vertex you have already visited, your tour is a Hamiltonian circuit iff it contains all vertices of the given graph.

However, for out-degree 2 the HCP is already $\mathcal{NP}$-complete.[1] So the vertex out-degree is an important parameter of the HCP in digraphs. Anyway, after transforming a given HCP-instance to SAT using the Iwama-Miyazaki method, the information about the digraph's vertex-out degree is no longer represented explicitly. On the contrary, there is no obvious difference between CNF-formulae corresponding to HCP-instances with out-degree 1 and 2. So one should be doubtful whether the simplicity of HCP-instances with out-degree 1 is conserved by the SAT-transformation. Indeed it would be staightforward to assume, that the performance of a SAT-algorithm on a class of transformed problem instances depends strongly on the transformation method used.

Considering application of GSAT or similar local search methods, there is one more important aspect about the structure of intances taken from these problem classes: the avarage number of solutions per instance. Many researchers agree, that problems having very few solutions are typically very hard for local search algorithms like GSAT (see for instance [3]). Obviously, for HCP-instances with vertex out-degree 1 there exists either one solution or no solution at all. Solvable instances with out-degree 2 may have several solutions, but as our experiments have shown, these typically also have only one solution. However, instances of both problem classes have usually a large number of near-solutions (i.e. assignments satisfying *almost* all conditions encoded by the clauses). This observation leads to the conjecture that the HC-formulae corresponding to the mentioned subclasses of the general HCP should be particularly difficult for GSAT-like SAT-algorithms, whereas SAT-transformed HCP instances with larger vertex out-degree, which usually have a greater number of solutions, should be solved more efficiently. Summing up, we get:

---

[1] This follows immidiatly from the standard prove for $\mathcal{NP}$-completeness of the general HCP, as can be found in [10].

Result 2. *HC-formulae should be expected to be very difficult for GSAT for several reasons: The large number of clauses can easily cause storage problems; due to complete variable-dependency the score-by-need method for efficient score evaluation is not effective; the inherent easiness of HCP-instances with vertex out-degree 1 might get lost by the transformation process; and problem instances with small vertex out-degree have only very few solutions but a large number of near-solutions.*

4 Experimental Results

In this section, we present and discuss the results of our experimental work, in which we applied variants of the GSAT algorithm to several sets of SAT-transformed HCP-instances. We use an empirical approach based on a method which is frequently applied in the analysis of SAT-algorithms (see [6, 1]). Since no suitable benchmark problems were available, we first generated several sets of solvable instances.[2] On these test-sets we evaluated the performance of several variants of the GSAT-algorithm (implemented as portable C-programs). As performance criteria we used the success rate $sr = st/tt$ (where st denotes the number of successful tries and tt the total number of tries for a specific experiment) and the average number of steps (i.e. flips) per successful try denoted by ss.[3] Measuring relative instead of absolute execution time (i.e. steps of the algorithm instead of CPU-time) has the great advantage of abstracting from specific properties of the implementation and the runtime environment and thus allows to concentrate on the behaviour of the algorithm itself. To capture these aspects, we introduce τ_s, defined as the average CPU-time needed to execute one single step of GSAT. Typical values for τ_s^{-1} (steps per second) measured for $\mathrm{HC}_1(n)$-Problems using our GSAT implementation on a DEC Alpha 5/250 machine range from 500 for $n = 10$ to 4.29 for $n = 50$.[4]

Because of the reasons stated above, the following subclasses of HCP were being considered for experimental analysis:

$\mathrm{HC}_1(n)$: solvable instances with n vertices and out-degree 1
$\mathrm{HC}_2(n)$: solvable instances with n vertices and out-degree 2

As pointed out before, one should expect that these classes are different wrt. their difficulty for GSAT-like algorithms. For various problem sizes n between 10 and 30, test-sets for these classes were randomly generated by constructing first a Hamiltonian circuit of the given size and afterwards adding randomly selected edges, until every vertex had exactly the specified out-degree. To abstract from random differences between the $\mathrm{HC}_2(n)$ instances, the test-sets for these classes consisted each of 100 problem instances.

[2] Since GSAT is an incomplete algorithm, it doesn't make much sense to apply it to potentially unsolvable problem instances for evaluation purposes.

[3] We consider only successful tries, because in unsuccessful tries always exactly maxSteps steps are performed.

[4] One should be able to improve the performance of our implementation by further optimizing the program code.

4.1 GSAT with Random Walk

From the experimental work of other authors we know that GSAT's performace can be significantly increased on a large number of problem classes with different structural properties by using the random walk strategy [13]. Since early in our experimental work it became clear, that this also holds for the HC-formulae, we decided to study the influence of the walkProb-parameter on the performance of GSAT in more detail. Note, that basic GSAT (without random walk) is equivalent to the extended algorithm with walkProb=0, whereas walkProb=1 yields a variant which does *only* walk-steps.

For these experiments the test-sets for $HC_1(10)$ (one instance) and $HC_2(10)$ (100 instances) were used. For each instance maxTries=100 tries were being performed; the maxSteps-parameter was experimentally adjusted to 2000 to ensure maximal success rates between 50% and 100%. The diagrams in Fig.1 show the success rates sr resp. solution costs ss as a function of the walk-probability walkProb (denoted as wp).

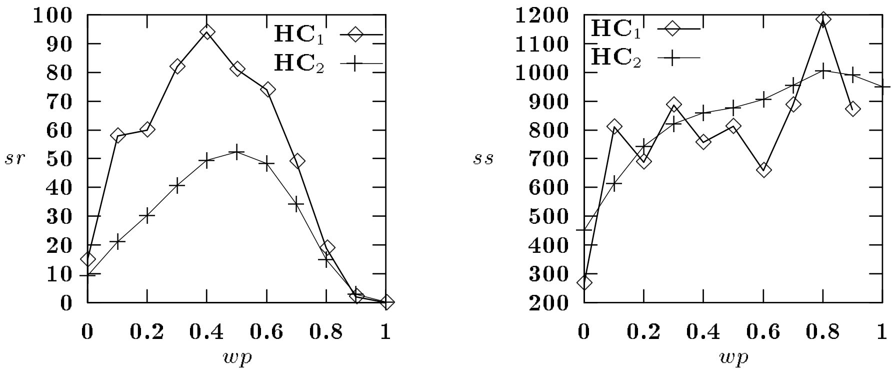

Fig. 1. Influence of random walk on GSAT's performance

Both sr-curves display a distinct, single maximum at walkProb=0.4 for HC_1 and walkProb=0.5 for HC_2.[5] Note further that for both problem classes, increasing walkProb beyond this optimal value causes rapid decreases in success rates. Other than basic GSAT (walkProb=0), the pure random walk strategy (walkProb=1) finds no solutions at all. The different locations of the maximum for HC_1 and HC_2 indicate that the problems with vertex out-degree 1 require a slightly more directed search (and therefore less walk-steps) than the much more difficult problems with out-degree 2. The reason for this may be that the objective function for HC_2-problems shows a greater number of local optima (at least for these problems there exist much more vertex permutations which use *almost* only edges of the given graph; the corresponding variable assignments therefore satisfy almost all clauses of the given formula).

[5] The optimal value of walkProb=0.5 is frequently used in other empirical studies and is often cited as a recommendation of Bart Selman.

There is one more interesting observation about these results: Although beyond the optimal walk-probability the number of solutions found by GSAT decreases steeply, the solution costs keep increasing for a while. As the problem instances usually have only one solution, this clearly indicates that for these walkProb-settings, walk-steps more and more impede the local search.

Summing up these observations, we get the following result: GSAT's performance on both problem classes HC_1 and HC_2 can be signficantly improved by using the random walk strategy. For fixed maxSteps-values, optimal settings of walkProb=0.4 for HC_1 and walkProb=0.5 for HC_2-instances could be indentified. As a consequence, these walkProb-settings were used for all other experimental series.

4.2 GSAT with Clause Weights

As mentioned before, the method of clause weighting has been found to improve GSAT's performance on several classes of structured problems [11, 3]. Therefore, one should expect that this extension of basic GSAT might also cause an increase in performance for the SAT-transformed HCP-instances which show a distinct structure as discussed in Sect.3. To check this hypothesis, we applied GSAT *with* and *without* clause weighting (denoted as GSAT+clw resp. GSAT–clw) to an identical test-set of 100 $HC_2(10)$-problems. All clause weights were initialized to 1 at the beginning of each experiment, and eventually increased by 1 for the clauses left unsatisfied at the end of unsuccessful tries (cf. Sect.2.2). For each problem instance, 100 tries were being performed with a walk-probability of 0.5, the value found to be optimal in another experimental series (see above). The performance values determined by these experiments are shown in Table 1 (where ms denotes the value of the maxSteps-parameter, sr the success rate and ss the average number of steps per solution).

Table 1. Influence of clause weighting on GSAT's performance on $HC_2(10)$-problems

	maxSteps	success rate sr	steps/solution ss
GSAT–clw	1000	32.31%	484.31
	2000	52.41%	876.11
GSAT+clw	1000	36.76%	479.33
	2000	57.29%	872.09

Whereas the solution costs ss are merely different for both variants, there is a slight increase in the success rate for GSAT+clw. Due to the great variance in the performance values for the single experiments this is not a statistically significant result in our opinion. Nevertheless, compared with the performance increases which have been reported to be achieved by using clause weights on other problem classes [11, 3], our results clearly indicate that for the given HC-formulae, clause weighting is not a very efficient method to improve GSAT's performance. We believe that this observation is independent of the problem size which is supported by analogous results on the test-set for $HC_2(8)$. Thus we conclude:

Result 3. *Using clause weighting has no significant influence on the performance of GSAT on Iwama-Miyazaki-transformed HC_2-instances.*

4.3 RBR-Extension of HC-Formulae

Another approach to cope with structured problems being very difficult for the GSAT algorithm (like the hierarchical structured cross-word-puzzle problems [9]), has been recently proposed by Kask and Dechter [8]. This method is called *restriced bound resolution (RBR)* and consists basically of adding resolvents of the original clauses to the given CNF-formula. To avoid creating too many new clauses, there is a restriction on the length of the added resolvents. RBR-resolution is usually realized as a prepocessing step for GSAT or other SAT-algorithms. The idea behind this procedure is to make the structure of the problem more explicit by adding resolvents of the original clauses.

Without going into the technical details of RBR-resolution here, we now report our results on HC-formulae. To see whether GSAT's performance can be improved by RBR-extending the problem instances, we first evaluated two different RBR-extensions of the formulae from our standard test-set for $HC_2(10)$ (100 instances) using bounds of 4 resp. 8 for the length of resolvents.[6] The resulting sets of extended instances shall be denoted as RBR_4 resp. RBR_8.

To these test-sets we applied GSAT with random walk (walkProb was set to the optimal value of 0.5). Despite the result of Sect.4.2, we also used clause weights, because a priori it is not clear, whether this method shows the same behaviour on RBR-extended formulae.[7] For every problem instance, 10 tries were performed. Table 2 shows the results obtained from experiments together with the results for GSAT on the original test-set (same parameter values).

Table 2. Influence of RBR-resolution on GSAT's performance on $HC_2(10)$-problems

	maxSteps	success rate sr	steps/solution ss
no RBR	1000	36.76%	479.33
	5000	84.04%	1742.35
RBR_4	1000	30.70%	491.04
	5000	81.00%	1787.14
RBR_8	1000	29.30%	456.55
	5000	81.60%	1670.26

[6] The choice of these bounds was motivated by the fact that the original clauses of $HC_2(10)$-problems are always of length 4 or 8.

[7] However, it is not to be expected that this way the results might become worse.

Considering the structure of the HC-formulae, one might have expected an improved performance of GSAT on the RBR-extended problems. But quite on the contrary, the results show slightly lower success rates and comparable solution costs for the extended test-sets. So we might conclude:

Result 4. *GSAT's performance on Iwama-Miyazaki-transformed HC_2-instances is not improved by application of RBR-extension; considering its time complexity, this kind of preconditioning does not seem to be profitable here.*[8]

5 Conclusion

Summing up the results presented in the previous sections, we have shown that HC-formulae (i.e. HCP-instances, transformed into SAT-problems by the Iwama-Miyazaki method) form a problem class which is very difficult for GSAT-like SAT-algorithms. Because of complete variable-dependency, a structural property which distinguishes these formulae from other classes of SAT-problems (as Random-k-SAT with constant clauses/variables ratio), the score-by-need strategy for efficient update of variable scores is rather useless in the case of HC-formulae.

Furtheron, our experimental results indicate that for solvable HCP-instances with vertex out-degree 2 transformed by the Iwama-Miyazaki method, standard mechanisms for improving GSAT's performance (RBR-extension and clause weighting) are apparently not effective. However, by using random walk (another standard extension of GSAT), a significant improvement can be achieved. Apparently, these formulae are structured in a way that makes them difficult not only for GSAT, but also for various extensions of the basic algorithm developped especially for improving its performance on structured problems. Thus, our results suggest that the class of HC_2-formulae can be used as a benchmark for evaluating the performance of SAT-algorithms based on local search.

The question whether systematic, complete SAT-algorithms (like efficient variants of the Davis Putnam procedure) might outperform GSAT-like local search methods on the problem class under consideration is still to be answered.[9] We also don't know yet, what specific structural aspects of HC-formulae make them so difficult for GSAT and cause both RBR-extension and clause weighting to fail. But we are convinced that further examination of these issues might lead to important insights concerning both SAT-transformations of problem instances from other domains and universal SAT-algorithms.

With respect to SAT-transformation, our results show clearly that for practical purposes other criteria than just the number of propositional variables have to be considered. As we have seen, for the application of GSAT to SAT-transformed problems, at least variable-dependency plays an important role. Thus we are convinced that the identification of these criteria and the development of more

[8] However, it cannot be ruled out that for different walk-probabilities (esp. walkProb=0), RBR-extension might be more effective.

[9] Actually, there is reason to believe, that the kind of forward checking realized by DP-algorithms may prove to be very effective for suitably coded HCP-instances.

efficient SAT-algorithms are closely related issues. The general approach of transforming specific instances of application problems into a logical domain (like SAT) and practically solving these transformed problems in an adequate way using general-purpose algorithms is of great importance to the field of AI. Therefore we strongly believe that further investigation of this approach, based on concrete applications, is a rewarding subject of AI-research.

References

1. A. Beringer, G. Aschemann, H.H. Hoos, M. Metzger, and A. Weiß. GSAT versus Simulated Annealing. In A. G. Cohn, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 130–134. John Wiley & Sons, 1994.
2. N. Boissin and J.-L. Lutton. A parallel simulated annealing algorithm. *Parallel Computing*, 19:859–872, 1993.
3. Byungki Cha and Kazuo Iwama. Performance test of local search algorithms using new types of random CNF formulas. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 304–310, 1995.
4. J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 21–27. MIT press, 1993.
5. Ian P. Gent and Toby Walsh. An empirical analysis of search in gsat. *(Electronic) Journal of Artificial Intelligence Research*, 1:47–59, 1993.
6. Ian P. Gent and Toby Walsh. Towards an understanding of hill–climbing procedures for SAT. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 28–33. MIT press, 1993.
7. Kazuo Iwama and Shuichi Miyazaki. SAT-variable complexity of hard combinatorial problems. In *Proceedings of the World Computer Congress of the IFIP*, pages 253–258 (Volume 1). Elsevier Science B.V., North Holland, 1994.
8. Kalev Kask and Rina Dechter. SAT and local consistency. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 612–622, 1995.
9. Kurt Konolige. Easy to be hard: difficult problems for greedy algorithms. In *Proceedings of the International Conference on Principles of Knowlege Representation and Reasoning*, pages 374–378, 1994.
10. Karl Rüdiger Reischuk. *Einführung in die Komplexitätstheorie*. Teubner Verlag, Stuttgart, 1990.
11. Bart Selman and Henry A. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In R. Bajcsy, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 1, pages 290–295. Morgan Kaufmann Publishers Inc., 1993.
12. Bart Selman and Henry A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 46–51. MIT press, 1993.
13. Bart Selman, Henry A. Kautz, and B. Cohen. Local search strategies for satisfiabiliy testing. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1993.
14. Bart Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 440–446. MIT press, 1992.
15. W.M. Spears. Simulated Annealing for hard satisfiability problems. Technical report, Naval Research Laboratory, Washington D.C., 1993.