GSAT and Local Consistency [*]

Kalev Kask
Computer Science Department
University of California at Irvine
Irvine, CA 92717
USA

Rina Dechter
Computer Science Department
University of California at Irvine
Irvine, CA 92717
USA

Abstract

It has been shown that hill-climbing constraint satisfaction methods like min-conflicts [Minton *et al.*, 1990] and GSAT [Selman *et al.*, 1992] can outperform complete systematic search methods like backtracking and backjumping on many large classes of problems. In this paper we investigate how preprocessing improves GSAT. In particular, we will focus on the effect of enforcing local consistency on the performance of GSAT. We will show that enforcing local consistency on uniform random problems has very little effect on the performance of GSAT. However, when the problem has hierarchical structure, local consistency can significantly improve GSAT. It has been shown [Konolige, 1994] that there are certain structured problems that are very hard for GSAT while being very easy for the Davis-Putnam procedure. We will show that they become very easy for GSAT once a certain level of local consistency is enforced.

1 Introduction

Local search algorithms like min-conflicts [Minton *et al.*, 1990] and GSAT [Selman *et al.*, 1992] have been successfully applied to different classes of constraint satisfaction problems like SAT, graph coloring, binary CSPs and scheduling. The popularity of local search algorithms can be attributed to their efficiency - they can outperform complete systematic search methods like backtracking and backjumping on large classes of problems. The question that arises is whether local search methods are always better, or at least not worse, than complete systematic search methods. This question is somewhat ambiguous since there is no one single version of GSAT - most of them employ clever heuristics that significantly improve their performance over the basic version of GSAT reported in [Selman *et al.*, 1992]. Moreover, several problems that once seemed very hard have been successfully solved once certain heuristics, like clause weighting and random walk, were added to GSAT.

In this paper we will investigate whether or not GSAT can be improved by applying preprocessing. Preprocessing algorithms are run on the problem in advance, before the search algorithm is tried, and change the problem representation into a different, but equivalent one. Preprocessing algorithms include a variety of consistency-enforcement algorithms. These algorithms make the problem more explicit by adding new constraints that are induced by the existing constraints. Enforcing local consistency can improve the performance of complete systematic search methods like backtracking and backjumping by eliminating dead ends, thus reducing the search space. We will apply the idea of enforcing local consistency to GSAT with the hope that its performance can also be improved by making the problem more explicit. In particular, we will focus on different forms of partial path consistency since full path consistency is not cost effective for large problem instances.

We will focus on two different classes of problems - random uniform problems that do not have any structure and random structured problems. As we will show, local consistency has a very different effect on the performance of GSAT on these two classes of problems. On uniform random problems, enforcing local consistency can help GSAT solve more problems but the overhead associated with enforcing local consistency and the added complexity of induced constraints eliminates any net gain. However, on a class of structured cluster problems, local consistency dramatically improved the performance of GSAT.

In a recent paper [Konolige, 1994] it was shown that there are certain classes of structured problems that are very hard for GSAT, even if the best currently known heuristics like clause weighting and random walk are used, while being very easy for the Davis-Putnam procedure. In this paper we will examine a similar class of structured 3SAT problems. These problems have a cluster structure - they contain clusters of variables each of which is a small group of variables tightly linked with constraints (3-SAT clauses). Clusters themselves are linked together by a different set of constraints (3-SAT clauses). It turns out that these kinds of hierarchical problems can be extremely hard for GSAT using the best currently known heuristics. But surprisingly, these

*This work was partially supported by NSF grant IRI-9157636, by the Electrical Power Research Institute (EPRI), and by grants from Toshiba of America, Xerox Northrop and Rockwell.

problems will become trivial for GSAT once a certain amount of local consistency is enforced, in this case a restricted form of Bound-3 resolution. The effect of enforcing this kind of local consistency is that it makes constraints more explicit by adding new induced constraints. This will change the GSAT search space by eliminating many near solutions [1] so that they will become assignments whose cost is high.

2 GSAT

Local search algorithms like GSAT work by first choosing an initial assignment and then incrementally improving it by flipping the value of a variable so that the new value leads to the largest increase in the number of satisfied constraints. This is done until all constraints are satisfied, or a predetermined number of flips (MAX_FLIPS) is reached, or GSAT reaches a local minimum.

The following is a standard GSAT procedure:

```
Procedure GSAT (CSP problem P, MAX_TRIES,
        MAX_FLIPS)
for i=1 to MAX_TRIES
    let A be a random initial assignment
    for j=1 to MAX_FLIPS
        if A satisfies P, return true
        else let F be the set of variable-value pairs that,
            when flipped to, give a maximum increase
            in the number of satisfied constraints;
            pick one f ∈ F and let new A be
            current A with f flipped
        end
    end
return false
end
```

This algorithm is almost never used in practice as it is here because its performance can be improved significantly by adding a number of heuristics. Our version of GSAT uses several heuristics that include, we believe, the best known heuristics today.

The basic GSAT is non-deterministic because it does not specify how to break ties between two or more variables having an equally good flip, or between two or more values that would give the same increase. [Gent and Walsh, 1993] suggest using historic information instead of breaking ties randomly. They propose that in the event of a tie, a variable that was flipped the longest ago be chosen.

We also use clause weighting as proposed by the Breakout method of P. Morris [Morris, 1993] and in a different form in [Selman and Kautz, 1993]. This method proposes a method of escaping local minimums by reweighting constraints. In addition, we use a version of random walk called random noise strategy in [Selman *et al.*, 1992]. This method suggests picking, with probability p, a variable that appears in an unsatisfied constraint and flipping its value. Unlike [Selman *et al.*, 1992], we flip not one, but three variables at a time and the probability p is not 50-60%, but 10-15%. This gives a little improvement over the original method because if only one variable is flipped, most of the time GSAT will flip it right back.

[1] Near solutions are assignments that have a cost of almost zero.

The third heuristic we use is similar to the one proposed in [Yugami *et al.*, 1994]. Their method proposes a way of escaping local minimums by using value propagation over unsatisfied constraints. We pick an unsatisfied constraint and check to see if it contains any variables whose value has not yet been flipped. If there is at least one, we will flip one of them so that the constraint becomes satisfied. There are two differences in what we do - [Yugami *et al.*, 1994] computes a closure under value propagation, whereas we do only a fixed number of steps. Second, in [Yugami *et al.*, 1994] this is done every time a local minimum is reached. We do it only at the end of every try as a way of generating a new initial assignment for the next try.

Last, there is always the problem of choosing MAX_TRIES and MAX_FLIPS. We solve this problem by using a heuristic that determines the length of every try (ie. MAX_FLIPS) automatically during every try [Hampson, 1993]. The idea is that we search as long as we are making progress, and if we haven't made progress for a while we give up and start a new try. Progress is measured as finding an assignment that satisfies more constraints than satisfied by any other assignment found by GSAT during that particular try. Every time we find such an assignment, we give GSAT time equal to the amount of time it has spent up until that point from the beginning of the try. If during this time no better assignment was found, we give up and start a new try. Using this strategy, we need to give only one parameter, MaxFlips, that bounds the total maximum number of flips that GSAT will spend on a problem.

3 Problem Format

The first class of 3SAT problems we experimented with is a set of cluster structures. These problems are characterized by the following parameters:

1. N - the number of variables per cluster.
2. C - the number of clauses per cluster.
3. cN - the number of clusters.
4. cC - the number of clauses between clusters.

Every cluster structure is generated by first generating cN clusters (each N variables and C clauses) and then generating cC clauses such that all 3 variables in a clause come from different clusters.

We also used binary constraint satisfaction problems such that all variables had the same domain of size K of natural numbers $\{1, \ldots, K\}$. All binary CSP problems are characterized by the following parameters:

1. N - the number of variables.
2. K - the number of values.
3. C - the number of constraints. For binary constraints $C_{max} = N \cdot (N-1)/2$.
4. T - the tightness of the constraint, as a fraction of the maximum K^2 pairs of values that are nogoods.

Every binary CSP problem is generated by first uniformly randomly choosing C pairs of variables and then

Figure 1: A Cluster Structure.

creating a constraint for every pair. To create a constraint, we randomly pick $T \times K^2$ tuples and mark them as pairs of values that are allowed.

When evaluating the performance of an algorithm, it is always important to know how many of the problems that were tried were actually solvable. This introduces an additional problem for GSAT since it is an incomplete algorithm. Normally in this situation, every problem is first solved using the Davis-Putnam procedure or any other complete algorithm. Unfortunately, this allows us to solve only small problems. Real life problems are likely to be large and intractable for any complete algorithm known today. We can get around this problem if we can generate problems for which we know in advance that they are solvable. The straightforward way of getting solvable 3SAT formulas is to first generate a solution and then generate random clauses and keep only those that are satisfied by the solution (in other words, at least one literal in the clause matches the solution). Unfortunately, these kind of problems are very easy. However, it turns out that if we throw away not only those clauses that have 0 literals satisfied by the solution, but also those that have exactly 2 literals satisfied by the solution we get random 3SAT problems that are on the average at least as hard, for GSAT, as the class of all solvable problems. All 3SAT cluster structures tested in this paper were generated this way.

Another property of solvable 3SAT formulas generated this way is that when the formulas are uniformly randomly generated, the hardest problems for GSAT are in the area where the ratio of the number of variables to the number of clauses is between 4.5 and 5. When this ratio is smaller or larger, problems are easy for GSAT. For example, when the ratio is 8, problems have only 1 solution [2] and on the average, finding this only solution is much easier for GSAT than solving a problem from the 4.5 - 5 range.

4 Random Cluster Structures

On the class of cluster problems described in the previous section GSAT performs very poorly. In Table 1. we have the results of experiments with cluster structures of $cN = 50$ clusters, $cC = 200$ clauses between clusters, $N = 5$ variables per cluster and $C = 30 - 40$ clauses per cluster. Each cluster by itself is very easy for GSAT because it is strongly over-constrained (the ratio of the number of clauses over the number of variables varies from 6 to 8) and as a result each cluster has only very few solutions. But taken together they appear extremely difficult for GSAT. For example, when the number of clauses per cluster grows from 30 to 37 the number of problems GSAT is able to solve drops from 100 to 41, and it takes on the average 252,000 flips to find a solution when it can find one (the upper bound $MaxFlips$ is 512K). When we increase the number of clauses per cluster to 40, GSAT fails to solve any problems (remember that all problems are solvable). For comparison, we have also included the running time of the Davis-Putnam procedure on the same problems.

This is surprising since 250 variable uniform random 3SAT formulas are fairly easy for this GSAT program. Our hypothesis is that this phenomena can be attributed to the structure of the problem. When the number of clauses per cluster increases from 30 to 40, the number of solutions each cluster has, when taken separately, decreases from a few to one. But the number of near solutions [3] remains large. When we start GSAT on an initial assignment, it always quickly converges to an assignment that is a near solution. The hardest part for any GSAT algorithm is to improve a near solution so that it becomes a real solution. On the cluster problems, GSAT quickly finds an assignment that for many clusters is a near solution. But it seems to be unable to improve this assignment. In order to improve it many changes in different clusters need to be made. But the structure of the problem – tight clusters with loose constraints between them – does not provide good guidance for GSAT.

However, we can enforce local consistency that will change the structure of the problem by adding new, induced constraints. We ran GSAT on the same problems after a preprocessing algorithm RBR-3 was run on them. RBR-3 computes a restricted form of Bound-3 resolution [Dechter and Rish, 1994] by resolving only pairs of original clauses and keeps only those resolvents that have no more that 3 literals. The results of these experiments are

[2]We know there is only one solution because when we compute the closure under Bound-3 resolution we almost always get a total of $C_{max} = 7 \cdot \binom{n}{3}$ clauses, which is the maximum number of clauses a solvable 3SAT formula can have. Also, when we have C_{max} clauses, the formula has only one solution. Finally, notice that new clauses added by Bound-3 resolution do not remove any solutions of the original formula.

[3]A near solution is an assignment that satisfies almost all clauses, and therefore, for which the value of the cost function is almost zero.

Solvable 3SAT cluster structures, 100 instances, MaxFlips = 512K 5 variables per cluster, 50 clusters, 200 clauses between clusters Restricted Bound-3 Resolution : only original clauses resolved Running times, number of flips and clauses added are given as an average per problem solved									
	Before Resolution			After Resolution					
C/cluster	Solved	Time	Flips	Solved	RBR-3 Time	Total Time	Flips	New Clauses	DP
30	100	0.52 sec	4.5K	100	3.6 sec	3.7 sec	189	1736	1.03 sec
31	100	0.71	5.1K	100	3.88	3.91	176	1731	1.04
32	100	1.03	8.4K	100	4.16	4.20	162	1722	1.09
33	100	1.54	12K	100	4.36	4.39	155	1708	1.11
34	100	3.44	26K	100	4.66	4.70	151	1690	1.15
35	100	6.38	49K	100	4.92	4.95	140	1668	1.18
36	90	21.7	161K	100	5.23	5.26	135	1640	1.19
37	41	35.5	252K	100	5.42	5.45	131	1609	1.23
38	3	28.4	202K	100	5.94	5.97	125	1574	1.27
39	0	-	-	100	5.95	5.98	121	1540	1.29
40	0	-	-	100	6.13	6.17	115	1503	1.29

Table 1: Bound-3 Resolution and GSAT

in Table 1. We see that after RBR-3 was run, problems became almost trivial. GSAT can solve all problems and on the average it needs only 115-190 flips. Almost all of the time was used by RBR-3.

When we look at where the new clauses are added we see that almost all of them are local clauses, namely all three literals are from the same cluster. In fact, the total number of clauses per cluster roughly doubles and is close to the maximum possible number of clauses $C_{max} = 7 \cdot \binom{n}{3}$ that a solvable 3SAT problem can have. This has the effect of eliminating many near solutions. Many assignments that previously satisfied all, except very few, clauses violate many of the new induced clauses.

5 Random Uniform Problems

What Bound-3 resolution did to cluster structures was that it added new induced constraints that in effect changed the search space by eliminating many near solutions. It would be natural to ask what would be the effect of local consistency enforcement on problems that do not have any special structure to begin with. Can they benefit the same way cluster structures did?

In this section we will focus on uniform random problems - the constraint graph of which does not have any special structure. We ran a series of experiments with both binary CSP problems and 3SAT formulas. Given a random problem, first we ran a local consistency enforcement algorithm and then ran GSAT. On 3SAT formulas, the local consistency algorithm computes a closure under Bound-3 resolution. On binary CSPs, the local consistency algorithm computes partial path consistency. We now take (in section 5.1) a small detour to discuss different versions of path-consistency algorithms.

5.1 Partial Path Consistency

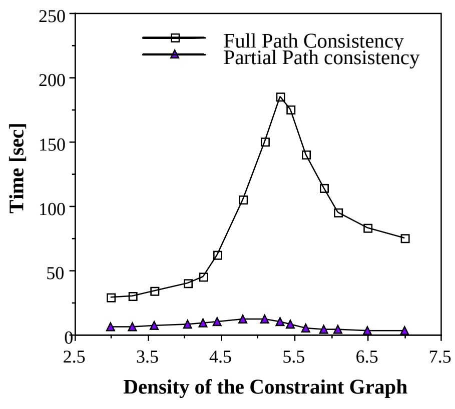
A problem is *path consistent* (or *3-consistent*) iff any instantiation of any two variables that is locally consistent can be extended to a consistent assignment of any third variable [Montanari, 1974]. Enforcing path consistency makes the problem more explicit: constraints that are induced by other constraints are added to the problem and thus become explicit.

It was shown that path consistency can potentially improve the performance of any backtracking search algorithm [Mackworth, 1977], since it frequently eliminates all dead ends from the search space [Dechter and Meiri, 1994]. It would be interesting to know if path consistency will also improve any local search algorithm like GSAT.

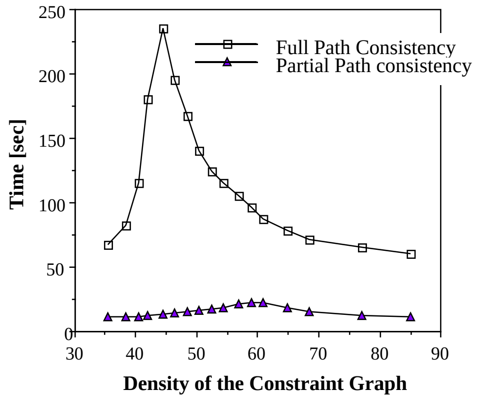
Unfortunately the complexity of enforcing path consistency is $\Theta(n^3k^3)$ which is too large for big problems. It was shown in [Dechter and Meiri, 1994] that for many problems the overhead of path consistency is not cost effective. Therefore instead of computing path consistency exactly we will approximate it by using a restricted form of path consistency called partial path consistency.

The idea is the following. We want partial path consistency (PPC) to be as close to path consistency (PC) as possible. This means that whenever PC removes a tuple from a constraint, we would like PPC to do the same. In the extreme case, all tuples will be removed from the constraint and it becomes empty, which means that the problem is inconsistent. In the following experiments we use that as a criteria to measure the quality of our PPC algorithm – whenever PC generates an empty constraint, we want PPC to also generate an empty constraint.

The amount of changes made by path consistency depends on the amount of local consistency present in the problem in the beginning [van Beek, 1994], [van Beek and Dechter, 1994]. Intuitively, the tighter the constraints and the denser the constraint graph, the more changes PC will make. It turns out that partial path consistency based on the following heuristic is almost as good as full path consistency – we choose a subset of the variables that have the highest degree and are tightly grouped together, and perform path consistency on the subproblem induced by these variables:

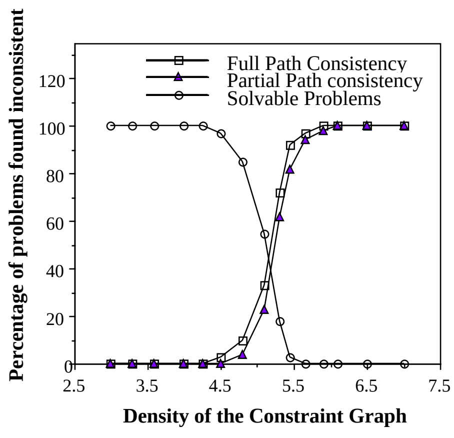


(a) 100 variables, 8 values, 32/64 tightness

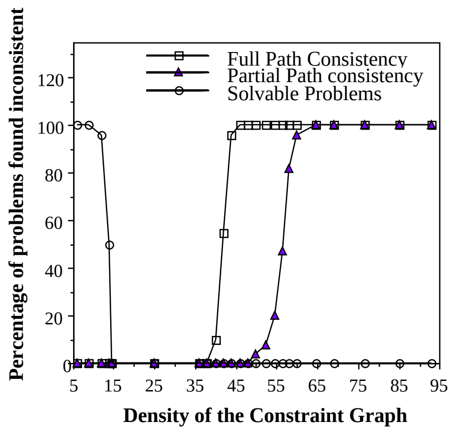


(b) 100 variables, 8 values, 16/64 tightness

Figure 2: Full and Partial Path Consistency.



(a) 100 variables, 8 values, 32/64 tightness



(b) 100 variables, 8 values, 16/64 tightness

Figure 3: Full and Partial Path Consistency.

```
Procedure PPC (CSP problem P, number of variables
        chosen m, locality parameter C)
for i=1 to n
    $a_i$ = degree of variable $X_i$
    end
let S be the set of variables and let S' = $\emptyset$
for i=1 to m
    let x be the variable in S with the largest $a_i$
    let S = S - x and let S' = S' $\cup$ x
    for all neighbors $X_j$ of x in S
        $a_j = a_j$ + C
        end
    end
do path consistency on S'
end
```

where C is a "locality" parameter that determines the order in which variables are chosen. If C = 0 then it would result in a MaxDegree ordering; if C = n, then it would result in a MaxCardinality ordering ([Tarjan and Yannakakis, 1984]). Our experiments show that we should set C equal to the average degree of the variable in the constraint graph. Parameter m is the number of variables chosen. Clearly, the larger m the closer partial path consistency is to full path consistency, but the longer it takes to compute it. We used $m = 40$ which resulted in a partial path consistency algorithm that was close enough to path consistency, but took significantly less time on large problems.

We ran a number of experiments comparing PPC to PC. In Figures 2 and 3 we have the results of running both full and partial path consistency on 300 randomly generated problems with 100 variables and 8 values; and the tightness of 32/64 and 16/64. On the left side we have the running time per problem and on the right side we have the percentage of problems that were solvable or inconsistent, as a function of the density of the constraint graph ($\frac{c}{c_{max}}100\%$).

For the 32/64 tightness problems the hardest area (50% solvability) is when problems have about 250 (5% of the maximum possible) constraints and for the 16/64 tightness problems, the hardest area is when problems have 655 (14% of the maximum possible) constraints. As

N=100, K=8, T=32/64, 200 instances, MaxFlips = 512K								
C	Solvable	Algorithm	Solved	Tries	Flips	PPC Time	Total Time	BJ-DVO
265	88.5 %	GSAT	139	336	147K	0 sec	36 sec	19 min
		PPC + GSAT	152	292	140K	8 sec	66 sec	
270	66 %	GSAT	78	406	191K	0 sec	45 sec	33 min
		PPC + GSAT	83	381	195K	14 sec	92 sec	
N=30, K=64, T=2048/4096, 100 instances, MaxFlips = 128K, C_{crit}=180?								
163		PPC + GSAT	56	276	59K	56 sec	153 sec	*
		GSAT	58	247	53K	0 sec	89 sec	*

Table 2: Partial Path Consistency and GSAT

Uniform random 3SAT, N=600, C=2550, 100 instances, MaxFlips = 512K					
Algorithm	Solved	Tries	Flips	BR-3 Time	Total Time
GSAT	36	63	176K	0 sec	15.3 sec
BR-3 + GSAT	31	45	125K	0.3 sec	15.0 sec

Table 3: Bound-3 Resolution and GSAT

we can see, the effectiveness of path consistency depends on the tightness of the constraints. For 32/64 problems, both full and partial path consistency discover almost all inconsistent problems. For 16/64 problems, full path consistency is not able to discover inconsistent problems in the 50% area and will become effective only when the density of the constraint graph grows. We also see that partial path consistency is only slightly less effective than full path consistency, especially for problems with tight constraints.

We also tested arc consistency and found that while it was very fast, it very seldom changed the problem, except for problems that were very overconstrained. This shows that although arc consistency is computationally attractive, having the worst case complexity of $O(n^2k^2)$, it is not nearly as powerful as path consistency, and will be useful only when constraints are very tight.

5.2 Local Consistency on Random Uniform Problems

In Table 2. we have the results of running GSAT on two sets of binary CSP problems, one with $N = 100$ variables, $K = 8$ values and tightness $T = 32/64$ (sparse constraint graphs) and the other with $N = 30$ variables, $K = 64$ values and $T = 2048/4096$ tightness[4] (dense constraint graphs). We ran two experiments on the same set of problems, first with just GSAT and then partial path consistency (PPC) followed by GSAT. For comparison we have included the average running time per problem of a backjumping algorithm with dynamic variable ordering of [Frost and Dechter, 1994].

As we can see, enforcing partial path consistency does help GSAT solve slightly more problems given the same upper bound $MaxFlips$. But if we include time in our consideration we see that this strategy is not very useful since the total time it takes to solve a problem gets much worse. There are two reasons for this. First, enforcing (partial) path consistency is computationally expensive, although the complexity of partial path consistency grows very slowly. Second, and more importantly, if the constraint graph was not complete, it will be complete after path consistency (or very close in case of partial path consistency). This will add additional complexity to the GSAT search algorithm since it has to consider additional constraints at every step. Notice that for cluster structures adding constraints was cost-effective while here it is not.

In Table 3. we have the results of experiments of running GSAT with and without a preprocessing algorithm BR-3 on uniform random 3SAT formulas with $N = 600$ variables and $C = 2550$ constraints (note that this class of 3SAT formulas contains both consistent and inconsistent problems). Unlike RBR-3 which resolves only original clauses, RB-3 computes a closure under Bound-3 resolution. In this case GSAT with BR-3 was even slightly worse than just GSAT in terms of the number of problems solved, although GSAT with BR-3 used fewer flips and was slightly faster. We also tried the Davis-Putnam procedure, but it took far too long.

6 Conclusions

In this paper we focused on the problem of how preprocessing improves the performance of GSAT. In particular, we have investigated the effect of enforcing a certain degree of local consistency, as a preprocessing step, on two different classes of problems – random uniform problems that do not have any structure and random structured problems. The effect of local consistency is sharply different on these two classes of problems.

Our experiments show that when problems do not have any special structure, local consistency does not

[4]This tightness was chosen because problems with this tightness are not path consistent [van Beek, 1994].

have a significant effect on the performance of GSAT. Even disregarding the cost of preprocessing, GSAT was frequently less effective on the preprocessed problem. However, on certain classes of structured problems, local consistency can significantly improve the performance of GSAT.

We have shown that there are structured problems that are extremely hard for GSAT while easy for the Davis-Putnam procedure. These problems are so hard that even heuristics like clause weighting which was originally designed to help escape local minimums caused by the special structure of the problem do not seem to help much. The characteristic feature of these problems is the presence of tightly connected clusters of variables which, in turn, are loosely connected by another set of global constraints.

This class of problems was first discovered by Konolige [Konolige, 1994]. In this paper we have shown how to deal with these kinds of problems. Our experiments show that enforcing local consistency, like bounded resolution, can make these problems almost trivial for GSAT. The overhead associated with enforcing this kind of local consistency is much less than the computation needed to solve the problem without it.

Acknowledgments

We would like to thank Irina Rish for running experiments with the Davis-Putnam procedure, and Dan Frost for letting us use his backjumping algorithm with dynamic variable ordering. We would also like to thank Steve Hampson for interesting and stimulating discussions on GSAT.

References

[Dechter and Meiri, 1994] R. Dechter and I. Meiri. Experimental Evaluation of Constraint Processing. *Artificial Intelligence*, 68, 211–241, 1994.

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional Resolution: The Davis-Putnam Procedure, Revisited. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 134–145, 1994.

[Frost and Dechter, 1994] D. Frost and R. Dechter. In Search of the Best Constraint Satisfaction Search. In *Proceedings of AAAI*, pages 301–306, 1994.

[Gent and Walsh, 1993] I. P. Gent and T. Walsh. Towards an Understanding of Hill-Climbing Procedures for SAT. In *Proceedings of AAAI*, pages 28–33, 1993.

[Hampson, 1993] S. Hampson. Changing Max-Flips Automatically. Personal Communication, 1993.

[Konolige, 1994] K. Konolige. Easy to be Hard: Difficult Problems for Greedy Algorithms. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 374–378, 1994.

[Mackworth, 1977] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1), 99–118, 1977.

[Minton *et al.*, 1990] S. Minton, M. D. Johnston, A. B. Philips, and P. Laired. Solving Large Scale Constraint Satisfaction and Scheduling Problems Using Heuristic Repair Methods. In *Proceedings of AAAI*, pages 17–24, 1990.

[Montanari, 1974] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7, 95–132, 1974.

[Morris, 1993] P. Morris. The Breakout Method for Escaping From Local Minima. In *Proceedings of AAAI*, pages 40–45, 1993.

[Selman *et al.*, 1992] B. Selman, H. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proceedings of AAAI*, pages 337–343, 1994.

[Selman *et al.*, 1992] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI*, pages 440–446, 1992.

[Selman and Kautz, 1993] B. Selman and H. Kautz. An Empirical Study of Greedy Local Search for Satisfiability Testing. In *Proceedings of AAAI*, pages 46–51, 1993.

[Tarjan and Yannakakis, 1984] E. Tarjan and M. Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs and Selectively Reduce Acyclic Hypergraphs. *SIAM J. Comput.*, 13(3), pages 566–579, August 1984.

[van Beek, 1994] P. van Beek. On the Inherent Level of Local Consistency in Constraint Networks. In *Proceedings of AAAI*, pages 368–373, 1994.

[van Beek and Dechter, 1994] P. can Beek and R. Dechter. Constraint Tightness versus Global Consistency. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 572–582, 1994.

[Yugami *et al.*, 1994] N. Yugami, Y. Ohta, and H. Hara. Improving Repair-Based Constraint Satisfaction Methods by Value Propagation. In *Proceedings of AAAI*, pages 344–349, 1994.