A Graph-Based Method for Improving GSAT

Kalev Kask and Rina Dechter
Department of Information and Computer Science
University of California, Irvine, CA 92717
{kkask, dechter}@ics.uci.edu

Abstract

GSAT is a randomized greedy local repair procedure that was introduced for solving propositional satisfiability and constraint satisfaction problems. We present an improvement to GSAT that is sensitive to the problem's structure. When the problem has a tree structure the algorithm is guaranteed to find a solution in linear time. For non-tree networks, the algorithm designates a subset of nodes, called cutset, and executes a regular GSAT algorithm on this set of variables. On all the rest of the variables it executes a specialized local search algorithm for trees. This algorithm finds an assignment that, like GSAT, locally minimizes the sum of unsatisfied constraints and also globally minimizes the number of conflicts in every tree-like subnetwork. We will present results of experiments showing that this new algorithm outperforms regular GSAT on sparse networks whose cycle-cutset size is bounded by 30% of the nodes.

Introduction

GSAT is a randomized greedy local repair procedure that was introduced for solving propositional satisfiability and constraint satisfaction problems (Minton et al. 1990), (Selman et al. 1992). GSAT for a CSP starts from a random assignment to each variable. It then 'flips' a value of some variable such that the new value leads to the largest increase in the number of satisfied constraints. This is repeated until all constraints are satisfied, until a predetermined number of flips is reached, or until GSAT reaches a local minimum. Recently, such local repair algorithms were successfully used on various large-scale hard problems such as 3-SAT, N-queen, scheduling and constraint satisfaction (Minton et al. 1990), (Selman et al. 1992).

It is known that optimization and constraint satisfaction can be accomplished in linear time and in a distributed fashion when the constraint graph of the problem is tree-structured (Bertelé & Brioschi 1972), (Dechter et al. 1990), (Collin et al. 1991). The guarantees provided by the tree algorithm are very attractive and we would like to combine it with GSAT and extend it to general constraint networks using the idea of *cycle-cutset decomposition* (Dechter 1990).

The work presented in this paper grows out of the work on improving connectionist energy minimization (Pinkas & Dechter 1992) where the idea of computing global minimization over subtrees induced by a subset of instantiated cutset variables was first introduced. Here we adapt this general method to the case of constraint satisfaction, in which the energy function is the sum of unsatisfied constraints, and the connectionist energy minimization algorithm is any local repair algorithm like GSAT. The result is a local search algorithm having two types of variables: a subset of variables that execute the traditional GSAT algorithm (the *cutset variables*), and the rest of the variables that execute the tree algorithm (the *tree variables*) in the context of the entire network. We subsequently provide the first empirical evidence that this method is successful in practice on classes of problems with relatively small cycle-cutset.

A *binary constraint satisfaction problem (CSP)* [1] (Dechter 1992), (Mackworth 1992) is the problem of finding an assignment to n variables, $X_1, ..., X_n$, where each variable has a domain of values D_i, such that all the given binary constraints R_{ij} are satisfied. R_{ij} is a constraint between variables X_i, X_j, and is given as the set of all pairs of values that are allowed. We also denote $R_{ij}(v,u) = 1$ if $(v,u) \in R_{ij}$, and '0' otherwise. A constraint satisfaction problem can be associated with a *constraint graph* where each variable is associated with a node and any two variables appearing in the same constraint are connected.

This paper is organized as follows. Section 2 presents a local search algorithm for trees that is guaranteed to

[1] For simplicity of exposition we restrict ourselves to binary constraint problems. However everything is applicable to the general CSP.

converge to a solution in linear time. Section 3 extends the approach, resulting in a local search algorithm for arbitrary networks. This algorithm is a combination of GSAT and Tree Algorithm. Sections 4 and 5 present some experimental results using our new algorithm.

Tree Algorithm for Networks with Cycles

It is well-known that tree-like constraint problems can be solved in linear time (Mackworth & Freuder 1985). We use the idea of cycle-cutset (Dechter 1990) to generalize it for networks with cycles, an idea used both in Bayes networks and constraint networks. The cycle-cutset decomposition is based on the fact that an instantiated variable cuts the flow of information on any path on which it lies and therefore changes the effective connectivity of the network. Consequently, when the group of instantiated variables cuts all cycles in the graph, (e.g., a cycle-cutset), the remaining network can be viewed as cycle-free and can be solved by a tree algorithm. The complexity of the cycle-cutset method when incorporated within backtracking schemes can be bounded exponentially in the size of the cutset in each nonseparable component of the graph (Dechter 1990).

In this section we will present a *Tree Algorithm* that generalizes the original tree algorithm (Mackworth & Freuder 1985) to cyclic networks. When the problem has a tree structure, this algorithm finds a solution like the original tree algorithm. However, when the constraint graph contains cycles, it finds an assignment that minimizes the sum of unsatisfied constraints over all its tree subnetworks (Pinkas & Dechter 1995).

Assume that we have a constraint problem P that is arc consistent and has all of its variables V divided into two disjoint sets of variables, *cycle cutset* variables Y and *tree variables* X such that the subgraph of P induced by X is a forest.

Given a fixed assignment of values to cycle cutset variables $Y = y$ we define a cost $C_{X_i}(v)$ for every tree variable $X_i \in X$ and for each of its values $v \in D_{X_i}$ with respect to the parent $p(X_i)$ of X_i as follows. Let $T_{X_i}^{p(X_i)}$ be the subproblem of P induced by Y and the subtree of P rooted at X_i (including X_i). The cost $C_{X_i}(v)$ is the minimum sum of unsatisfied constraints in $T_{X_i}^{p(X_i)}$ when $X_i = v$ and conditioned on assignment $Y = y$.

Given this we can solve the constraint problem P in two steps. First we can compute the minimum cost of all trees conditioned on a fixed assignment $Y = y$ and then minimize the resulting function over all possible assignments to Y. Let $C(X|Y = y)$ be the sum of unsatisfied constraints in the entire network conditioned on assignment $Y = y$, and C_{min} the minimum overall sum of unsatisfied constraints. Clearly,

$$C_{min} = \min_{Y=y} C(y) = \min_{Y=y} \min_{X=x} \{C(X \mid Y = y)\}$$

The conditional minima of $C(X \mid Y = y)$ can be computed efficiently using the *Tree Algorithm* in Figure 1. The overall minima could be achieved by enumerating over all possible assignments to Y.

The Tree Algorithm works by first computing the cost $C_{X_i}(k)$ for every tree variable X_i assuming a fixed cutset assignment $Y = y$. It then computes new values for all tree variables X_i using the cost values computed earlier and the new value of the parent of X_i.

GSAT with Cycle Cutset

The Tree Algorithm described in the previous section leads to a combined algorithm for solving constraint problems. The Tree Algorithm is a basic operation that minimizes the cost of tree subnetworks given a cycle cutset assignment. However, the combined algorithm depends on a second search algorithm to find cycle cutset assignments. We could use a complete backtracking type algorithm for enumerating all cycle cutset assignments. The complexity of this algorithm would be exponential in the size of the cutset, and it would be practical only if the cutset size is very small (Dechter 1990).

We could also use an incomplete search algorithm to generate cycle cutset assignments. In this paper we have chosen to use GSAT for that purpose, because GSAT has proven to be superior to complete systematic search methods on large classes of constraint satisfaction problems.

We will next show how to implement the combined algorithm of GSAT and Tree Algorithm, called GSAT + CC (GSAT with Cycle Cutset). Given a constraint problem we assume that the set of variables is already divided into a set of cycle cutset variables $Y = \{Y_1, ..., Y_k\}$ and a set tree variables $X = \{X_1, ..., X_n\}$. The Tree Algorithm will be used as a basic subroutine in GSAT + CC. GSAT will always keep track of the current cycle cutset assignment and occasionally will ask the Tree Algorithm to compute an assignment for the tree variables that would minimize the cost of tree subnetworks.

However, the role of Tree Algorithm in this combined algorithm is more than just a simple subroutine. By computing a minimal assignment for the tree variables, Tree Algorithm is providing information for GSAT that guides the GSAT search as well.

This exchange of information between GSAT and Tree Algorithm is implemented by a feature of the Tree

Tree Algorithm : minimizing the cost of a tree-like subnetwork:

Input: An arc consistent CSP problem with its variables V divided into cycle cutset variables Y and tree variables X, $V = X \cup Y$ and $X \cap Y = \emptyset$. The set of cycle cutset variables has a fixed assignment $Y = y$.

Output: An assignment $X = x$ that minimizes the cost $C(X \mid Y = y)$ of the entire network conditioned on assignment $Y = y$.

1. For every value of every variable compute its cost :

   (a) For any value k of any cycle cutset variable Y_i, the cost $C_{Y_i}(k)$ is 0.

   (b) For any value k of a tree variable X_i, the cost $C_{X_i}(k)$ is

$$C_{X_i}(k) = \sum_{child\ X_j\ of\ X_i} min_l\ (C_{X_j}(l)\ +\ W_{X_i,X_j}(k,l))$$

   where $W_{X_i,X_j}(k,l)$ is the weight of the constraint between variables X_i and X_j and is either 0 if $(k,l) \in R_{X_i,X_j}$ or some positive number otherwise.

2. Compute new values for every tree variable :

   For a tree variable X_i, let d_{X_i} be the set of values of X_i that is consistent with the newly assigned value $v(p(X_i))$ of the parent $p(X_i)$ of X_i. Then the new value of X_i is

$$v(X_i) = arg\ min_{k\ \in\ d_{X_i}}(C_{X_i}(k) + W_{X_i,p(X_i)}(k,\ v(p(X_i))))$$

Figure 1: Tree Algorithm

Algorithm : one TRY of GSAT with Cycle Cutset :

Input: A CSP problem with its variables V divided into cycle cutset variables C and tree variables T, $V = C \cup T$ and $C \cap T = \emptyset$.

Output: An assignment $X = x, Y = y$ that is a local minimum of the overall cost function $C\ (X, Y)$.

1. Create a random initial assignment for all variables.

2. Alternatively execute these two steps until either the problem is solved, the Tree Algorithm does not change the values of the variables, or no progress is being made.

   (a) Run Tree Algorithm on the problem, where the values of cycle cutset variables are fixed.

   (b) Run GSAT on the problem, where the values of tree variables are fixed.

   To measure progress, we are using a system of credits. Every time GSAT flips a variable, it spends one credit. Whenever GSAT finds a new assignment that satisfies more constraints than any other assignment it has found during this try, we give GSAT new credit that is equal to the number of flips it has performed during this try so far. Whenever GSAT runs out of credit, it stops.

Figure 2: GSAT with Cycle Cutset

Algorithm in Figure 1. When a new value is computed for a tree variable X_i, the new value of the parent of X_i (that is computed before X_i gets a new value) is used to filter the domain of X_i. Only those values in the domain of X_i are candidates that are consistent with the new value of the parent of X_i. This restriction comes from the observation that if the Tree Algorithm leaves a constraint inside the tree subnetwork (ie. a constraint that only contains tree variables) unsatisfied, then this will not provide any useful information for GSAT since GSAT will not see it. The only tree variables that have effect on GSAT are those that are adjacent to cycle cutset variables. Therefore, whenever a constraint involving a tree variable has to remain unsatisfied, the Tree Algorithm has to make sure that it is adjacent to a cycle cutset variable [2].

The combined algorithm GSAT + CC will execute a number of tries, each of which starts from a new random initial assignment. The number of tries GSAT + CC executes is a function of some input parameter. Within one try, GSAT + CC will alternatively use both GSAT and Tree Algorithm. First it will run Tree Algorithm on the problem, such that the values of cycle cutset variables are fixed. Once the Tree Algorithm has finished, it will run GSAT on the problem such that the values of tree variables are fixed. Once GSAT has finished, for whatever reason, it will switch back to the Tree Algorithm. One try of the combined algorithm GSAT + CC is described in Figure 2.

Theorem 1 *The Tree Algorithm in Figure 1 is guaranteed to find an assignment that minimizes the sum of unsatisfied constraints in every tree-like subnetwork, conditioned on the cutset values.*

If the weights of all constraints are fixed at 1, GSAT + CC is guaranteed to converge to an assignment that is a local minimum of the number of unsatisfied constraints. In general, it will converge to an assignment that is a local minimum of the weighted sum of unsatisfied constraints [3]*.* □*.*

Experimental Methodology

In order to test our new algorithm, we have generated several sets of random binary CSPs. For every set of test problems, we have run both regular GSAT alone and GSAT with Cycle Cutset (GSAT + CC) on exactly the same problems.

In our experiments all variables had the same domain size K of natural numbers $\{1, \ldots, K\}$. All binary CSP problems we generated are characterized by the following parameters:

1. N - number of variables.
2. K - number of values.
3. C - number of constraints.
4. T - 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 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 not allowed.

Both GSAT and GSAT + CC are incomplete randomized algorithms. In order to compare their performance on a given problem, we will run both algorithms on the problem such that both have the same amount of CPU time. For final comparison, given a set of test problems, we have measured the number of problems they are able to solve provided that both algorithms were given the same amount of CPU time per problem.

In our experiments we have used a version of GSAT (Kask & Dechter 1995) that we believe includes most of the currently known best heuristics. We use a heuristic proposed by Gent and Welsh (Gent & Walsh 1993) for breaking ties. We also use constraint weighting as proposed by the Breakout method of P. Morris (Morris 1993) and by Selman and Kautz in a different form (Selman & Kautz 1993). This method proposes a way of escaping local minimums by reweighting constraints. We also use a version of random walk called a random noise strategy (Selman et al. 1992). This method suggest picking, with some small probability, a variable that appears in an unsatisfied constraint and flipping its value so that the constraint becomes satisfied. Finally, we also use a system of credits for automatically determining the length of every try of GSAT.

The amount of time GSAT (GSAT + CC) spends on a problem is determined by a parameter $MaxFlips$. In general, the length of a each try is different from the length of the previous try, because it is determined automatically by GSAT at runtime. Whenever GSAT finishes a try, the number of flips it performed during this try is subtracted from the given $MaxFlips$. If the remaining $MaxFlips$ is greater than 0, GSAT will start another try. Note that GSAT and GSAT + CC will use a different input parameter $MaxFlips$ in order to use the same amount of CPU time.

It is worth noting that the version of GSAT used in GSAT + CC did not use constraint reweighting. We observed that since we already use the Tree Algorithm the added benefit of constraint reweighting is insignificant (except when the graph is a complete graph) and can sometimes actually hurt the performance of the

[2] If we don't enforce this condition, the performance of GSAT + CC would deteriorate by several orders of magnitude.

[3] In some versions of GSAT weights of constraints change dynamically during the search.

Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 44/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
125	11 %	29 sec	46	10 sec	90	2 sec
130	12 %	46 sec	29	16 sec	77	6 sec
135	14 %	65 sec	13	23 sec	52	10 sec
Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 40/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
160	20 %	52 sec	33	20 sec	90	7 sec
165	21 %	60 sec	13	30 sec	80	17 sec
170	22 %	70 sec	4	40 sec	54	22 sec
Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 32/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
235	34 %	52 sec	69	14 sec	66	18 sec
240	35 %	76 sec	57	22 sec	57	29 sec
245	36 %	113 sec	40	43 sec	40	43 sec
Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 28/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
290	41 %	55 sec	74	13 sec	30	25 sec
294	42 %	85 sec	80	25 sec	23	41 sec
300	43 %	162 sec	63	45 sec	19	82 sec

Table 1: GSAT vs. GSAT + CC

combined algorithm.

Once we have generated a random problem, we have to find a cycle cutset required by GSAT + CC as input. We have used a very simple greedy heuristic to approximate this NP-complete problem. For every variable we compute the number of cycles it participates in using depth first search. We pick the variable that participates in the most cycles, place it in the cycle cutset, remove it from the graph and repeat the whole process. However, the maximum number of cycles a variable can belong to is exponential. Therefore we have placed a limit (in our experiments it was a couple of hundred) on the number of cycles a node has to be in before it was placed in the cycle cutset and removed. This greedy heuristic was very fast and usually took no more than a couple of seconds per problem [4].

Experiments

We have run three sets of experiments. All problems in the first set of experiments had 100 variables, each having 8 values. We varied the tightness of the constraints and the number of constraints. In Table 1 we have the results of these experiments. We have organized the results by increasing cycle cutset size. We can see that when the cycle cutset size is less than 30% of the variables, GSAT + CC can solve up 3-4 times more problems, given the same amount of CPU time, than GSAT alone. When the cycle cutset size is about 30% both algorithms are roughly equal. When the cutset size grows further, GSAT alone is better than GSAT + CC. The same is true when we look at the time each algorithm takes to solve a problem (provided it is able to solve the problem). When the cutset size is less than 30%, GSAT + CC can solve problems faster than GSAT alone. When the cutset size is more than 30%, GSAT alone is faster. Note that each line in the table is the sum of results for 100 random instances.

The time bound is set so that the $MaxFlips$ parameter for GSAT alone ranges from 300,000 to 1,000,000, and for GSAT + CC from 10,000 to 60,000. We have tried to set the tightness of constraints and the number of constraints so that we get problems that are close to the 50% solvability region.

[4] In our experimental results we have not counted this time as part of the GSAT + CC running time.

Binary CSP, 100 instances, 125 variables, 6 values, tightness 21/36				
# of constraints	avg. cutset size	Time Bound	GSAT solved	GSAT+CC solved
170	14 %	36 sec	81	100
180	16 %	54 sec	62	93
190	17 %	66 sec	30	72
Binary CSP, 100 instances, 125 variables, 6 values, tightness 18/36				
# of constraints	avg. cutset size	Time Bound	GSAT solved	GSAT+CC solved
245	26 %	66 sec	66	90
250	27 %	77 sec	42	79
255	28 %	90 sec	25	66
Binary CSP, 100 instances, 125 variables, 6 values, tightness 16/36				
# of constraints	avg. cutset size	Time Bound	GSAT solved	GSAT+CC solved
290	33 %	41 sec	84	76
300	34 %	62 sec	64	47
305	35 %	86 sec	41	34

Table 2: GSAT vs. GSAT + CC

In Tables 2 and 3 we have the results of experiments with two sets of random problems (with 125 variables, each having 6 values; and 150 variables, each having 4 values). The results are very similar to those with the first set of problems - when the cutset size is less than 30% GSAT + CC is better than GSAT. But when the cutset size is larger than 30%, GSAT alone is better.

Conclusion

We have presented a new combined algorithm that is parameterized by a subset of cutset variables, given as input. In this algorithm cutset variables execute a regular search algorithm (like GSAT) while the rest of the variables execute the Tree Algorithm, a local search algorithm that finds solutions on tree subnetworks.

On acyclic networks this algorithm is guaranteed to find a solution in linear time. When all the nodes are designated as cutset nodes this algorithm reduces to the main search algorithm (like GSAT).

Our experiments with this new algorithm have provided a simple criteria for deciding when to use this new algorithm - whenever we can find a small cutset (no more than 30% of the variables in the cutset), GSAT + CC is superior to one of the best alternative algorithms, GSAT.

References

Bertelé, U., and Brioschi, F. 1972. *Nonserial Dynamic Programming.* Academic Press.

Collin, Z.; Dechter, R; and Katz, S. 1991. On the Feasibility of Distributed Constraint Satisfaction. In *Proceedings of IJCAI.* Sydney, Australia.

Dechter, R. 1990. Enhancement Schemes for Constraint Processing: Backjumping, Learning and Cutset Decomposition. *Artificial Intelligence* 41(3): 273–312.

Dechter, R. 1992. Constraint networks. *Encyclopedia of Artificial Intelligence* 2nd ed. John Wiley & Sons, Inc., 276–285.

Dechter, R.; Dechter, A.; and Pearl, J. 1990. Optimization in Constraint Networks. In Oliver, R.M., and Smith, J.Q., *Influence Diagrams, Belief Nets and Decision Analysis.* John Wiley & Sons, Inc.

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

Kask, K., and Dechter, R. 1995. GSAT and Local Consistency. In *Proceedings of IJCAI.* Montreal, Canada.

Mackworth, A. 1992. Constraint Satisfaction. *Encyclopedia of Artificial Intelligence* 2nd ed. John Wiley & Sons, Inc., 285–293.

Mackworth, A.K., and Freuder, E.C. 1985. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25: 65-74.

Minton, S.; Johnson, M.D.; and Phillips, A.B. 1990. Solving Large Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.

Binary CSP, 100 instances, 150 variables, 4 values, tightness 8/16				
# of constraints	avg. cutset size	Time Bound	GSAT solved	GSAT+CC solved
215	14 %	50 sec	56	98
220	15 %	56 sec	45	94
225	16 %	63 sec	21	84
Binary CSP, 100 instances, 150 variables, 4 values, tightness 7/16				
# of constraints	avg. cutset size	Time Bound	GSAT solved	GSAT+CC solved
270	23 %	46 sec	59	82
275	24 %	59 sec	44	70
280	25 %	64 sec	32	62
Binary CSP, 100 instances, 150 variables, 4 values, tightness 6/16				
# of constraints	avg. cutset size	Time Bound	GSAT solved	GSAT+CC solved
335	31 %	56 sec	72	46
340	32 %	65 sec	65	41
345	33 %	74 sec	53	29

Table 3: GSAT vs. GSAT + CC

In *Proceedings of the Eighth Conference on Artificial Intelligence*, 17–24.

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

Pinkas, G., and Dechter, R. 1995. On Improving Connectionist Energy Minimization. *Journal of Artificial Intelligence Research (JAIR)*, 3: 223–248. A shorter version appeared in AAAI-92.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440–446.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise Strategies for Improving Local Search. In *Proceedings of AAAI*, 337–343.

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