This paper appeared at the
1st International Joint Workshop on Artificial Intelligence and Operations Research,
Timberline, Oregon, 1995.

Solving Problems with Hard and Soft Constraints Using a Stochastic Algorithm for MAX-SAT

Yuejun Jiang, Henry Kautz, and Bart Selman
AT&T Bell Laboratories
Direct correspondence to:
Henry Kautz
600 Mountain Ave., Room 2C-407
Murray Hill, NJ 07974
{kautz}@research.att.com

Abstract

Stochastic local search is an effective technique for solving certain classes of large, hard propositional satisfiability problems, including propositional encodings of problems such as circuit synthesis and graph coloring (Selman, Levesque, and Mitchell 1992; Selman, Kautz, and Cohen 1994). Many problems of interest to AI and operations research cannot be conveniently encoded as simple satisfiability, because they involve both hard and soft constraints – that is, any solution may have to violate some of the less important constraints. We show how both kinds of constraints can be handled by encoding problems as instances of weighted MAX-SAT (finding a model that maximizes the sum of the weights of the satisfied clauses that make up a problem instance). We generalize our local-search algorithm for satisfiability (GSAT) to handle weighted MAX-SAT, and present experimental results on encodings of the Steiner tree problem, which is a well-studied hard combinatorial search problem. On many problems this approach turns out to be competitive with the best current specialized Steiner tree algorithms developed in operations research. Our positive results demonstrate that it is practical to use domain-independent logical representations with a general search procedure to solve interesting classes of hard combinatorial search problems.

1 Introduction

Traditional satisfiability-testing algorithms are based on backtracking search

(Davis and Putnam 1960). Surprisingly few search heuristics have proven to be generally useful; increases in the size of problems that can be practically solved have come mainly from increases in machine speed and more efficient implementations (Trick and Johnson 1993). Selman, Levesque, and Mitchell (1992) introduced an alternative approach for satisfiability testing, based on stochastic local search. This algorithm, called GSAT, is only a partial decision procedure – it cannot be used to prove that a formula is unsatisfiable, but only find models of satisfiable ones – and does not work on problems where the structure of the local search space yields no information about the location of global optima (Ginsberg and McAllester 1994). However, GSAT is very useful in practice. For example, it is the only approach that can solve certain very large, computationally hard, formulas derived from circuit synthesis problems (Selman, Kautz, and Cohen 1994). It can also solve randomly generated Boolean formulas that are two orders of magnitude larger than the largest handled by any current backtracking algorithm (Selman and Kautz 1993a).

The success of stochastic local search in handling formulas that contain thousands of discrete variables has made it a viable approach for directly solving logical encodings of interesting problems in AI and operations research (OR), such as circuit diagnosis and planning (Selman and Kautz 1993b). Thus, at least on certain classes of problems, it provides a general model-finding technique that scales to realistically-sized instances, demonstrating that the use of a purely declarative, logical representation is not necessarily in conflict with the need for computational efficiency. One issue that arises in studying this approach to problem-solving is developing problem encodings where a solution corresponds to a satisfying model (Kautz and Selman 1992), instead of having a solution correspond to a refutation proof (Green 1969). But for some kinds of problems no useful encoding in terms of propositional satisfiability can be found – in particular, problems that contain both hard and soft constraints.

Each clause in a CNF (conjunctive normal form) formula can be viewed as a constraint on the values (true or false) assigned to each variable. For satisfiability, all clauses are equally important, and all clauses must evaluate to "true" in a satisfying model. Many problems, however, contain two classes of constraints: hard constraints that must be satisfied by any solution, and soft constraints, of different relative importance, that may or may not be satisfied. In the language of operations research, the hard constraints specify the set of feasible solutions, and the soft constraints specify a function to be optimized in choosing between the feasible solutions. When both kinds of constraints are represented by clauses, the formula constructed by conjoining all the clauses is likely to be unsatisfiable. In order to find a solution to the original problem using an ordinary satisfiability procedure, it is necessary to repeatedly try to exclude different subsets of the soft constraints from the problem representation, until a satisfiable formula is found. Performing such a search through the space of soft constraints, taking into account their relative importance, can be complex and costly in a practical sense, even when the theoretical complexity of the entire process is the same as ordinary satisfiability.

A more natural representation for many problems involving hard and soft constraints is *weighted maximum satisfiability* (MAX-SAT). An instance of weighted MAX-SAT consists of a set of propositional clauses, each associated with a positive integer weight. If a clause is not satisfied in a truth assignment, then it adds the cost of the weight associated with the clause to the total cost associated with the truth assignment. A solution is a truth assignment that maximizes the sum of the weights of the satisfied clauses (or, equivalently, that minimizes the sum of the weights of the unsatisfied clauses). Note that if the sum of the weights of all clauses that correspond to the soft constraints in the encoding of some problem is l, and each hard constraint is represented by a clause of weight greater than l, then assignments that violate clauses of total weight l or less exactly correspond to feasible solutions to the original problem. The basic GSAT algorithm can be generalized, as we will show, to handle weighted MAX-SAT in an efficient manner. An important difference between simple SAT and weighted MAX-SAT problems is that for the latter, but not the former, near (approximate) solutions are generally of value.

The main experimental work described in this paper is on Boolean encodings of *network Steiner tree problems*. These problems have many applications in network design and routing, and have been intensively studied in operations research for several decades (Hwang *et al.* 1992). We worked on a well-known set of benchmark problems, and compared our performance with the best published results. One of our implicit goals in this work is to develop representations and algorithms that provide state-of-the-art performance, and advance research in both the AI and operations research communities (Ginsberg 1994).

Not all possible MAX-SAT encodings of an optimization problem are equally good. For practical applications, the final size of the encoding is crucial, and even a low-order polynomial blowup in size may be unacceptable. The number of clauses in a straightforward propositional encoding of a Steiner tree problem is quadratic in the (possibly very large) number of edges in the given graph. We therefore developed an alternative encoding, that is instead linear in the number of edges. This savings is not completely free, because the alternative representation only approximates the original problem instance – that is, theoretically it might not lead to an optimal solution. Nonetheless, the experimental results we have obtained using this encoding and our stochastic local search algorithm are competitive in terms of both solution quality and speed with the best specialized Steiner tree algorithms from the operations research literature.

The general approach used in our alternative representation of Steiner problems is to break the problem down into small, tractable subproblems, pre-compute a set of near-optimal solutions to each subproblem, and then use MAX-SAT to assemble a global solution by picking elements from the pre-computed sets. This general technique is applicable to other kinds of problems in AI and operations research.

In a sense this paper describes a line of research that has come full circle: much of the initial motivation for our earlier work on local search for satisfiability testing came from work by Adorf and Johnston (1990) and Minton *et al.* (1990) on using local search for scheduling problems that did involve both hard and soft constraints. Thus, we turned a method for optimization problems into one for decision problems, and now are returning to optimization problems. However, instead of creating different local search algorithms for each problem domain, we translate instances from different domains into weighted CNF, and use one general, highly optimized search algorithm. Thus we retain the use of purely propositional problem representations, and our finely-tuned randomized techniques for escaping from local minima during search.

2 A Stochastic Search Algorithm

The GSAT procedure mentioned in the introduction solves satisfiability problems by searching through the space of truth assignments for one that satisfies all clauses (Selman, Levesque, and Mitchell 1992). The search begins at a random complete truth assignment. The neighborhood of a point in the search space is defined as the set of assignments that differ from that point by the value assigned to a single variable. Each step in the search thus corresponds to “flipping” the truth-value assigned to a variable. The basic search heuristic is to move in the direction that maximizes the number of satisfied clauses. Similar local-search methods to satisfiability testing has also been investigated by Hanson and Jaumard (1990) and Gu (1992).

Thus GSAT can already be viewed as a special kind of MAX-SAT procedure, where all clauses are treated uniformly, and which is run until a completely satisfying model is found. We have experimented with many modifications to the search heuristic, and currently obtain the best performance with the following specific strategy for picking a variable to change. First, a clause in the problem instance that is unsatisfied by the current assignment is chosen at random – the variable to be flipped will come from this clause. Next, a coin is flipped. If it comes up heads (with a probability that is one of the parameters to the procedure), then a variable that appears in the clause is chosen at random. This kind of choice is called a “random walk”. If the coin comes up tails instead, then the algorithm chooses a variable from the clause that, when flipped, will cause as few clauses as possible that are currently satisfied to become unsatisfied. This kind of choice is called a “greedy” move. Note that flipping a variable chosen in this manner will always make the chosen clause satisfied, and will tend to increase the overall number of satisfied clauses – but sometimes will in fact decrease the number of satisfied clauses. This refinement of GSAT was called “WSAT” (for “walksat”) in Selman, Kautz, and Cohen (1994).

The *weighted* MAX-SAT version of Walksat, shown in Fig. 1, uses the sum of

```
**procedure** Walksat(WEIGHTED-CLAUSES, HARD-LIMIT, MAX-FLIPS,
        TARGET, MAX-TRIES, NOISE)
M := a random truth assignment over the variables that
    appear in WEIGHTED-CLAUSES;
HARD-UNSAT := clauses not satisfied by M with weight $\geq$ HARD-LIMIT;
SOFT-UNSAT := clauses not satisfied by M with weight $<$ HARD-LIMIT;
BAD := sum of the weight of HARD-SAT and SOFT-UNSAT;
TOPLOOP: for I := 1 to MAX-TRIES do
    for J := 1 to MAX-FLIPS do
        if BAD $\leq$ TARGET then break from TOPLOOP; endif
        if HARD-UNSAT is not empty then
            C := a random member of HARD-UNSAT;
        else C := a random member of SOFT-UNSAT; endif
        Flip a coin that has probability NOISE of heads;
        if heads then
            P := a randomly chosen variable that appears in C;
        else
            for each proposition Q that appears in C do
                BREAKCOUNT[Q] := 0;
                for each clause C' that contains Q do
                    if C' is satisfied by M, but not
                        satisfied if Q is flipped then
                        BREAKCOUNT[Q] $+=$ weight of C'
                    endif
                endfor
            endfor
            P := a randomly chosen variable Q that appears in C and whose
                BREAKCOUNT[Q] value is minimal;
        endif
        Flip the value assigned to P by M;
        Update HARD-UNSAT, SOFT-UNSAT, and BAD;
    endfor
endfor
print "Weight of unsatisfied clauses is", BAD;
print M;
end Walksat.
```

Figure 1: **The Walksat procedure for weighted MAX-SAT problems.**

the weights of the affected clauses in computing the greedy moves. The parameter HARD-LIMIT is set by the user to indicate that any clause with that weight or greater should be considered to be a hard constraint. The algorithm searches for MAX-FLIPS steps, or until the sum of the weights of the unsatisfied clauses is less than or equal to the TARGET weight. If the target is not reached, then a new initial assignment is chosen and the process repeats MAX-TRIES times. The parameter NOISE controls the amount of stochastic noise in the search, by adjust the ratio of random walk and greedy moves. The best performance on the problems in this paper was found when NOISE $= 0.2$.

Walksat is biased toward satisfying hard constraints before soft constraints. However, while working on the soft constraints, one or more hard constraints may again become unsatisfied. Thus, the search proceeds through a mixture of feasible and infeasible solutions. This is in sharp contrast with standard operations research methods, which generally work by stepping from feasible solution to feasible solution. Such methods are at least guaranteed (by definition) to find a local minimum in the space of feasible solutions. On the other hand, there is no such guarantee for our approach. It therefore becomes an empirical question as to whether local search on a weighted MAX-SAT encoding of problems with both hard and soft constraints would work even moderately well.

Our initial test problems were encodings of airline scheduling problems that had been studied by researchers in constraint logic programming (CLP) (Lever and Richards 1994). The results were encouraging; we found solutions approximately 10 to 100 times faster than the CLP approach. However, for the purposes of the paper, we wished to work on a larger test set, that had been studied more intensively over a longer period of time. We found such a set of benchmark problems in the operations research community, as we describe in the next section.

3 Steiner Tree Problems

Network Steiner tree problem have long been studied in operations research (Hwang *et al.* 1992), and many well-known, hard benchmark instances are available. The problems we used can be obtained by ftp from the OR Repository at Imperial College (mscmga.ms.ic.ac.uk). We ran our experiments on these problems so that our results could be readily compared against those of the best competing approaches. A network Steiner tree problem consists of an undirected graph, where each edge is assigned a positive integer *cost*, and a subset of its nodes, called the Steiner nodes. The goal is to find a subtree of the graph that spans the Steiner nodes, such that the sum of the costs of the edges of the tree is *minimal*. Fig. 2 shows an example of a Steiner problem. The top figure shows the graph, where the Steiner nodes are nodes 1, 2, 3, 6, and 7. The weights are given along the edges. The bottom figure shows a Steiner

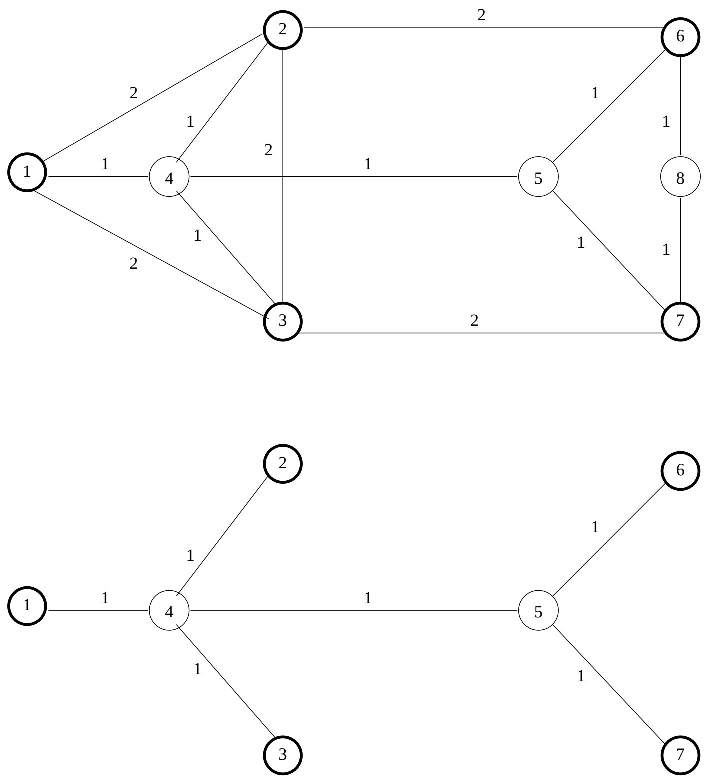
Figure 2: An example of a network Steiner problem and its solution.

tree connecting those nodes. Note that the solution involves two non-Steiner nodes (4 and 5). In general, finding such a Steiner tree is NP-complete.

There is a direct translation of Steiner problems into MAX-SAT. The encoding requires $2|E|^2$ variables, where $|E|$ is the number of edges in the entire graph. While this encoding is of theoretical interest, it is not practical for realistically-sized problems: even a quadratic blowup in the number of variables relative to the number of edges in original instance is simply too large. As we will see below, many of the problems we wish to handle contain over 10,000 edges, and we cannot hope to process a formula containing 100,000,000 variables! Therefore we developed an alternative encoding of Steiner tree problems that is only linearly dependent on the number of edges.

The intuition behind our encoding is that the original problem is broken down into a set of tractable subproblems; a range of near-optimal solutions to the subproblems are pre-computed; and then MAX-SAT is used to combine a selection of solutions to the subproblems to create a global solution. For Steiner tree problems, the subproblems are smaller Steiner trees that connect just *pairs* of nodes from the original Steiner set. Such two-node Steiner problems are tractable, because a solution is simply the shortest path between the nodes. A range of near-optimal solutions, *i.e.* the shortest path, the next shortest path, *etc.*, can be generated using a modified version of Dijkstra's algorithm. This approach actually only approximates the original problem instance, because we do not generate *all* paths between pairs of nodes, but only the *k* shortest paths for some fixed *k*. (We discuss the choice of *k* below.) Pathological problem instances exist that require very non-optimal subproblem solutions. However, we shall see that the approach works quite well in practice.

We illustrate the encoding using the example from Fig. 2. First, we introduce a variable for each edge of the graph. For example, the edge between nodes 1 and 2 is represented by variable $e_{1,2}$. The interpretation of the variable is that if the variable is true, then the corresponding edge is part of the Steiner tree. To capture the cost of including this edge in the tree, we include a unit clause of the form $(\neg e_{1,2})$ with weight 2, the cost of the edge. This clause is soft constraint. Note that when this edge is included in the solution, *i.e.*, $e_{1,2}$ is true, this clause is unsatisfied, so the truth-assignment incurs a cost of 2. Similarly we have a clause for every edge.

Second, we list the Steiner nodes in an arbitrary order, and then for each successive pair of nodes in this list, we generate the k shortest paths between the nodes. We associate a variable with each path. For example, if $k = 2$, then the two shortest paths between Steiner nodes 1 and 2 are 1–2 and 1–4–2. We name the variables $p_{1,2}$ and $p_{1,4,2}$.

Third, we introduce hard constraints that assert that a solution must contain a path between each pair of Steiner nodes. For example, the clause $(p_{1,2} \vee p_{1,4,2})$ is a hard constraint, and therefore assigned a high weight (greater than the sum of all soft constraints). Hard constraints also assert that if a path appears in a solution, then the edges it contains appear. For example, for the path 1-4-2, we introduce the clauses $(p_{1,4,2} \supset e_{1,4})$ and $(p_{1,4,2} \supset e_{4,2})$. This concludes our encoding.

The encoding requires $|\mathrm{E}|+k(|\mathrm{S}|-1)$ variables, where |E| is the number of edges in the graph, |S| is the number of Steiner nodes, and k is the number of shortest paths pre-computed between each pair. The total number of clauses is $O(|E|+kL(|\mathrm{S}|-1))$, where L is the maximum number of edges in any of the pre-computed paths.

4 Empirical Results

A good description of our benchmark problems appears in Beasley (1989). The set contains four classes (B, C, D, E) of problem instances of increasing size and complexity. We omitted class B because the problems are small and easy to solve. Each class has 20 instances.

Tables 1, 2, and 3 contain our results, as well as those of the two best specialized Steiner tree algorithms, as reported Beasley (1989) and Chopra *et al.* (1992). In the table, |V| denotes the number of nodes in the graph, |E| the number of edges, and |S| the number of Steiner nodes. The columns labeled "Soln" give the weight of the best

Steiner tree found by each method. The solutions found by Chopra *et al.* are globally optimal, except for instance E18. For some problems we also give the second best solution (labeled “Soln2”) found by Walksat, to indicate how effective the procedure can be in practice, since it may locate a near-global optimum in a very short time.

Walksat ran on a SGI Challenge with a 150 MHz MIPS R4400 processor. Beasley’s algorithm ran on a Cray XMP, and Chopra’s on a Vax 8700. A hyphen in the table in the case of Beasley’s algorithm indicates that the problem was not solved after 21,600 seconds; in the case of Chopra’s algorithm, it indicates that problem was not solved after 10 days.

We have not attempted to adjust the numbers for machine speed. Caution must be used in comparing different algorithms running on radically different kinds of hardware (the SGI has a RISC architecture, the Vax is CISC, and the Cray is a parallel vector processing machine). The SGI is rated is 136 MIPS, while the Vax is rated at 6 MIPS. This would indicate a ratio of 22 in relative speed; however, at least one user of both machines (Johnson 1994) reports a maximum speedup factor of 15 on combinatorial algorithms, with as small a factor as 3 on large instances. The Cray is rated 230 peak MIPS, which would appear to be faster than the SGI; however, Cray Research also reports that code that performs no vector processing at all runs at only 30 MIPS. Thus, differences in hardware could account for a speedup of between 3 and 22 when comparing Chopra’s VAX to our SGI, and of between 0.6 and 4.5 when comparing Beasley’s Cray to our SGI. In any case, this indicates that all of the differences in performance described below cannot be attributed entirely to differences in machine speed.

We found that we could obtain good solutions with a value of k, the number of pre-computed paths between pairs of nodes, of up to 150 for the smaller instances ($\leq$ 10 Steiner nodes), and up to 20 for the larger instances. The timing results for Walksat are averaged over 10 runs.

The running times in the table do not include the time to pre-compute the set of paths between successive Steiner nodes. This is reasonable because in practice one often deals with a fixed network, and wants to compute Steiner trees for many different subsets of nodes. For example, in teleconferencing applications, the network is fixed, and each problem instance involves finding a Steiner tree to connect a set of sites. Given a fixed network, one can pre-compute, using Dijkstra’s algorithm, sets of paths between every pair of nodes.

From the tables we can see that for problems with up to 10 Steiner nodes, Walksat usually find an optimal solution at least as fast as the other two approaches, even allowing differences in machine speeds. For example, for D1 and D2, Walksat is about 100 times faster than the other two in reaching the global optimum. For D6, Walksat runs about 50 times faster than Beasley and 30 times faster than Chopra. The difference is particularly dramatic for E1, where Walksat finds the optimal solution in less than 1 second, and Beasley and Chopra both take over 1,000 seconds. On

Problem Parameters				Beasley		Chopra *et al.*		Walksat			
ID	$\|V\|$	$\|E\|$	$\|S\|$	Soln	CPU secs (Cray XMP)	Soln	CPU secs (Vax 8700)	Soln1	CPU (SGI)	Soln2	CPU (SGI)
C1	500	625	5	85	113.57	85	27.3	85	1.11		
C2			10	144	5.84	144	811.7	144	72.69	146	30.57
C3			83	766	152.78	754	543.4	808	0.05		
C4			125	1094	3.61	1079	509.6	1128	0.09		
C5			250	1594	2.73	1579	473.9	1654	0.12		
C6		1000	5	55	48.55	55	48.9	55	3.41		
C7			10	106	4.44	102	83.2	102	3.02	103	2.95
C8			83	524	8.63	509	674.4	553	0.07		
C9			125	722	198.97	707	1866.3	754	0.09		
C10			250	1112	4.53	1093	245.6	1169	0.16		
C11		2500	5	34	188.02	32	333.3	32	0.44	34	0.22
C12			10	48	25.04	46	119.8	46	65.64	47	39.41
C13			83	265	166.53	258	9170.3	286	0.23		
C14			125	336	8.67	323	211.7	349	0.25		
C15			250	563	7.30	556	210.6	587	0.40		
C16		12500	5	11	32.37	11	10.1	11	6.25		
C17			10	20	24.17	18	98.0	18	19.50	19	6.52
C18			83	123	104.34	113	45847.7	130	4.89		
C19			125	155	86.48	146	116.9	165	5.25		
C20			250	269	157.80	267	14.9	278	5.79		

Table 1: Computation Results for Beasley's C class Steiner Tree Problems

E2, Walksat takes about 800 seconds to reach the global optimum 214, which is comparable to Chopra's 6000 seconds (a ratio of 7.5). Walksat takes only about 28 seconds to reach a tree with weight 216, compared to Beasley who takes 7000 seconds to reach only 231. On E6, Walksat takes less than 2 seconds, compared to over 670 seconds for Chopra. A near-optimal solution takes less than 1 seconds, compared to 1700 seconds for Beasley.

Surprisingly, Walksat can locate some of the optimal and near-optimal solutions for the large E-class instances that cannot be found by Beasley in a reasonable amount of time. For example, for E12, Walksat finds a local optima of 68 which was not reached by Beasley within the time limit of 21,600 seconds. For E7, Walksat finds the global optimum of 145, while Beasley only reaches 157.

On problems with a larger numbers of Steiner nodes, Walksat usually produces less optimal solutions than the other two methods. The problem Walksat has on instances with a large number of Steiner nodes may due to the fact that the MAX-

Problem Parameters				Beasley		Chopra *et al.*		Walksat			
ID	$\|V\|$	$\|E\|$	$\|S\|$	Soln	CPU secs (Cray XMP)	Soln	CPU secs (Vax 8700)	Soln1	CPU (SGI)	Soln2	CPU (SGI)
D1	1000	1250	5	107	226.27	106	475.6	106	2.61	107	0.85
D2			10	228	252.47	220	283.5	220	1.54	227	0.98
D3			167	1599	21.85	1565	2290.1	1646	0.21		
D4			250	2170	11.71	1935	3529.0	2044	0.28		
D5			500	3360	11.76	3250	810.6	3419	0.53		
D6		2000	5	71	4065.69	67	2339.5	67	75.51	70	12.37
D7			10	103	18.71	103	99.7	103	0.47		
D8			167	1108	475.14	1072	6984.5	1180	0.35		
D9			250	1684	243.48	1448	4629.7	1585	0.41		
D10			500	2235	20.21	2110	1312.1	2219	0.72		
D11		5000	5	31	3290.48	29	1374.4	29	2.78	30	2.07
D12			10	42	48.04	42	305.0	42	0.79		
D13			167	520	36.06	500	1864.0	544	1.07		
D14			250	688	443.26	667	3538.4	740	0.74		
D15			500	1208	32.25	1116	1409.7	1193	1.70		
D16		25000	5	14	161.43	13	871.3	13	18.29		
D17			10	25	277.20	23	6965.2	23	735	24	20
D18			167	247	222.15	223	245192.1	262	20.48		
D19			250	384	256.15	310	878.3	359	21.52		
D20			500	544	1023.60	537	47.1	558	24.45		

Table 2: Computation Results for Beasley's D class Steiner Tree Problems

SAT encodings simply become too large to be processed efficiently. (For example, the number of flips per second goes down significantly on very large formulas.) Nonetheless, given the fact that Walksat is a completely general algorithm, as opposed to the specialized algorithms of Beasley and Chopra, it performs surprisingly well on these hard benchmark problems.

It is important to note that Walksat scales up to problems based on large graphs, especially when the set of Steiner nodes is relatively small. This should be contrasted with some other local-search style approaches to solving Steiner trees using simulated annealing (Dowsland 1991) and genetic algorithms (Kapsalis *et al.* 1993). Despite the fact that these local search algorithms were designed specifically for solving Steiner problems, they can only handle the smallest instances in the B and C classes. This has led Hwang *et al.* (page 172) to conclude that simulated annealing and hill-climbing (a form of local search) are ill-suited for Steiner tree problems. However, our work demonstrates that local search can in fact be successful for Steiner problems.

Problem Parameters				Beasley		Chopra *et al.*		Walksat			
ID	$\|V\|$	$\|E\|$	$\|S\|$	Soln	CPU secs (Cray XMP)	Soln	CPU secs (Vax 8700)	Soln1	CPU (SGI)	Soln2	CPU (SGI)
E1	1000	3250	5	115	1116.80	111	1149.6	111	0.54	113	0.35
E2			10	231	7124.10	214	6251.2	214	817.70	216	28.13
E3			417	4131	1346.05	4013	26468.4	4282	1.43		
E4			625	5208	378.66	5101	46007.6	5398	2.10		
E5			1250	8413	98.22	8128	12564.1	8518	3.95		
E6		5000	5	78	1760.49	73	678.0	73	1.71	78	0.81
E7			10	157	——	145	27124.0	145	5170.50	149	275.62
E8			417	2733	4459.30	2640	118617.5	2899	2.05		
E9			625	3721	18818,53	3604	24527.8	3913	2.65		
E10			1250	5899	311.57	5600	39260.7	5957	4.94		
E11		12500	5	39	3061.45	34	1900.6	34	622.71	35	7.47
E12			10	69	——	67	7199.7	68	5325.67	69	374.79
E13			417	1336	——	1280	207058.6	1417	7.21		
E14			625	1773	——	1732	29262.6	1884	8.69		
E15			1250	3008	457.98	2784	7666.0	3125	157.67		
E16		62500	5	15	7880.40	15	179.0	15	352.26	16	117.26
E17			10	26	445.69	25	36039.9	27	160.92		
E18			417	840	——	(563.03)	—	667	129.33		
E19			625	923	——	758	6371.8	853	132.70		
E20			1250	1376	14037.13	1342	272.2	1400	160.97		

Table 3: Computation Results for Beasley's E class Steiner Tree Problems

Our positive results are due to both an effective problem encoding and the use of an efficient implementation of our search procedure with a good stochastic technique for escaping from local minima.

5 Discussion and Conclusions

In this paper, we have shown how to adapt Walksat, a variant of the GSAT satisfiability testing algorithm, to handle *weighted* MAX-SAT problems. One of the problems in encoding optimization problems as propositional satisfiability problems is the difficulty of representing both hard and soft constraints. In a weighted MAX-SAT encoding, hard constraints simply receive a high weight (for example, larger than the sum of the soft constraints). Any solution where the sum of the weights of the violated clauses is less than that of any hard constraint is guaranteed to be feasible (*i.e.*, satisfies all hard constraints).

Another problem with translating optimization problems into satisfiability problems is handling numeric information. Even though in principle a polynomial transformation often exists, SAT encodings of realistic problem instances may become too large to solve. In our weighted MAX-SAT encoding, much of the numeric information in the problem instances can be captured effectively in the clause weights.

In order to test this approach, we considered a set of hard benchmark Steiner tree problems, and compared our results to specialized state-of-the-art algorithms. We chose the Steiner tree problem because of its long history and the public availability of a well-established set of benchmark instances. Our results showed that our weighted MAX-SAT strategy is competitive with specialized algorithms, especially on (possibly large and computationally difficult) instances involving small numbers of Steiner nodes. We must stress that we are not arguing that our approach is *the* best way to find Steiner trees. It is certainly the case that every particular class of combinatorial problems has some structure that can be best exploited by some specialized algorithm. The significance of our experiments is that they showed good performance using a completely general algorithm, that incorporates no heuristics specific to Steiner tree problems.

As mentioned above, the search performed by Walksat proceeds through truth-assignments that correspond to both feasible and infeasible solutions to the original optimization problem. This is an inherent aspect of our approach, simply because feasible solutions of the original problem may be several variable “flips” apart. Note that in constructing *specialized* local search algorithms for particular problem domains, one generally makes larger changes and only moves between feasible solutions. It is therefore surprising to discover how well Walksat performs. It is important to note that negative performance results would have argued against our overall approach of using a domain-independent logical representation with a general search procedure such as Walksat.

Part of the success of the approach is due to the particular MAX-SAT encoding we developed for the problems. In particular, our encoding is significantly shorter than a more direct one. The general approach we used, which is based on combining solutions from tractable subproblems, could also be useful for encoding other kinds of optimization problems. In particular, Crawford and Baker (1994) have observed that a direct SAT encoding of job-shop scheduling problems leads to formulas that are very large and hard to solve. It would be interesting to see if our piecewise encoding technique is applicable in the job-shop scheduling domain.

In conclusion, we have demonstrated that the use of efficient MAX-SAT encodings with a domain-independent stochastic local search algorithm is a promising approach for solving hard optimization problems in AI and operations research.

References

Adorf, H. M., and Johnston, M. D. (1990) A discrete stochastic neural network algorithm for constraint satisfaction problems. *Proceedings of the International Joint Conference on Neural Networks*, San Diego, CA.

Beasley, J. (1989) An SST-based algorithm for the Steiner Tree problems in graphs. *Networks* 19, 1-16.

Chopra, S., Gorres, E., and Rao, M. (1992) Solving the Steiner Tree problem on a graph using branch and cut. *ORSA Journal on Computing* 4(3), 3-18.

Crawford, J. M., and Baker, A.B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. *Proceedings AAAI-94*, Seattle, WA, 1092-1097.

Davis, M., and Putnam, H. (1960). A computing procedure for quantification theory. *J. Assoc. Comput. Mach.* 7, 201–215.

Dowsland, K. (1991) Hill-climbing simulated annealing and the Steiner problem in graphs. *Eng. Opt.* 17, 91-107.

Ginsberg, M. and McAllester, D. (1994) GSAT and dynamic backtracking. *Proceedings KR-94*, Bonn, Germany, 226–237.

Ginsberg, M. (1994) Organizational meeting for AI/OR initiative, Oct. 1994.

Green, C. (1969) Application of theorem proving to problem solving. *Proceedings IJCAI-69*, Washington, DC, 219–239.

Gu, J. (1992) Efficient local search for very large-scale satisfiability problems. *Sigart Bulletin* 3(1), 8–12.

Hansen, P., and Jaumard, B. (1990) Algorithms for the maximum satisfiability problem. *Computing* 44, 279–303.

Hwang, F.K, Richards, D.S., and Winter, P. (1992) *The Steiner Tree Problem*, Amsterdam: North-Holland (Elsevier Science Publishers).

Johnson, D.S., (1994) Personal communication.

Kapsalis, A., Rayward-Smith, V., and Smith, G. (1993) Solving the graphical Steiner tree problem using genetic algorithms. *J. Oper. Res. Soc.* 44(4), 397-406.

Kautz, H., and Selman, B. (1992). Planning as satisfiability. *Proceedings ECAI-92*, Vienna, Austria.

Lever, J., and Richards, B. (1994) A CLP approach to flight scheduling problems. *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, 1994.

Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1990) Solving large-scale constraint satisfaction an scheduling problems using a heuristic repair method. *Proceedings AAAI-90*, Boston, MA, 17–24.

Papadimitriou, C.H., and Steiglitz, K. (1982) *Combinatorial Optimization.* Englewood Cliffs, NJ: Prentice-Hall.

Selman, B., Levesque, H.J., and Mitchell, D.G. (1992) A new method for solving hard satisfiability problems. *Proceedings AAAI-92,* San Jose, CA, 440–446.

Selman, B., and Kautz, H. (1993a) Domain-independent extensions to GSAT: solving large structured satisfiability problems. *Proceedings IJCAI-93*, Chambéry, France, 290–295.

Selman, B. and Kautz, H. (1993b) An empirical study of greedy local search for satisfiability testing. *Proceedings AAAI-93*, Washington, DC, 46–51.

Selman, B., Kautz, H., and Cohen, B. (1994) Noise strategies for local search. *Proceedings AAAI-94*, Seattle, WA, 1994.

Trick, M., and Johnson, D.S. (Eds.) (1993) Working notes of the DIMACS Algorithm Implementation Challenge, Rutgers University, New Brunswick, NJ.