DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 00, 0000

A General Stochastic Approach to Solving Problems with Hard and Soft Constraints

HENRY KAUTZ, BART SELMAN, AND YUEYEN JIANG

ABSTRACT. Many AI problems can be conveniently encoded as discrete constraint satisfaction problems. It is often the case that not all solutions to a CSP are equally desirable — in general, one is interested in a set of "preferred" solutions (for example, solutions that minimize some cost function). Preferences can be encoded by incorporating "soft" constraints in the problem instance. We show how both hard and soft 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 a local-search algorithm for satisfiability to handle weighted MAX-SAT. To demonstrate the effectiveness of our approach, we present experimental results on encodings of a set of well-studied network Steiner-tree problems. This approach turns out to be competitive with some of the best current specialized algorithms developed in operations research.

1. Introduction

Traditional satisfiability-testing algorithms are based on backtracking search [6]. 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 [27]. Selman, Levesque, and Mitchell [22] and Gu [9, 10] introduced an alternative approach for satisfiability testing, based on stochastic local search. The algorithms in this family are partial decision procedures — they cannot be used to prove that a formula is unsatisfiable, but only find models of satisfiable ones. For many interesting classes of formulas, they can solve problem instances that are two orders of magnitude larger than those that can be handled by any systematic search algorithm [23]. A recent variant, called "Walksat" [25], is currently one of the fastest and most robust versions of the basic algorithm [26].

1991 *Mathematics Subject Classification.* 90C09, 90B12 68R05, 68R10, 69T15.

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 [**24**]. 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. 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.

We have modified the Walksat algorithm mentioned above 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. Local search is, in general,

a widely applicable heuristic method for finding approximate solutions to optimization problems, and versions of it have been previously applied to MAX-SAT [**12**]. Our algorithm is notable for employing a particularly simple, efficient, and effective method for escaping from local minima in the search space.

The main experimental work described in this paper is on Boolean encodings of *network Steiner tree problems*. It is an NP-complete problem involving finding paths in graphs, as will be explained below. Network Steiner tree problems have many applications in network design and routing, and have been intensively studied in operations research for several decades [**13**]. 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 [**8**].

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, precompute 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. The approach is called "partitioning" in independent work by Gu and Puri [**11, 21**] in applying SAT to circuit synthesis problems.

2. A Stochastic Search Algorithm

Local search algorithms for satisfiability work by searching through the space of truth assignments for one that satisfies all clauses [**22**]. 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 minimizes the number of unsatisfied clauses; such moves are called "greedy" moves. Local minima are avoided by occasionally

```
Weighted-Walksat:
input: weighted-clauses, max-flips, max-tries, target, noise.

soln := a random truth assignment;
for i := 1 to max-tries
    for j := 1 to max-flips
        compute the sum of the weights of clauses unsatisfied by soln;
        if the sum ≤ target then
            return "Success, solution is", soln;
        c := a randomly chosen unsatisfied clause;
        with probability noise
            flip value assigned by soln to a randomly chosen variable from c;
        with probability (1-noise)
            for each variable in c, compute the sum of the weights of the
                clauses currently satisfied by soln that would become
                unsatisfied if that variable were flipped;
            flip a variable in c that minimizes that sum;
    end;
end;
return "Failure, best assignment found is", soln;
```

FIGURE 1. The Weighted-Walksat procedure.

randomly flipping a variable that appears in some unsatisfied clause [**23**]; such moves are called "random walk" moves. The frequency with which random walk moves are made is controlled by a "noise" parameter.

Fig. 1 presents a version of the Walksat algorithm [**25**], here modified for weighted MAX-SAT problems. The change simply involves taking the sum of the weights of the set of unsatisfied clauses, rather than merely counting the number of such clauses. A new parameter, "target", provides a threshhold for terminating the algorithm when the current assigned is "good enough". Hard constraints are encoded in clauses by giving each a weight greater than the target.

Because hard constraints have greater weight than soft constraints, Weighted-Walksat is biased toward satisfying those constraints first. 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) [**18**]. 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 [**13**], 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. (Recently Khoury *et. al.* [**16**] have made available a test problem generator that creates hard random Steiner tree problem instances; in future work we hope to also compare our approach with others on instances from this source as well.)

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 "terminal" nodes. The goal is to find a subtree of the graph that spans the terminal nodes, such that the sum of the costs of the edges of the tree is the *minimum*. The nodes in this subgraph that are not terminals are called the "Steiner nodes". 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 tree connecting those nodes. Note that the solution involves two Steiner nodes (4 and 5). In general, finding such a Steiner tree is NP-complete.

3.1. Exact Encodings. The exact translation of Steiner problems into MAX-SAT is not entirely obvious, because of the difficulty in axiomatizing "connectedness". The following translation requires $O(2|E|^2)$ variables, where $|E|$ is the number of edges in the entire graph. First, replace every edge e in the graph by a pair of *directed* edges, f and g. A Steiner tree will then correspond to a *tour* of a subgraph that includes all the Steiner nodes. The tour corresponds to walking around the perimeter of the Steiner tree — equivalently, making a depth-first traversal of the tree. Let there be a Boolean variable for every possible position that every possible edge could take in the tour. (Of course, some edges do not appear in the tour at all.) For example, the proposition f_3^6 would be true if one of the directed edges derived from the original edge e_3 appears as the 6th element of the tour. This requires $(2|E|)^2$ variables. Then we include a set of hard constraints that state that the directed-edge variables that are "true" make up a tour that includes the terminal nodes, where parts of the tour may be repeated. Finally, a set of soft constraints represents the costs assigned to the edges. For example, if edge e_3 has cost 30, then one simply includes the unit clause $(\neg e_3)$ with weight 30 in the MAX-SAT encoding.

Unfortunately this encoding 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 too large. Many of the problems we wish to handle

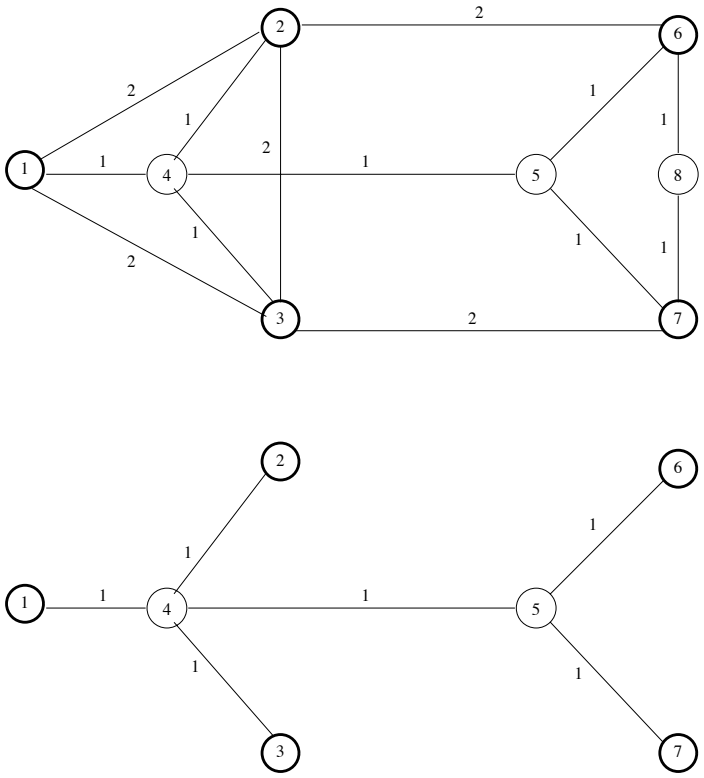

FIGURE 2. An example of a network Steiner problem and its solution.

contain over 10,000 edges, and we cannot hope to solve a formula containing 100,000,000 variables on current machines.

3.2. Approximate Encodings. The desire to handle such large instances led us to search for more compact MAX-SAT encodings. Ultimately we found an encoding that is in *linear* in two factors, the number of edges in the original problem instance, and a parameter k, which controls how well the encoding *approximates* the original problem instance. We say the encodings are approximate because the solutions they yield are not *guaranteed* to be optimal solutions to the original instances, except when k is very large — that is, $O(2^{|E|})$. In practice, however, we have found good quality solutions, and often optimal solutions, when k is quite small (*e.g.*, $k \leq 30$ on problems with up to 25,000 edges).

The intuition behind our encoding is that the original problem can be 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 terminal 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 make a list of the terminal nodes, and then for each successive pair of nodes in this list, we generate the k shortest paths between the nodes. (We will consider the issue of how the nodes should be ordered in this list below.) We associate a variable with each path. For example, if $k = 2$, then the two shortest paths between terminal 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 terminal 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 the encoding.

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

3.3. Selecting Subproblems. If k is so large that all of the paths between each pair of terminal nodes are included in each subproblem, then the solutions to the encodings are sure to be optimal for the original problem instances. But it is obviously impractical to make k so large — indeed, the MAX-SAT encoding would be exponentially larger than the original problems, thus defeating the entire approach. The key, then, is to devise a way of selecting the subproblems so that it is likely that a good global solution can be assembled even when only

a few solutions are generated for each subproblem.

Given a set T of terminal nodes, we have to choose $|T-1|$ pairs of nodes. We will consider three strategies for chosing pairs.

a) Random. Begin with an arbitrary linear ordering of the terminal nodes. Choose pairs from every adjacent pair of nodes in the ordering. For example, in Fig. 2, an ordering of $< 1, 3, 7, 6, 2 >$ would correspond to selecting the pairs $(1, 3)$, $(3, 7)$, $(7, 6)$, and $(6, 2)$.

b) Greedy. Again begin with an arbitrary linear ordering of the terminal nodes. Step through this list constructing pairs according to the following algorithm:

```
for each terminal node i (except the last in the ordering)
    find nearest terminal node j that does not appear earlier
        in the ordering;
    output pair (i, j);
end for;
```

The step of finding the nearest terminal node j to a node i can be computed using Dijkstra's single-source shortest paths algorithm in $O(|V|^2)$ time. Thus the overall time required is $O(|V|^2|T|)$.

c. Minimum Spanning Tree. Construct a matrix with the distances between all pairs of terminal nodes. This can be done by using the Floyd-Warshall all pairs shortest path algorithm in $O(|V|^3)$ time, or by using Johnson's algorithm for sparse graphs in $O(|V|^2 \log |V| + |V||E|)$ time. This defines a complete weighted graph, G, on the pairs of terminal nodes. Each edge weight gives the length of the shortest path between two terminal nodes. Now, find a minimum spanning tree of G. Each edge in the spanning tree defines a pair of nodes.

Note how the third strategy takes a more "global view" of the pair selection process than the first two strategies. Selecting the shortest path between each pair of nodes would give us an initial approximation of a Steiner tree connecting the terminal nodes. The Greedy heuristic can be viewed as computing a "pretty good" spanning tree over the terminal nodes, but it is not a true minimum spanning tree algorithm. One would expect that the minimum spanning tree strategy would give the best initial approximation, and the Random strategy the worst. This is confirmed in our experiments below. By allowing more than one path between each pair of nodes, Walksat can start looking for paths that fit together better (note that paths can share edges), thereby reducing the overall weight of the connecting spanning tree. The question is which ordering strategy allows us to find the best global solutions.

4. Empirical Results

A good description of our benchmark problems appears in Beasley (1989) [**2**]. The set contains four classes (B, C, D, E) of problem instances of increasing size

and complexity. The instances in the B and C class are relatively small and easy to solve. The D and E classes contain many hard benchmark instances. In order to keep our presentation concise, we limit our discussion to the results on the D class instances; we obtained qualitatively similar results for the E class.[1]

Table 1 contains our results on the D class instances, as well as those of the two of the best specialized Steiner tree algorithms, as reported in papers by Beasley [**2**] and Chopra [**3**]. In the table, $|V|$ denotes the number of nodes in the graph, $|E|$ the number of edges, and $|T|$ the number of terminal 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. In the column marked "basis/number paths", we give the subproblem selection strategy (G for greedy; M for minimum spanning tree) with which we obtained the best results. We also give the number of path used between pairs of nodes.

As we have discussed above, our Walksat algorithm is inherently incomplete, although it did find optimal solutions for many of the instances. Chopra's algorithm is complete. Beasley's algorithm is complete in theory, but in practice is incomplete, because it could not be be run to termination due to time and space limitations on many of the instances. Recently Khoury and Pardalos [**17**] have invented a new, complete algorithm for the Steiner problem, that also appears to be quite robust and efficient; in future work, we plan to provide comparisons with their method as well.

Walksat ran on a SGI Challenge with a 150 MHz MIPS R4400 processor. (Execution times for Walksat on Sun Sparc 20 series machines with similar clock speeds are virtually identical.) Beasley's algorithm ran on Cray XMP, and Chopra's on a Vax 8700. Note that we have not attempted to adjust the numbers for machine speed. However, it is unlikely that all of the differences in performance described below can be attributed entirely to differences in machine speed. The running times in the table do not include the time to pre-compute the set of paths between successive terminal 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. Given a fixed network, one can pre-compute, using Dijkstra's algorithm, sets of paths between every pair of nodes.

From Table 1, we can see that for problems with up to 10 terminal nodes, Walksat always found an *optimal* solution more quickly than the other two approaches. 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 fact that we found the optimal solutions in these cases means two things. First, it shows that the subproblem decomposition contains the global optimal solution. Second, it shows that Weighted Walksat was able to synthesize this global optimal solution; or in other words, it solved the weighted MAX-SAT

[1]Experimental data and code is available via ftp and via the home page of the first author.

Problem Parameters			Beasley		Chopra *et al.*		Walksat		
ID	$\|E\|$	$\|T\|$	soln	secs (Cray)	soln	secs (Vax)	soln	basis / num. paths	CPU (SGI)
D1	1250	5	107	226	106	475	106	G,M / 10	3
D2		10	228	252	220	283	220	G,M / 5	2
D3		167	1599	21	1565	2290	1640	M / 1	1
D4		250	2170	11	1935	3529	2008	M / 1	1
D6	2000	5	71	4065	67	2339	67	G / 30	75
D7		10	103	18	103	99	103	G,M / 5	1
D8		167	1108	475	1072	6984	1156	M / 1	1
D9		250	1684	243	1448	4629	1540	M / 1	1
D11	5000	5	31	3290	29	1374	29	G / 30	3
D12		10	42	48	42	305	42	G,M / 10	2
D13		167	520	36	500	1864	535	M / 1	1
D14		250	688	443	667	3538	702	M / 1	1
D16	25000	5	14	161	13	871	13	G,M / 10	75
D17		10	25	277	23	6965	23	G / 25	1
D18		167	247	222	223	245192	251	M / 1	1
D19		250	384	256	310	878	342	M / 1	1

TABLE 1. Computation Results for Beasley's D class Steiner Tree Problems

problem to optimality.[2]

On the instances with a larger number of terminal nodes, we found solutions that were not optimal, but were often better than those found by Beasley (see D4, D9, and D19). However, in these cases we used encodings that only had a single path between terminal nodes, so all the work was essentially done by the terminal ordering heuristic — that is, by the selection of subproblems. This is because the MAX-SAT encodings using larger numbers of paths between terminals simply became too large for the Walksat procedure to handle.

In Table 2, we compare the the various subproblem selection strategies on selection of the instances.

Consider the 1-path solution column. This gives the global solution obtained by simply combining the single best solution to each subproblem. Thus, it reflects the quality of the terminal ordering heuristics alone. As we expected, the minimum spanning tree heuristic yields the best solutions, followed by the greedy heuristic, with the random ordering far behind.

The story changes when we used Walksat to synthesis global solutions by picking from several solutions to each subproblem. In this case, the greedy heuristic gives the best global solutions. Sometimes the minimum spanning tree heuristic is as good, but not in all cases — see D11. The random heuristic is generally the worst. However, it is interesting to note the data for D17, where

[2]Recently Davenport [5] applied a complete branch and bound MAX-SAT algorithm to the encodings we generated, including much of the data not discussed here. In every case he considered, he discovered that Weighted Walksat had indeed found the optimal solution to the MAX-SAT instance, and that in most cases the branch and bound algorithm required much more running time.

	Random			Greedy			Min. Span. Tree		
ID	1 path soln	best soln found	best num paths	1 path soln	best soln found	best num paths	1 path soln	best soln found	best num paths
D11	41	30	30	32	29	30	31	30	50
D12	67	44	30	42	42	1	42	42	1
D13	1252	–	–	544	–	–	535	–	–
D14	1844	–	–	740	–	–	702	–	–
D16	17	13	30	16	13	10	16	13	10
D17	33	25	30	26	23	25	26	26	50
D18	642	–	–	262	–	–	251	–	–
D19	925	–	–	359	–	–	342	–	–

TABLE 2. Comparing subproblem selection strategies.

random actually outperforms the minimum spanning tree heuristic.

One may be surprised to see the superiority of the simple greedy heuristic over the minimum spanning heuristic, when local search is used to combine multiple local solutions. The intuitive explanation for this that the minimum spanning tree decomposition corresponds to a deep local minimum in the *global* solution space. It is difficult for local search to escape from this local minimum. On the other hand, the greedy decomposition allows more room for improvement.

This is an interesting lesson for local search methods: Trying to generate very good initial solutions may actually hurt the final result! The difference between greedy and random shows that some degree of preprocessing can be beneficial, if it is not taken too far.

In summary, our results show that Walksat is competitive with the best specialized methods, and is often considerably faster. On problems with large numbers of terminal nodes our encodings become too large to handle effectively. However, it is important to note the problems we do handle are still very large, based on graphs containing up to 25,000 nodes (including non-terminal nodes). Real world applications of Steiner trees often share this characteristic (for example, in setting up a real-time connection between a small group of computers among a network of thousands of machines).

As noted above, we used Steiner tree problems to evaluate solving MAX-SAT encodings with Weighted Walksat because it's a well-defined problem, and there is well-known collection of hard benchmark instances. 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. Given its success in this domain, we believe that this is a promising approach for tackling other other problems in AI and OR that involve both hard and soft constraints combined with numeric information.

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. Such numeric information is a crucial part in many AI applications, for example in representing utility functions, preference criteria, and probabilities. 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 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, summarized in Table 1 show that our weighted MAX-SAT strategy is in fact competitive with and sometimes superior to the specialized algorithms. It should be stressed that our approach is general, in the sense that our Walksat algorithm incorporates no heuristics specific to Steiner tree problems.

Our experiments also provided some general insights into the issue of problem decomposition. We considered three different methods for a breaking a Steiner tree problem into local subproblems. While a completely random decomposition was not effective, we also found that a too sophisticated approach can create deep local minima, that are hard to improve upon. The best approach appears to be the middle ground: The best *global* solutions can be found by combining a range of solutions to subproblems created by a somewhat sub-optimal decomposition.

As mentioned above, the search performed by Weighted 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. Interestingly, we found that Weighted Walksat does not get stuck in infeasible states. Note that in constructing *specialized* local search algorithms for particular problem domains, one generally makes larger changes and only moves between feasible solutions. Our results suggest that, by contrast, it is not always necessary to restrict search to the feasible region of a problem space.

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 en-

coding other kinds of optimization problems. In particular, Crawford and Baker [4] 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 problems in AI that involve both hard and soft constraints and numeric information.

References

1. H.M. Adorf and M.D. Johnston, *A discrete stochastic neural network algorithm for constraint satisfaction problems.* Proceedings of the International Joint Conference on Neural Networks, San Diego, CA, 1990.
2. J. Beasley, *An SST-based algorithm for the Steiner Tree problems in graphs*, Networks, 19 (1989), 1-16.
3. S. Chopra, E. Gorres, and M. Rao, *Solving the Steiner Tree problem on a graph using branch and cut*, ORSA Journal on Computing, 4(3) (1982), 3-18.
4. J.M. Crawford and A.B. Baker, *Experimental results on the application of satisfiability algorithms to scheduling problems*, Proceedings AAAI-94, Seattle, WA, 1994, 1092-1097.
5. A. Davenport, *Panel on Systematic versus Stochastic Methods*, First International Workshop on AI and OR, Timberline, OR, 1995.
6. M. Davis, and H. Putnam *A computing procedure for quantification theory*, J. Assoc. Comput. Mach., 7 (1960), 201–215.
7. K. Dowsland, *Hill-climbing simulated annealing and the Steiner problem in graphs*, Eng. Opt., 17 (1991), 91-107.
8. Organizational meeting for AI/OR initiative, Oct. 1994.
9. J. Gu, *Efficient local search for very large-scale satisfiability problems*, Sigart Bulletin, 3(1) (1992), 8–12.
10. J. Gu, *Efficient local search for very large-scale satisfiability problems*, IEEE Trans. on Systems, Man, and Cybernetics, 23(4) (1993), 1108–1129.
11. J. Gu and R. Puri, *Asynchronous Circuit Synthesis by Boolean Satisfiability*, IEEE Transactions on CAD of Integrated Circuits and Systems, 14(8) (1995), 961–973.
12. P. Hansen, and B. Jaumard, *Algorithms for the maximum satisfiability problem*, Computing, 44 (1990), 279–303.
13. F.K. Hwang, D.S. Richards, P. Winter, *The Steiner Tree Problem*, North-Holland (Elsevier Science Publishers), Amsterdam, 1992.
14. A. Kapsalis, V. Rayward-Smith, and G. Smith, G. *Solving the graphical Steiner tree problem using genetic algorithms*, J. Oper. Res. Soc., 44(4) (1993), 397–406.
15. H. Kautz, and B. Selman, *Planning as satisfiability*, Proceedings ECAI-92,, Vienna, Austria, 1992.
16. B.N. Khoury, P.M. Pardalos, and D.Z. Du, *A test problem generator for the Steiner problem in graphs*, ACM Transactions on Mathematical Software, 19(4) (1993), 509-522.
17. B.N. Khoury, B.N. and P.M. Pardalos, *A heuristic for the Steiner Problem in Graphs*, Computational Optimization and Applications, 6 (1995), 5-14.
18. J. Lever and B. Richards, *A CLP approach to flight scheduling problems*, Proceedings of the International Symposium on Methodologies for Intelligent Systems, 1994.
19. S. Minton, M.D. Johnston, A.B. Philips, and P. Laird, *Solving large-scale constraint satisfaction an scheduling problems using a heuristic repair method*, Proceedings AAAI-90, Boston, MA, 17–24, 1990.
20. C.H. Papadimitriou, and K. Steiglitz, *Combinatorial Optimization.* Prentice-Hall, Englewood Cliffs, NJ, 1992.

21. R. Puri, and J. Gu, *A Modular Partitioning Approach for Asynchronous Circuit Synthesis*, Proc. 28th IEEE/ACM Design Automation Conference, San Diego, 63-69, 1994.
22. B. Selman, H.J. Levesque, and D.G. Mitchell, *A new method for solving hard satisfiability problems*, Proceedings AAAI-92, San Jose, CA, 440–446, 1992.
23. B. Selman and H. Kautz, *Domain-independent extensions to GSAT: solving large structured satisfiability problems*, Proceedings IJCAI-93, Chambéry, France, 290–295, 1993.
24. B. Selman and H. Kautz, *An empirical study of greedy local search for satisfiability testing*, Proceedings AAAI-93, Washington, DC, 46–51, 1993.
25. B. Selman, H. Kautz, and B. Cohen, *Noise strategies for local search*, Proceedings AAAI-94, Seattle, WA, 1994.
26. B. Selman, H. Kautz, and B. Cohen, *Local Search Strategies for Satisfiability Testing*. In Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, D. S. Johnson and M. A. Trick (eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, American Mathematical Society, 1996.
27. M. Trick and D.S. Johnson (eds.), *Working notes of the DIMACS Algorithm Implementation Challenge*. Rutgers University, New Brunswick, NJ, 1993.

AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
E-mail address: kautz@research.att.com

AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
E-mail address: selman@research.att.com

AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
Current address: Pacific Telesis Tecnologies Laboratories, 5000 Executive Pky, 333, San Ramon, CA 94583
E-mail address: jjiang@ttl.pactel.com