Applying Local Search to Disjunctive Temporal Problems

Michael D. Moffitt and Martha E. Pollack
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109, USA
{mmoffitt, pollackm}@eecs.umich.edu

Abstract

We present a method for applying local search to overconstrained instances of the Disjunctive Temporal Problem (DTP). Our objective is to generate high quality solutions (i.e., solutions that violate few constraints) in as little time as possible. The technique presented here differs markedly from previous work on DTPs, as it operates within the total assignment space of the underlying CSP rather than the partial assignment space of the related meta-CSP. We provide experimental results demonstrating that the use of local search leads to substantially improved performance over systematic methods.

1 Introduction

Previous work on temporal reasoning has focused primarily on exact and complete methods for efficiently solving temporal problems (e.g., [Stergiou and Koubarakis, 1998; Tsamardinos and Pollack, 2003; Armando *et al.*, 2004]). The purpose of these algorithms is to find a solution that satisfies a set of constraints, or else prove, by means of exhaustive search, that no solution exists. Recently, there has also been work on temporal formalisms where the goal is instead to optimize an objective function; for instance, to maximize a user's preference when the problem permits many solutions [Khatib *et al.*, 2003; Peintner and Pollack, 2004; Morris *et al.*, 2004], or to minimize the number of violated constraints when no complete solution exists [Moffitt and Pollack, 2005]. Although these algorithms are guaranteed to find optimal solutions, they are computationally expensive, and hence may not be applicable to large problems. Given this, it may be advantageous to instead search for approximate solutions when strict optimality is not required.

Local search algorithms are known to be among the most effective methods for solving computationally intractable problems such as propositional satisfiability, scheduling, and constraint satisfaction [Hoos and Stutzle, 2004]. While they do not guarantee optimality, they are often able to produce high-quality solutions within a short amount of time. Despite the significant attention that local search has received in combinatorial optimization, its application to problems of quantitative temporal reasoning has as yet been largely overlooked.

In this paper, we show how local search can be successfully applied to overconstrained instances of the Disjunctive Temporal Problem (DTP) [Stergiou and Koubarakis, 1998], a particularly expressive form of temporal constraint satisfaction problem. While local search has been commonly applied to finite-domain CSPs, its application to DTPs is of particular interest, as the domains of the temporal variables are infinite, ranging over the entire set of either integers or real numbers. Previous work on DTP-solving has focused on a reformulation of the problem, in which a meta-CSP is constructed and searched. In contrast, we apply local search directly to the original CSP, i.e., the DTP itself. We discuss issues such as solution initialization, neighbor generation, and cost computation, and provide experimental results demonstrating that the use of local search in solving overconstrained DTPs can lead to substantially improved performance as compared to systematic methods.

It should be noted that ours is not the first attempt to apply local search to temporal reasoning. This work is closely related to [Beaumont *et al.*, 2004]; however, that line of research deals with a qualitative interval algebra representation, which is strictly less expressive than that allowed by DTPs. Nonetheless, their endpoint-ordering technique bears similarity to our approach, in that it also abandons the meta-CSP for the original CSP search space. Our work is also related to [Walser, 1998], where local search is applied to overconstrained integer programs without disjunctions.

2 Disjunctive Temporal Problems

A *Disjunctive Temporal Problem* (DTP) is a constraint satisfaction problem defined by a pair $\langle X, C \rangle$, where each element $X_i \in X$ designates a time point, and C is a set of constraints of the following form:

$$c_{i1} \vee c_{i2} \vee ... \vee c_{in}$$

where in turn, each c_{ij} is of the form $x - y \leq b$; $x, y \in X$ and $b \in \Re$. (In practice, b is often restricted to the integers.) DTPs are thus a generalization of Simple Temporal Problems (STPs), in which each constraint is limited to a single inequality [Dechter *et al.*, 1991]. A solution to a DTP is an assignment of values to time points such that all constraints are satisfied.

Several algorithms have been developed for solving DTPs [Stergiou and Koubarakis, 1998; Armando *et al.*, 1999;

Oddi and Cesta, 2000; Tsamardinos and Pollack, 2003]. Typically, these algorithms transform the problem into a *meta-CSP*, in which the original DTP is viewed as a collection of alternative STPs. Using this approach, the algorithm selects a single disjunct from each constraint of a given DTP. The resulting set forms an STP, called a *component STP*, which can then be checked for consistency in polynomial time using a shortest-path algorithm. Clearly, a DTP D is consistent if and only if it contains at least one consistent component STP. Furthermore, any solution to a consistent component STP of D is also a solution to D itself. Consequently, it is standard in the DTP literature to consider any consistent component STP to be a solution of the DTP of which it belongs.

A number of pruning techniques can be used to focus the search for a consistent component STP of a DTP, including conflict-directed backjumping, removal of subsumed variables, and semantic branching. The DTP solver Epilitis [Tsamardinos and Pollack, 2003] integrated all these techniques, in addition to no-good recording. At the time it was developed, Epilitis was the fastest existing DTP solver, though it was recently surpassed by TSAT++ [Armando *et al.*, 2004].

DTP solvers such as Epilitis perform total constraint satisfaction — that is, their objective is to find a solution that satisfies all the constraints of a DTP. In the event that a DTP is inconsistent, these solvers are capable of detecting such infeasibility, but are incapable of providing partial solutions that come close to satisfying the problem. In response, the DTP solver Maxilitis [Moffitt and Pollack, 2005] was designed to find solutions that maximize the number of satisfied constraints. Partial constraint satisfaction has the disadvantage of being expensive, as the pruning techniques typically used in DTP solving become weaker when relaxations are allowed. If the overconstrained DTP is very large, the systematic search that Maxilitis performs becomes intractable, and one must instead settle for an approximate solution. Fortunately, Maxilitis is an anytime algorithm, and one can interrupt it at any point to extract a (possibly suboptimal) partial assignment. It is not, however, obvious that a systematic search is the fastest way to find an approximate solution.

3 Local Search

When exact, systematic methods for solving hard combinatorial problems are too expensive, local search offers an alternative approach for quickly generating approximate solutions. Applications of local search are numerous, and include planning [Ambite and Knoblock, 1997], scheduling [Storer *et al.*, 1992], and constraint satisfaction [Minton *et al.*, 1992].

The Satisfiability Problem (SAT) is a classic domain for application of local search. The objective of SAT is to find an assignment to a set of binary variables that satisfies a Boolean formula F, typically given in Conjunctive Normal Form (CNF). A common variation on SAT is MAX-SAT, where the objective is to maximize the number of satisfied clauses in F. A number of local search algorithms have been constructed for both SAT and MAX-SAT. One of the most successful variants is GSAT [Selman *et al.*, 1992]. In this algorithm, one begins with a random assignment of truth values to the propositional variables, and then repeatedly chooses a variable to flip that results in the maximal decrease in violated clauses, breaking ties randomly. To avoid getting stuck in local minima, a technique called *random restarts* is often used, where the algorithm begins anew with another random assignment. Several variants exist for GSAT; of these, GSAT/Tabu [Mazure *et al.*, 1997] is among those with best performance. It maintains a *tabu list* of recently flipped variables, which are then temporarily prohibited from being selected for another flip.

The basic method used in GSAT and GSAT/Tabu can be applied to other problems besides SAT. Indeed, the structure of a DTP closely resembles that of a SAT problem, and it is for this reason that a SAT solving approach has been applied to the problem of finding complete solutions to DTPs (as in TSAT [Armando *et al.*, 1999] and TSAT++ [Armando *et al.*, 2004]).

3.1 Application to DTPs

In applying local search to possibly overconstrained DTPs, we again need to decide whether to work in the meta-CSP or in the original CSP (the DTP). To illustrate these candidates, consider the following very small problem:

$$C_1 : \{c_{11} : a - b \leq 10\}$$
$$C_2 : \{c_{21} : b - a \leq -15\} \vee \{c_{22} : c - a \leq -25\}$$
$$C_3 : \{c_{31} : b - c \leq 10\}$$
$$C_4 : \{c_{41} : a - c \leq 20\}$$

Clearly there is no complete solution to this problem, since c_{21} conflicts with c_{11}, and c_{22} conflicts with the constraint induced by the transitive composition of c_{11} and c_{31}. As a result, our objective will be to find a solution that maximizes the number of constraints satisfied in the DTP. We define the *cost* of a solution to be the number of constraint violations it induces; thus, low-cost solutions are better.[1]

Partial Assignment Space of the Meta-CSP

One way to view a solution to overconstrained DTPs such as the example above is as a *partial* assignment to the meta-CSP, in which some of the meta-level variables are left unassigned. For example, consider the assignment $(C_1, C_2, C_3, C_4) \leftarrow (c_{11}, c_{22}, \epsilon, \epsilon)$, in which the constraints C_3 and C_4 are not given an assignment (indicated by ϵ). Here the (partial) component STP that is selected consists of c_{11} and c_{22}, and has a cost of 2. This particular partial assignment cannot be extended to any solution of lower cost, since inclusion of either of the disjuncts c_{31} or c_{41} would result in an inconsistency. However, this does not mean that a solution of higher quality does not exist. Indeed, the partial assignment $(c_{11}, \epsilon, c_{31}, c_{41})$ with cost 1 is both consistent and optimal.

While search in the partial assignment space of the meta-CSP is common to most systematic methods for solving DTPs, it is less attractive for the application of local search. First, systematic methods work within a backtracking tree, where disjuncts are removed in the order in which they were

[1] Other cost functions – for instance, the maximum deviation from the bound of any constraint – are possible. However, there currently exist no systematic methods to compare against that minimize such alternative cost functions.

added.[2] One technique that is commonly used in DTP solving, *incremental full-path consistency* [Mohr and Henderson, 1986], exploits this property by maintaining a stack of the temporal network updates made during search, using it to cheaply repair path dependencies when backtracking. Local search requires the ability to modify arbitrary values in the partial assignment, not necessarily respecting the order in which they were originally assigned; thus it cannot exploit the incremental approach. Second, several of the powerful pruning techniques used by DTP solvers such as Epilitis and Maxilitis have no meaning outside the context of a systematic search tree. For example, semantic branching is able to acquire additional network constraints by exhaustively exploring particular assignments of disjuncts to constraints. In local search, no such exhaustive search is performed, and consequently it is hard to imagine how to adapt this mechanism. Finally, in local search, the number of neighbors for a partial assignment will typically be much larger than the number of successors in systematic search, thus making evaluation of alternatives prohibitively expensive.

Total Assignment Space of the Original CSP

An alternative approach is to perform search in the space of total assignments to the time points in the original CSP. For the example DTP above, one possible assignment is $(a, b, c) \leftarrow (30, 25, 0)$. Since this solution violates constraints C_3 and C_4 it has a cost of 2. A better solution would be $(20, 10, 0)$, which violates only constraint C_2.

A key issue for this search space is how to define the neighbors of an assignment. One approach is to include all solutions whose variable assignments differ in exactly one position; for example, $(a, b, c) \leftarrow (30, 15, 0)$ would be a neighbor of $(a, b, c) \leftarrow (30, 25, 0)$, since the two differ only on the assignment to b. However, because the value of each variable may take any real (or any integer), the cardinality of the set of possible neighbors is uncountably (or countably) infinite. It is for this reason that most of the prior work on DTPs has avoided search in this space: the infinite cardinality of the variables' domains precludes the use of methods for exhaustive search. We address this concern in the next section, where we describe our local search algorithm in detail.

4 Localitis

In this section, we present Localitis, our algorithm for using local search to perform partial constraint satisfaction in a DTP. Its basic framework derives from the GSAT/Tabu algorithm that is commonly used for SAT and MAX-SAT instances. It also bears considerable resemblance to WalkSAT(OIP), used in [Walser, 1998] to solve overconstrained integer programs without disjunctions[3].

[2] With chronological backtracking, the last disjunct added is the first one removed; with more sophisticated techniques such as conflict-directed backjumping, additional disjuncts may be removed at the same time, but the removal always follows the structure of the backtracking tree.

[3] While any disjunctive program can be cast as an integer program, the conversion process requires the creation of additional variables and constraints. Our method instead operates directly on the constraints in their disjunctive form.

4.1 Generating the Initial Solution

Typically in algorithms such as GSAT, the initial solution is chosen by selecting random assignments for each of the Boolean variables. A similar initialization can be done with DTPs: each time point can be randomly assigned a value within some interval $[L, U]$. If one employs random restarts, this initialization can be done several times to ensure adequate exploration of the search space.

A potentially better alternative is to make a greedy assignment to the time points, so that the algorithm begins with a reasonable solution. We adopt this approach in Localitis. To generate the initial solution, we make use of the Maxilitis solver. Recall that Maxilitis searches in the meta-CSP space. We invoke it and let it run until it generates its first solution, in which every constraint is assigned either a disjunct or ϵ. Since this solution is not necessarily optimal, Maxilitis would normally continue, but we instead terminate it, and project an assignment to time points from the component STP it has computed, making random assignments to all time points outside the scope of the component STP. Because of the variable and value ordering heuristics used by Maxilitis, the first solution it encounters is likely to be better than what a purely random assignment would provide. One could repeat this process for random restarts, placing bias on the selection of assignments so that the set of disjuncts chosen differs between runs.

4.2 The Neighborhood

As described earlier, the most obvious way to define the neighbor(s) of an assignment is as the set of assignments that differ in the value of only one time point (for example, $(a, b, c) \leftarrow (30, 25, 0)$ and $(a, b, c) \leftarrow (30, 15, 0)$). Unfortunately, this definition results in each assignment having infinitely many neighbors. The key to reducing the size of the search space is to note that if we hold fixed the values of all the variables but one (b in the current example), then only a small set of new values for the selected variable are significant. Specifically, we only need to consider those values for which the *slack* of a disjunct belonging to some constraint becomes zero — that is, when an inequality becomes a strict equality. This is somewhat similar to a pivot step in the simplex method for linear programming, which maintains a basic feasible solution that corresponds to an "active system" of constraints in the LP. For instance, in our running example, suppose we are given the assignment of $(30, 25, 0)$. Then the significant values for b are 20, 15, and 10, as they would make the disjuncts c_{11}, c_{21}, and c_{31} active, respectively. Some of these values may satisfy several new disjuncts at once (for instance, if b is assigned the value 10, it satisfies disjuncts c_{21} and c_{31} simultaneously). No other values for b are capable of satisfying a set of constraints that one of the these significant values cannot.

We can impose yet another restriction on the set of neighbors; namely, that they include only those new, significant values that change the resulting set of satisfied constraints. For instance, while the assignment of 20 to b is significant in that it makes the slack for c_{11} zero, it does not change the set of satisfied constraints. As a result, this new assignment is not particularly interesting, and need not be considered in the set of neighbors of this assignment.

4.3 Neighbor Selection and Tabu Moves

Since our objective is to satisfy the maximum number of constraints, selection of which neighbor to explore is a fairly straightforward process: choose the candidate that satisfies the maximum number of constraints possible. This is identical to GSAT, where the variable chosen to flip is the one that will minimize the number of unsatisfied clauses. In the case that several assignments would result in the same minimal number of violations, one is selected at random.

When a local search process is required to continually make greedy moves, it can easily get stuck in local minima. To avoid this, we adopt a common variation on the technique of tabu search [Glover and Laguna, 1997], and forbid local search to change the values of variables that were recently modified. A parameter tt, called the tabu tenure, determines the duration (in search steps) for which this restriction applies. When tabus are applied to SAT, no variable flipped at iteration number i is allowed to flip again until tt steps have passed. We introduce tabu moves into Localitis in a similar way. If the value of a variable x is changed at step i, its value is then fixed until tt steps have passed. To efficiently determine the tabu status of each variable x, we maintain an array it, where it_x stores the search step number when variable x was last changed. A variable is tabu if and only if $i - it_x < tt$.

4.4 Efficient Cost Computation

The simplest way to compute the cost of a neighbor assignment is to temporarily enforce the assignment being considered, and subsequently test all constraints for satisfiability, counting the number that are not satisfied. However, this straightforward implementation has been shown to be rather inefficient, since many of the constraints will not be affected by the local modification [Selman *et al.*, 1992].

A common technique is to instead compute the relative change in cost by testing only those clauses that contain the variable in question. This can be facilitated by a preprocessing step which creates a list for each variable, containing indices for those constraints that it participates in.

4.5 The Algorithm

Figure 1 provides the pseudocode for Localitis. The function Maxilitis-First-Path() is used to generate the initial assignment (line 1). The algorithm then performs local search, generating the neighbors of the current assignment by looking at the non-tabu variables in each disjunct of the DTP (lines 4-5), and considering "moves", i.e., assignments to significant values (line 6) that change the set of satisfied constraints (line 7). (The notation *X/Y*, where Y is an assignment $Z \leftarrow z$, denotes the substitution of Y for the original assignment to Z in X. The function Sat() returns the set of constraints in the DTP that are satisfied by its argument.) The set *moves* stores the set of minimal-cost neighbors (lines 8-13), from which one is eventually selected at random (lines 19-20). After storing the value of the best solution seen to date (lines 21-22) and updating tabu tenures (line 24), the process iterates.

Localitis(DTP *D*)

1. *Best-Assign* ← *Assign* ← Maxilitis-First-Path(*D*)
2. For *it* = 1 to *max-steps*
3. *min_cost* ← ∞, *moves* ← ∅
4. For each disjunct d: $x - y \leq b$
5. If x not tabu
6. *move* ← ($x \leftarrow y + b$)
7. If Sat(*Assign*) ≠ Sat(*Assign/move*)
8. If cost(*Assign/move*) = *min_cost*
9. *moves* ← *moves* ∪ {*move*}
10. EndIf
11. If cost(*Assign/move*) < *min_cost*
12. *moves* ← {*move*}
13. *min_cost* ← cost(*Assign/move*)
14. EndIf
15. EndIf
16. EndIf
17. Repeat lines 5 – 16 for y (i.e., $y \leftarrow x - b$)
18. EndFor
19. *new-move* ← Random-Member(*moves*)
20. *Assign* ← *Assign/new-move*
21. If cost(*Assign*) < cost(*Best-Assign*)
22. *Best-Assign* ← *Assign*
23. EndIf
24. Update tabu count for the time point in *new-move*
25. EndFor
26. return *Best-Assign*

Figure 1: Localitis, a local search algorithm for DTPs

5 Experimental Results

We implemented Localitis and conducted a set of experiments whose primary goals were 1) to determine the influence of the way in which the initial assignment is made, 2) to analyze the effect of various neighborhood functions and tabu tenures, and 3) to compare the anytime quality of this solver against its cousin Maxilitis, which performs an exhaustive search of the DTP in the meta-CSP space. No random restarts were used for these tests. To benchmark our algorithm, we used DTPs created by a random generator used in testing previous DTP solvers [Stergiou and Koubarakis, 1998]. The test case generator takes as arguments the parameters $\langle k, N, m, L \rangle$, where k is the number of disjuncts per constraint, N is the number of time points, m is the number of constraints, and L is the constraint width, i.e., a positive integer such that for each disjunct $x - y \leq b$, $b \in [-L, L]$ with uniform probability. In our experiments, we set $k = 2$, $N = 25$, $m = 175$, and $L = 100$. A derived parameter R (the ratio of constraints over variables, m/N) expressing constraint density was thus 7. For this set of parameters, 50 random problems were generated. The domains of the variables are integers instead of reals, which again is standard in DTP literature. Our implementation of Localitis was developed in Java, and our experiments were conducted on a 3 GHz Intel Pentium 4 machine running Windows XP and having 1 GB of memory.

In our first experiment, we tested 5 different values for the tabu tenure to measure its effect on solution quality as a function of time. In Figure 2, we plot a curve for each tabu tenure in the set $\{0, 1, 2, 3, 4\}$. The number of seconds elapsed is shown on the x-axis, and the number of constraint violations in the solution (averaged over the 50 test cases) is shown on

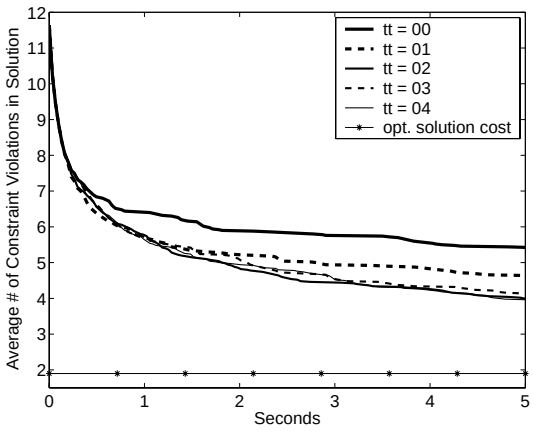

Figure 2: Anytime curves for various tabu tenures

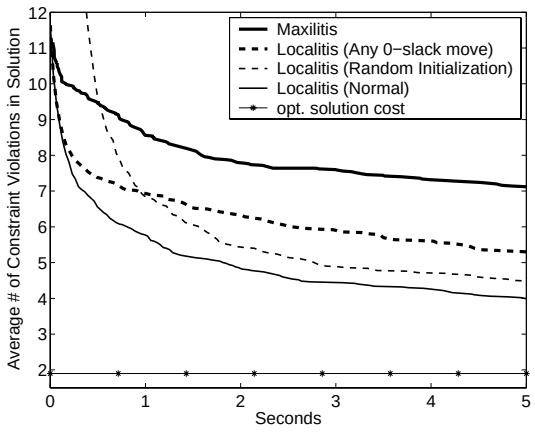

Figure 3: Anytime curves for Maxilitis & Localitis variations

the y-axis. We also show the optimal solution cost, which expresses the average number of violations in the optimal solution (i.e., what Maxilitis reported when run until completion). Convergence is slowest for when the tabu tenure is 0 (this effectively corresponds to the absence of tabu search), and is also somewhat slow for when the tabu tenure is 1. For values 2, 3, and 4, convergence is almost identical. Larger values (not shown) displayed no improvement over these curves, and so a tabu tenure of 2 was used for all subsequent tests.

In our second experiment, we studied two different strategies for generating the initial set of temporal values: one using random assignments, uniformly chosen from the range $[-L, L]$, and the other using Maxilitis to greedily generate an initial solution, as described previously. The results are shown in Figure 3 (among other results that will be addressed momentarily). Once again, the x- and y-axes represent the seconds elapsed and the solution cost, respectively. The curve labeled 'Localitis (Normal)' uses the greedy selection, and the curve labeled 'Localitis (Random Initialization)' does not. The former begins with an average cost of 11.64 violated constraints, where the latter begins with a drastically higher average of 43.60 not shown on the graph. The randomly initialized search is able to make up the difference fairly quickly, although it continues to lag for the duration of the search. The shape of the curves are roughly identical, despite the horizontal displacement. Thus, it appears that the greedy initialization does indeed improve the starting solution, although the effect on the convergence rate is negligible.

In our third experiment, we tested two different neighborhood criteria: the first includes all assignments of significant values (ones that force the slack of some disjunct to be zero), while the second also requires that there be some change in the set of satisfied constraints in the DTP. The results are shown in Figure 3. The curve labeled 'Localitis (Normal)' uses the more restricted definition, and the curve labeled 'Localitis (Any 0-Slack move)' does not. While the two begin at the same initial point (as they should, since both use greedy initialization), the 'Localitis (Normal)' curve is able to generate solutions of higher quality much earlier. For instance, after 5 seconds, 'Localitis (Any 0-Slack move)' has generated an average solution cost of 5.3. Only 1.28 seconds were required for 'Localitis (Normal)' to obtain this same average. As a result, it seems that one can expect significantly better results when the neighborhood is restricted to only those candidates which modify the partial assignment of the meta-CSP.

Finally, we compare the anytime quality of our Localitis solver to that of the systematic solver Maxilitis, which is (at present) the only other existing method for performing partial constraint satisfaction of DTPs. The results are again shown in Figure 3. The curves of interest are labeled 'Localitis (Normal)' and 'Maxilitis.' By inspection, it seems that despite its numerous pruning techniques and sophisticated constraint propagation methods, Maxilitis is far slower at producing high quality solutions. For instance, at the end of the 5 seconds shown, Maxilitis has generated an average solution cost of about 7.12. For Localitis to produce the same average cost requires only 0.29 seconds. To reach the average cost of 4 that Localitis achieves after 5 seconds requires 104 seconds of Maxilitis runtime. Based on these observations, the speedup achieved appears to be roughly 17 to 20 times faster.

6 Discussion and Future Work

In this paper, we have presented a method for applying local search to overconstrained instances of the Disjunctive Temporal Problem. In contrast to previous algorithms for solving DTPs, our technique abandons the meta-CSP and instead explores the total assignment space of the underlying CSP. Our results show that the computation time required to generate high-quality solutions is significantly reduced in comparison to traditional branch-and-bound algorithms for performing partial constraint satisfaction.

Given the freedom allowed by this alternative search space, one particularly interesting avenue of research would be to extend this approach toward more expressive cost functions. Whereas this paper concentrates on minimizing the number of violated constraints, one could instead capture the amount by which the constraints are violated. This would give higher value to those solutions whose assignments come close to falling within the prescribed bounds.

Another appealing possibility is to apply this same local search technique to underconstrained rather than overconstrained temporal formalisms. The recent addition of preferences to DTPs, appropriately labeled DTPs with Preferences (DTPPs) [Peintner and Pollack, 2004], allows preference functions to be defined over particular values of the temporal differences. As no efficient optimal algorithm is yet known for maximizing the weighted sum of preferences in DTPPs, greedy methods are currently being developed to generate approximate solutions. Local search may indeed prove to be a competitive alternative to these algorithms.

Acknowledgments

The authors thank Neil Yorke-Smith, Dushyant Sharma, and Bart Peintner for their input into this work. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010 and the Air Force Office of Scientific Research under Contract No. FA9550-04-1-0043. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA, the Department of Interior-National Business Center, or the United States Air Force.

References

[Ambite and Knoblock, 1997] Jose Luis Ambite and Craig A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 706–713, 1997.

[Armando *et al.*, 1999] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In *Proceedings of the 5th European Conference on Planning*, pages 97–108, 1999.

[Armando *et al.*, 2004] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. A SAT-based decision procedure for the boolean combination of difference constraints. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-2004)*, 2004.

[Beaumont *et al.*, 2004] Matthew Beaumont, John Thornton, Abdul Sattar, and Michael Maher. Solving overconstrained temporal reasoning problems using local search. In *Proceedings of the 8th Pacific Rim Conference on Artificial Intelligence (PRICAI-2004)*, 2004.

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[Glover and Laguna, 1997] Fred Glover and Manuel Laguna. *Tabu Search*. Kulwer Academic Publishers, Boston, MA, USA, 1997.

[Hoos and Stutzle, 2004] Holger Hoos and Thomas Stutzle. *Stochastic Local Search: Foundations and Applications*. Kluwer Academic Publishers, 2004.

[Khatib *et al.*, 2003] Lina Khatib, Paul Morris, Robert Morris, and K. Brent Venable. Tractable Pareto optimal optimization of temporal preferences. In *Proceedings of the 18th International Joint Conference on Artificial Intelligece (IJCAI-2003)*, pages 1289–1294, 2003.

[Mazure *et al.*, 1997] Bertrand Mazure, Lakhdar Sais, and Eric Gregoire. TWSAT: A new local search algorithm for SAT - performance and analysis. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 281–285, 1997.

[Minton *et al.*, 1992] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[Moffitt and Pollack, 2005] Michael D. Moffitt and Martha E. Pollack. Partial constraint satisfaction of disjunctive temporal problems. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, 2005.

[Mohr and Henderson, 1986] Roger Mohr and Thomas C. Henderson. Arc-consistency and path-consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Morris *et al.*, 2004] Paul Morris, Robert Morris, Lina Khatib, Sailesh Ramakrishnan, and A. Bachmann. Strategies for global optimization of temporal preferences. In *Proceedings of the 10th International Conference on Principles and Practices of Constraint Programming*, pages 408–422, 2004.

[Oddi and Cesta, 2000] Angelo Oddi and Amedeo Cesta. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 108–112, 2000.

[Peintner and Pollack, 2004] Bart Peintner and Martha E. Pollack. Low-cost addition of preferences to DTPs and TCSPs. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 723–728, 2004.

[Selman *et al.*, 1992] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

[Stergiou and Koubarakis, 1998] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 248–253, 1998.

[Storer *et al.*, 1992] Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. In *Management Science*, volume 38, pages 1495–1509, 1992.

[Tsamardinos and Pollack, 2003] Ioannis Tsamardinos and Martha E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1-2):43–90, 2003.

[Walser, 1998] Joachim P. Walser. *Domain-independent Local Search for Linear Integer Optimization*. PhD thesis, Universitat des Saarlandes, 1998.