Partial Constraint Satisfaction of Disjunctive Temporal Problems

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

Abstract

We present a method for finding optimal partial solutions to overconstrained instances of the Disjunctive Temporal Problems (DTP). The solutions are optimal in that they satisfy a maximal number of constraints. While partial constraint satisfaction (PCS) has been commonly applied to finite-domain CSPs, its application in this setting is of particular interest, as temporal problems are traditionally solved using a *meta-CSP* approach, in which the constraints of the original problem become the variables of the meta-level problem. We show how to adopt nearly all previous pruning techniques in DTP solving for use with PCS, and provide results demonstrating their effectiveness. We also introduce an incremental technique that leads to substantially improved performance.

Introduction

Temporal reasoning is an important tool in many areas of artificial intelligence, including planning, scheduling, and natural-language processing. A great deal of work has been done on temporal reasoning as a form of constraint satisfaction processing. Research on this topic has focused primarily on two key questions: how to expressively represent temporal information, and how to efficiently generate solutions for temporal problems. The former issue has been addressed with a number of different formalisms, where the underlying constraints have been augmented with conditional dependencies (Tsamardinos, Vidal, & Pollack 2003), uncertainties (Vidal & Fargier 1999), and preferences (Peintner & Pollack 2004). The latter issue has also received significant attention in the development of efficient algorithms for finding complete solutions to temporal reasoning problems (Stergiou & Koubarakis 1998; Tsamardinos & Pollack 2003; Choueiry & Xu 2004). However, most of the prior work has focused on finding complete solutions, while in practice, real-world temporal problems–and even synthetic ones–are often overconstrained. To be useful, a temporal constraint problem-solver should do more than generate a notice of failure when confronted with such a problem. What is really needed is a partial solution, i.e., a solution that satisfies a modified form of the original problem in which some of the constraints or variables have been weakened or removed.

Partial constraint satisfaction (PCS) (Freuder & Wallace 1992) techniques have been developed for finding partial solutions to overconstrained problems, by selecting a subset of constraints to relax. Often, this is done with the objective being to minimize the total number of weakened constraints.

In this paper, we show how partial constraint satisfaction can be successfully applied to Disjunctive Temporal Problems (DTPs) (Stergiou & Koubarakis 1998), a particularly expressive form of temporal constraint satisfaction problem that subsumes Simple Temporal Problems and Temporal Constraint Satisfaction Problems (Dechter, Meiri, & Pearl 1991). While partial constraint satisfaction has been commonly applied to finite-domain CSPs, its application in this setting is of particular interest, as temporal problems are traditionally solved using a *meta-CSP* approach, in which the constraints of the original problem become the variables of the meta-level problem. We show how to adopt nearly all of the pruning techniques for *meta-CSPs* implemented in the Epilitis DTP solver (Tsamardinos & Pollack 2003) so that they can be used with PCS. Some of these techniques have no previous exposure to partial constraint satisfaction, and as we will show, can prove to be quite useful in improving performance. We also introduce an incremental technique that leads to substantially improved performance.

It should be noted that ours is not the first attempt to combine partial constraint satisfaction and temporal reasoning. This work is closely related to (Beaumont *et al.* 2001); however, that paper deals an interval algebra representation, which is strictly less expressive than that allowed by DTPs.

Disjunctive Temporal Problems

A *Disjunctive Temporal Problem* (DTP) (Stergiou & Koubarakis 1998) 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 $b \in \Re$. DTPs are thus a generalization of Simple Temporal Problems (STPs), in which each constraint is limited to a single inequality. A solution to a DTP is an assignment of values to time points such that all constraints are satisfied. Notice that since each constraint C_i may involve more than

two temporal variables, it may not necessarily be a binary constraint.

Several algorithms have been developed for solving DTPs (Stergiou & Koubarakis 1998; Armando, Castellini, & Giunchiglia 1999; Oddi & Cesta 2000; Tsamardinos & Pollack 2003). Typically, these algorithms view the DTP as a collection of alternative STPs. Using this approach, the algorithm selects a single disjunct from each constraint of a given DTP D. The resulting set forms an STP, called a component STP of D, which can then be checked for consistency in polynomial-time using a shortest-path algorithm (Dechter, Meiri, & Pearl 1991). Specifically, an STP is consistent if and only if it contains no negative cycles, which can be determined by computing the all-pairs shortest path matrix and checking that the values along the main diagonal are non-negative. 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 given DTP. These include conflict-directed backjumping, removal of subsumed variables, and semantic branching. The DTP solver Epilitis (Tsamardinos & Pollack 2003) integrated all of these techniques, in addition to no-good recording. At the time it was developed, Epilitis was the fastest existing DTP solver, although it was recently surpassed by TSAT++ (Armando *et al.* 2004). We use Epilitis as the basis of our approach to partial constraint satisfaction in DTPs.

Partial Constraint Satisfaction

Partial Constraint Satisfaction (Freuder & Wallace 1992) deals with the issue of solving a constraint satisfaction problem by removing or weakening some of its constraints. Such a procedure is useful for cases in which the problem is over-constrained and thus lacks a complete solution, or is simply too difficult to solve completely and an approximate solution is acceptable. In *maximal constraint satisfaction*, the objective is to find a solution that satisfies the maximal number of constraints. Freuder and Wallace present an approach to maximal constraint satisfaction based on a natural backtracking search technique using branch and bound–whenever a solution is found that violates fewer constraints than previous solutions, the algorithm remembers it and afterwards prunes all search paths that cannot result in any better solution. They also demonstrate how basic pruning techniques such as forward checking and conflict-directed backjumping can be used to further reduce the search space during maximal constraint satisfaction.

Several subsequent enhancements have been made to the basic partial constraint satisfaction algorithm, most of which involve variations of arc consistency. Unfortunately, these enhancements are not especially useful for DTP solving: as discussed in (Choueiry & Xu 2004), performing generalized arc consistency on the meta-CSP is NP-hard and thus prohibitively expensive.

Application to DTPs

As illustration of the application of partial-constraint satisfaction to DTPs, consider the following very small problem:

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

Clearly there is no complete solution to this problem, since c_{21} conflicts with c_{11} (the only possible disjunct to be selected from C_1), while c_{22} conflicts with c_{31} (the only possible disjunct to be selected from C_3).

In this paper, our objective will be to find an assignment of values to time points that maximizes the number of constraints satisfied in the DTP, or, equivalently, that requires relaxation of the minimal number of constraints. For instance, the assignment $a \leftarrow 10, b \leftarrow 0, c \leftarrow 10, d \leftarrow 0$ which satisfies two constraints (C_1 and C_3), is optimal, in that no assignment can satisfy all three. Of course, there are many alternative assignments that also satisfy two constraints, and indeed, in this particular example, it is possible to satisfy any two of the three.

To achieve the goal of maximal constraint satisfaction in a DTP, we modify the original meta-level DTP solving algorithm so that instead of searching through the space of all possible component STPs, it effectively performs a search through all possible *partial* component STPs, where a partial component STP is one that may select STP constraints from only a proper subset of the meta-level constraints. Thus, with our current example, the algorithm might select c_{11} from C_1 and c_{31} from C_3, but leave C_2 unassigned–or, equivalently, make a null assignment to it. Of course, since the algorithm is designed to make the maximal number of assignments possible, it will still find a solution that satisfies all the constraints if the DTP is consistent. Note that even though one or more of the meta-level variables may be unassigned, all the variables of the original DTP, i.e., the time points, do receive assignments.

The meta-CSP approach taken in temporal constraint satisfaction processing differs somewhat from the approach taken when PCS is applied to traditional, finite-domain CSPs. In the latter case, the solution is a total assignment, where each constrained variable is assigned a value from its original domain; what may make the solution partial is the fact that those assignments may violate some of the original constraints. When we move to DTPs, a PCS solution is no longer necessarily a total assignment; instead a (meta)-variable may be left unassigned, signifying that none of the disjuncts associated with it should be enforced. In other words, the weakening of constraints in the original CSP maps directly to the removal of variables in the meta-CSP.

Figure 1 provides a simple partial constraint satisfaction algorithm for DTPs. The input variable A is the current set of assignments to meta-variables, and is initially $\oslash$; variable U is the set of unassigned variables (initially the entire set C); *cost* is the total number of meta-constraints given a null assignment (initially zero); and *upperbound* is the stored cost of the best solution found so far (initially set to the number of meta-constraints). The algorithm resembles a bare-bones backtracking algorithm that could be used for

```
Partially-Solve-DTP(A, U, cost, upperbound)
1. If (cost ≥ upperbound) return
2. If (U = ∅)
3.       best-solution-so-far ← A
4.       upperbound ← cost
5.       return
6. EndIf
7. C ← select-variable(U), U' ← U – {C}
8. For each value c of d(C)
9.       A' ← A ∪ {C ← c}
10.      If consistent(A')
11.            Partially-Solve-DTP(A', U', cost, upperbound)
12.      EndIf
13. EndFor
14. A' ← A ∪ {C ← ϵ}
15. Partially-Solve-DTP(A', U', cost + 1, upperbound)
```

Figure 1: A simple partial constraint satisfaction algorithm for DTPs

total constraint satisfaction in DTPs, with two notable differences. First, backtracking occurs only when the number of violated constraints (*cost*) equals or exceeds that of the current best solution (*upperbound*); in total constraint satisfaction, backtracking would occur whenever *cost* became nonzero. Second, when alternative assignments are made to meta-variables, there is the possibility of a null assignment (designated by ϵ in line 14), and thus the branching factor is one greater than that for total constraint satisfaction. Since the algorithm maintains a copy of the best solution found so far, this solution can be extracted at any time, thus making this an *anytime* algorithm.

Maxilitis

In this section, we present Maxilitis, our algorithm for maximal partial constraint satisfaction in a DTP. Its basic framework derives from the simple branch and bound algorithm given in Figure 1. On top of this framework, we add each of the additional pruning techniques adopted by Epilitis (Tsamardinos & Pollack 2003), with the exception of nogood recording, whose general applicability to partial constraint satisfaction has yet to receive significant attention. We will discuss each of these techniques, focusing on the changes that must be made to enable them to work with PCS. Certain other details, such as variable and value ordering heuristics, are omitted due to space limitations.

Forward Checking

Forward checking is one of the most basic mechanisms for dead-end detection and pruning in CSPs. It works by checking the domains of all unassigned variables against the current temporal network, removing any values discovered to be inconsistent. As is the case with most CSPs, forward checking is an essential component in any DTP solving algorithm. If the all-pairs shortest paths matrix is maintained and updated using incremental full path consistency, forward checking can be performed in $O(v+|X|^2)$ time, where X is the set of all of time points, and v is the number of remaining legal values.

With total constraint satisfaction, a failure is encountered whenever the domain of any variable is reduced to $\oslash$. However, when performing partial constraint satisfaction, one can no longer backtrack whenever the entire domain of a variable is exhausted. For example, it may be the case that the maximum solution includes exactly one unsatisfied constraint, that being the one with no remaining consistent values. As a result, one must increment *cost* whenever the domain of such a variable is obliterated, and backtrack only when this cost is no better than the current upper bound. This modification to forward-checking for Maxilitis directly parallels that presented in (Freuder & Wallace 1992).

Conflict-Directed Backjumping

Conflict-directed backjumping is a commonly used retrospective technique that skips over unrelated assignments when backtracking up the search tree. It is guaranteed never to prune away any potential solutions, since no alternative assignment to any of the unrelated variables would correct the inconsistency encountered. Most backjumping schemes use dependency pointers to label domain values removed during forward-checking with the assignments that precluded them. The variation of backjumping that Epilitis uses is slightly different, as it instead infers whether a constraint was involved in a failure by examining the time points that lie along the induced negative cycle; if it does not, then it is irrelevant to the current failure and can be skipped over.

In modifying Epilitis' model of backjumping for PCS, we again follow the approach of (Freuder & Wallace 1992). That is, instead of backjumping to the deepest level of failure, we must stop at any level that required an increase to *cost* (that is, whenever either a constraint was given a null assignment, or forward checking removed all disjuncts of another constraint due to negative cycle detection).

Removal of Subsumed Variables

A third key pruning technique used in DTP solving is the removal of subsumed variables. The basic idea here is that some unassigned constraints may already be satisfied by the current temporal network. In this case, one can safely refrain from making separate, additional attempts to satisfy them. For example, suppose that the disjuncts $x - y \leq 5$ and $y - z \leq 5$ have been assigned to their respective constraints, and one is about to consider satisfying another constraint C_i that includes the disjunct $x - z \leq 15$. The first two constraints imply that $x-z \leq 10$, and hence, of course $x-z \leq 15$. Thus C_i is necessarily satisfied, and no additional effort should be extended trying alternative disjuncts in it.

This technique becomes even more valuable when partial constraint satisfaction is invoked. Recall that when solving for the maximum number of constraints, the branching factor increases by one, since the search space must include the case where a constraint is left unsatisfied. But when a constraint has already been satisfied by previous assignments, the attempt to leave it unassigned is clearly needless. It should be noted that the implementation of removal of subsumed variables requires no special modifications when moving from total to partial constraint satisfaction.

Semantic Branching

One of the most powerful pruning methods used in solving DTPs is semantic branching (Armando, Castellini, & Giunchiglia 1999), a technique that exploits the fact that the constraints represent numeric inequalities. Suppose that the partial assignment $A \cup \{C \leftarrow c\}$ is explored in every possible way, yet leads to no solution. In such a case, if a solution exists that is an extension of A, then $\{C \leftarrow c\}$ cannot be a feasible assignment (or else its branch of search would have succeeded). Consequently, it must be the case that the negation of this constraint must hold in any solution that is an extension of A (if one exists). But remember that constraint c will have the form $x - y \leq b$. Thus, in every extension of A that is a solution to the DTP, it must be the case that $y - x < -b$. Semantic branching makes this requirement explicit, i.e., it adds a new constraint representing this inequality to the network, and thereby in general significantly prunes the subsequent search.

As in the case of removal of subsumed variables, partial constraint satisfaction has the potential to benefit significantly from semantic branching. Consider the case when all disjuncts of a constraint have been attempted. At this point, the algorithm will leave the constraint unsatisfied to see if better solutions can be found. With semantic branching enabled, the algorithm does more than simply ignore the constraint; it also ensures that in the remainder of the search the conjunction of all the negated disjuncts will be satisfied. In this way, it makes use of the fact that the constraint in question is intended to be violated, and (with the added increase to *cost*) can potentially do substantial pruning of the remaining search space.

As with removal of subsumed variables, semantic branching can be directly incorporated into a PCS approach to DTP-solving; no changes are needed to move from total to partial constraint satisfaction.

Experimental Results

In this section, we describe the results of a set of experiments that were performed on Maxilitis to test the effectiveness of PCS when applied to the meta-CSP approach of DTP-solving. The primary goals of these experiments were to 1) assess the performance penalty incurred when moving from total to partial constraint satisfaction, 2) determine the effect of the various pruning techniques, and 3) evaluate the anytime quality of the solver. To benchmark our algorithm, we used DTPs created by a random generator used in testing previous DTP solvers (Stergiou & 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 $k = 2$, $L = 100$, and $N = 20$. A derived parameter R (the ratio of constraints over variables, m/N) was varied from 2 to 7. For each setting of R, 50 random problems were generated. (For example, we generated 50 problems for the case where N is 20 and R is 7; those problems have 20 variables and 140 constraints each). The percentages of inconsistent problems for $R = (2, 3, 4, 5, 6, 7)$ were found to be $(0\%, 0\%, 0\%, 12\%, 72\%, 94\%)$, respectively. The domains of the variables are integers instead of reals, which again is standard in DTP literature. Unlike Epilitis, which was implemented in Lisp, our implementation of Maxilitis was developed in Java, and our experiments were conducted on a Sun Blade 1500 with a 1 GHz processor and 1 GB of memory.

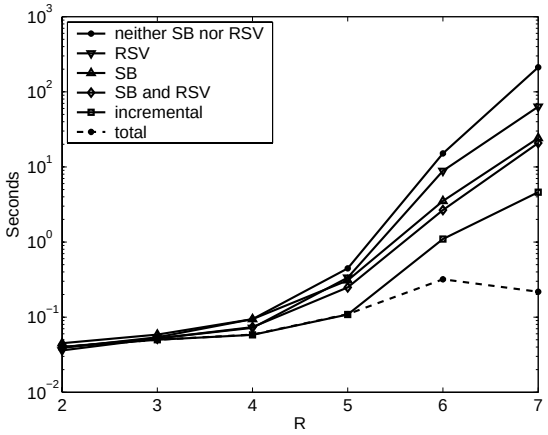

Figure 2: Median computation time for several variations of Maxilitis, averaged over 50 trials

In the first experiment, we varied the pruning strategies that were "turned on". Figure 2 shows the solution time in seconds as a function of the R value, which has been shown in the previous literature to be the critical variable for DTP-solving. Note that the y-axis is logarithmic. We tested conditions in which we used both Removal of Subsumed Variables (RSV) and Semantic Branching (SB), one or the other, or neither. (In all cases, we used forward-checking and conflict-directed backjumping, since these are fairly standard techniques for constraint solving.) We also tested a version of Maxilitis in which we set $upperbound$ to 1 and thereby forced the system to find a complete solution, i.e., one in which every constraint is satisfied; this is identified as "total" in the figure (in the case of failure, we report the time it takes for search to complete). The final condition is one called "incremental," which we discuss in a moment.

As can be seen in the figure, time to perform complete satisfaction peaks when $R = 6$ and then begins to decrease. This is consistent with previous literature, which has shown values of R near 6 to define the critical region for DTP-solving (Tsamardinos & Pollack 2003). Unfortunately, the time to find a solution for the PCS approach continues to rise for $R = 7$. This is because in partial constraint satisfaction, the computational benefits of overconstrained problems no longer apply: search cannot be aborted whenever any constraint becomes unsatisfiable. Instead, the algorithm drives on into much deeper search nodes, looking for solutions that may have several unsatisfied constraints. Consequently, one can expect these plots to rise indefinitely as R is increased, a

rather disappointing result, as it is these extremely overconstrained problems that often need partial satisfaction.

Figure 2 also provides a basis for comparing the effectiveness of RSV and SB for partial constraint satisfaction. As we would expect, the worst performance is observed when neither pruning technique is employed. Results improve when using RSV only; more so with SB only; and the best performance is obtained with the combination of techniques. From these results, one can conclude that semantic branching is the more powerful of the two techniques in PCS; this is consistent with observations made in (Tsamardinos & Pollack 2003) for complete constraint solving in DTPs. Furthermore, in tests where R is equal to 7, the algorithm experiences a full order of magnitude speedup when both pruning techniques are turned on relative to having them both off.

By far the fastest technique for PCS used, however, is the "incremental" algorithm, pseudo-code for which is given in Figure 3. Basically, the incremental algorithm starts by performing total constraint satisfaction on the DTP, using all the pruning techniques discussed in this paper. If that fails, it then performs another search, this time looking for a solution that violates exactly one constraint. If that then fails, it continues to iterate, each time performing search after incrementing the upper bound on the solution cost. Once a solution is discovered, the search can end immediately, as previous failed iterations imply that no better solution exists. Our incremental approach is closely related to Limited Discrepancy Search (Harvey & Ginsberg 1995) as well as iterative deepening, except that we increase the allowable number of constraint violations within the tree rather than the depth of the tree itself. This strategy is especially useful in the event that the DTP is consistent, since the first call to **Partially-Solve-DTP()** will return successfully. (It is for this reason that the incremental strategy requires the same amount of time as does the total constraint satisfaction algorithm up through R = 5, since most of these test cases are satisfiable). For even more constrained problems, where the expected cost may be much larger, one could possibly increment the upper bound by larger amounts after each iteration.

The success of the incremental algorithm is obviously dependent on the number of iterations it takes before a solution is found. For this reason, we calculated the distribution of solution costs–that is, the number of constraints violated in the optimal solution–for the case where when R = 7. For *cost* = (0, 1, 2, 3, 4, 5), the relative proportions were found to be (6%, 36%, 38%, 12%, 6%, 2%). Thus, although only a very small percentage of these problems have complete solutions, roughly 80% can be satisfied with 2 or fewer constraints removed, and only one case out of all 50 randomly generated problems required as many as 5 unsatisfied constraints. This is the primary reason why our incremental method proved to be so successful: in most cases, the upper bound only had to be increased once or twice until the minimal cost was discovered.

Incrementally-Solve-DTP(C)
1. *upperbound* ← 1
2. *best-solution-so-far* ← null
3. Do Until (*best-solution-so-far* ≠ null)
4. **Partially-Solve-DTP**($\oslash$, C, $|C|$, *upperbound*)
5. *upperbound* ← *upperbound* + 1

Figure 3: An incremental partial constraint satisfaction algorithm for DTPs

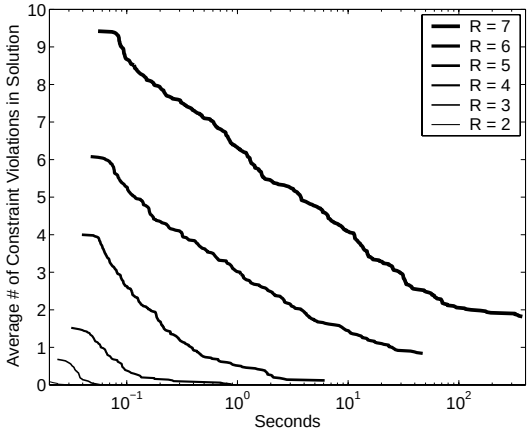

Figure 4: Anytime curves for various values of R

The one downside to the incremental algorithm is that it lacks the anytime property: one cannot interrupt the algorithm in the middle and ask for the best (possibly suboptimal) solution found thus far. Conseqently, one might be curious about the nature of anytime performance in the original algorithm. In Figure 4, we have made an effort to measure the anytime quality of our partial constraint satisfaction algorithm. For this experiment, we separately plotted the solution time of the algorithm with RSV and SB both turned on, for each value of R between 2 and 7. The number of seconds elapsed is shown on the x-axis, in a logarithmic scale; on the y-axis is the number of constraint violations in the solution, averaged over the 50 test cases for each value of R. Naturally, it appears from these curves that since higher values of R create problems that are more constrained, the average number of violations increases with R. Also, the number of violations obviously decreases over time–otherwise, there would be something very wrong with the algorithm. The most notable phenomenon to observe is the existence of a relatively constant deceleration in the number of constraints violated–that is, each order of magnitude of computation time appears to decrease the number of violated constraints by roughly the same amount. For example, consider the curve for which R = 7. At times 10^{-1}, 10^{0}, 10^{1}, and 10^{2} seconds, the average number of constraints violated are 8.67, 6.32, 4.09, and 2.05, respectively: a fairly steady decrease of just over 2 constraints per order of magnitude in computation time. Thus, the longer one waits, the slower the rate of improvement to the partial solution. Fortunately, this also means that the biggest improvements are made early on in the search rather than later.

Discussion and Future Work

In this paper, we have presented a method for dealing with overconstrained instances of the Disjunctive Temporal Problem (DTP). Specifically, the technique of partial constraint satisfaction was applied to obtain solutions that maximize the number of satisfied constraints. Our results have shown that the computation time required to partially solve DTP problems is dramatically reduced by applying the pruning techniques of removal of subsumed variables and semantic branching. Furthermore, we have demonstrated that by executing our solver multiple times with incrementally increasing upper bounds on the solution cost, the performance can be improved even more, at the cost of giving up the algorithm's anytime quality. Unfortunately, despite these speedups, exact partial constraint satisfaction appears to become intractable when the number of constraints or variables in the problem becomes substantially large.

One possible extension to this work is to give up altogether on an exact algorithm, and instead construct an approximate algorithm using local search. This is the approach taken in (Beaumont *et al*. 2004) for problems expressible in the interval algebra, and similarly in (Mouhoub 2001). The local search procedure may be performed in the partial assignment space of the meta-CSP, or alternatively in the total assignment space of the underlying CSP. In either case, we suspect that by generating an initial assignment to the DTP and repeatedly performing heuristic repair, it may be possible to generate high-quality partial solutions in much less time than with the use of systematic methods.

Another interesting line of research would be to extend this framework to allow for more expressive mechanisms for choosing which constraints to relax. As an example, one may want to attach weights to the constraints, and then attempt to maximize the weighted sum of the satisfied constraints. Luckily, our branch and bound algorithm can easily be modified to achieve this slightly different objective. We are currently investigating ways to combine Maxilitis with preference elicitation schemes that will provide the ability to learn such weights via interactions with a user.

One final possible avenue of research is to combine partial constraint satisfaction with the recent addition of preferences to DTPs (Peintner & Pollack 2004). As a matter of fact, the current implementation of DTPPs (Disjunctive Temporal Problems with Preferences) uses much of Maxilitis in its foundation, although at present it is only used in a total constraint satisfaction mode.

Acknowledgements

The authors thank Bart Peintner for his valuable 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

Armando, A.; Castellini, C.; Giunchiglia, E.; and Maratea, M. 2004. 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)*.

Armando, A.; Castellini, C.; and Giunchiglia, E. 1999. SAT-based procedures for temporal reasoning. In *Proceedings of the 5th European Conference on Planning (ECP-1999)*, 97–108.

Beaumont, M.; Sattar, A.; Maher, M.; and Thornton, J. 2001. Solving overconstrained temporal reasoning problems. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI-2001)*, 37–49.

Beaumont, M.; Thornton, J.; Sattar, A.; and Maher, M. 2004. Solving over-constrained temporal reasoning problems using local search. In *Proceedings of the 8th Pacific Rim Conference on Artificial Intelligence (PRICAI-2004)*.

Choueiry, B. Y., and Xu, L. 2004. An efficient consistency algorithm for the temporal constraint satisfaction problem. *AI Communications* 17(4):213–221.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58(1-3):21–70.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 607–615.

Mouhoub, M. 2001. Analysis of approximation algorithms for maximal temporal constraint satisfaction problems. In *The 2001 International Conference on Artificial Intelligence (ICAI-2001)*, 165–171.

Oddi, A., and Cesta, A. 2000. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, 108–112.

Peintner, B., and Pollack, M. E. 2004. Low-cost addition of preferences to DTPs and TCSPs. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004)*, 723–728.

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

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

Tsamardinos, I.; Vidal, T.; and Pollack, M. E. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints* 8(4):365–388.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: From consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11(1):23–45.