Low-cost Addition of Preferences to DTPs and TCSPs

Bart Peintner and Martha E. Pollack
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109 USA
{bpeintne, pollackm}@eecs.umich.edu

Abstract

We present an efficient approach to adding soft constraints, in the form of preferences, to Disjunctive Temporal Problems (DTPs) and their subclass Temporal Constraint Satisfaction Problems (TCSPs). Specifically, we describe an algorithm for checking the consistency of and finding optimal solutions to such problems. The algorithm borrows concepts from previous algorithms for solving TCSPs and Simple Temporal Problems with Preferences (STPPs), in both cases using techniques for projecting and solving component sub-problems. We show that adding preferences to DTPs and TCSPs requires only slightly more time than corresponding algorithms for TCSPs and DTPs without preferences. Thus, for problems where DTPs and TCSPs make sense, adding preferences provides a substantial gain in expressiveness for a marginal cost.

Introduction

Over the last decade, efficient algorithms have been developed for reasoning with a variety of temporal networks, which range in complexity from Simple Temporal Problems (STPs) and Temporal Constraint Satisfaction Problems (TCSPs) (Dechter, Meiri, & Pearl 1991) to Disjunctive Temporal Problems (DTPs) (Stergiou & Koubarakis 2000). Attention has been paid to checking networks for consistency, finding solutions, and correctly dispatching the steps in plans described by such networks. Only recently, though, has effort been made to increase the expressiveness of temporal networks. The two major extensions include uncertainty, i.e., the modeling of stochastic or uncontrollable events (Morris, Muscettola, & Vidal 2001; Tsamardinos, Pollack, & Ramakrishnan 2003), and preferences, i.e., the representation of soft constraints in addition to the hard constraints typical in temporal networks (Khatib *et al.* 2001; 2003). Where previous work has addressed the addition of preferences to STPs, in this paper we focus on adding preferences to DTPs and their subclass TCSPs.

Hard constraints in temporal networks specify the allowable temporal distances between time points in a network. For example, $X_2 - X_1 \in [5, 10]$ specifies that time point X_2 must occur 5-10 units after X_1. A temporal distance outside the interval violates the constraint. Adding preferences to a temporal network involves assigning to each constraint a preference function that maps each temporal distance value to a preference value. Preferences allow a knowledge engineer to designate some temporal differences as more desirable than others. With preferences, the problem is changed from simply finding a consistent assignment for time points to finding an optimal assignment.

The latest work on preferences in temporal networks provides a polynomial-time algorithm for checking the consistency of and finding an optimal assignment for STPs with semi-convex preference functions[1] (Khatib *et al.* 2003). The notion of optimality used in that work is called Weakest Link Optimality (WLO). With WLO, the preference value of an entire solution is set to the minimal (worst) preference value for any part of the solution, and hence the optimal solution is the one that makes the best worst assignment; i.e., it is a maximin decision criterion (Wald 1950). In this paper, we adopt the WLO approach, and present an algorithm for checking the consistency of and finding a WLO solution for DTPs with Preferences (DTPPs) and TCSPs with Preferences (TCSPPs). The algorithm borrows concepts from previous algorithms for solving TCSPs and STPs with Preferences (STPPs), in both cases using techniques for projecting and solving component sub-problems. We demonstrate that the added expressivity provided by preferences is computationally inexpensive. Specifically, we show that our algorithm not only falls into the same complexity class as the corresponding hard-constraint algorithms, but that the added cost of handling such preferences is worst case polynomial in the number of time points in the network.

After a brief review of temporal networks, we motivate the need for preferences with an example. We then present a modified version of an algorithm for solving standard TCSPs, and extend it to one that handles both TCSPPs and DTPPs. Finally, we analyze the algorithm's complexity and present future work.

Background

A Simple Temporal Problem (STP) is a pair $< X, C >$, where the elements $X_i \in X$ designate time points, and C is a set of binary constraints of the following form:

$$X_j - X_i \in [a_{ij}, b_{ij}].$$

---

[1]Their definition of a semi-convex function is one in which, for all Y, the set $\{X : f(X) \geq Y\}$ forms an interval.

A *solution* to an STP is an assignment of values to each time point such that all constraints are satisfied. An STP is said to be *consistent* if at least one solution exists. Consistency-checking in a STP can be cast as an all-pairs shortest path problem: the STP is consistent iff there are no negative cycles in the all-pairs graph. This check can be performed in $|X|^3$ time. A by-product of this check is the *minimal network*, which is the tightest representation of the STP constraints that still contains all solutions present in the original network. A single solution can be extracted from the minimal network in $|X|^2$ time (Dechter, Meiri, & Pearl 1991).

To extend an STP to an STP with preferences (STPP), one adds to each constraint a preference function that maps each value in the interval identified by that constraint to a *preference value*, a quantitative measure of the value's desirability. Hence the constraints in an STPP have the following form:

$$< X_j - X_i \in [a_{ij}, b_{ij}], f_{ij} : x \in [a_{ij}, b_{ij}] \rightarrow \Re > .$$

A Temporal Constraint Satisfaction Problem (TCSP) extends an STP by allowing multiple intervals in constraints:

$$X_j - X_i \in \{[a_{ij1}, b_{ij1}], ..., [a_{ijn}, b_{ijn}]\}.$$

This expresses the constraint that the distance between X_i and X_j must be in one of the intervals listed. The extra expressiveness comes at a high cost: determining consistency for a general TCSP is NP-hard (Dechter, Meiri, & Pearl 1991). Note that a TCSP constraint can be viewed as a disjunction of STP constraints over the same two time points: $X_j - X_i \in [a_{ij1}, b_{ij1}] \vee X_j - X_i \in [a_{ij2}, b_{ij2}] \vee \ldots$. A standard way of checking the consistency of a TCSP is to view it as a collection of component STPs, where, intuitively, each component STP is defined by selecting one STP constraint (i.e., one interval) from each TCSP constraint. Checking the consistency of the TCSP then amounts to searching for a consistent component STP, since a solution to a component STP will also be a solution to the complete TCSP.

A Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis 2000) removes the binary restriction of TCSPs, resulting in constraints with the following form:

$$(X_{j1} - X_{i1} \in [a_{i1j1}, b_{i1j1}]) \vee (X_{j2} - X_{i2} \in [a_{i2j2}, b_{i2j2}]) \vee \cdots$$
$$\vee (X_{jn} - X_{in} \in [a_{injn}, b_{injn}]).$$

Clearly, the TCSP is a subcase of the DTP in which $X_{j1} = X_{j2} = \ldots X_{jn}$, and $X_{i1} = X_{i2} = \ldots X_{in}$. As with a TCSP, the standard way to solve a DTP is to extract component STPs, by selecting one disjunct from each constraint, and searching for a consistent component STP.

To our knowledge, there has not been previous work on extending TCSPs and DTPs to handle preferences. It is clear, however, what the form of such an extension should be: each individual disjunct of a TCSP or DTP constraint is given an associated preference function that maps the values in the identified interval to preference values.

Example

To motivate the need for preferences we describe an example from the domain of Autominder, a system that uses DTPs to manage the daily plans for persons with memory impairment

(Pollack *et al.* 2003). An elderly woman must exercise for 30 uninterrupted minutes in late afternoon (3-6pm) and take heart medication shortly after. A friend will visit from 4:30 to 5, so it is best if the exercise occurs well before the visit, to allow for cool-down, or else after it. The task is to model this scenario with a temporal network and use consistency-checking to determine when to begin exercising and when to take the medicine. The example has seven time points to model: a temporal reference point (TRP), which is used for stating absolute (clock-time) constraints; and the start and end of the Exercise, TakeMeds, and Visit actions.

```
Time Points:   TRP (3pm):   TRP
               Exercise:    E_S, E_E
               TakeMeds:    T_S, T_E
               Visit:       V_S, V_E
Constraints:   T_S - E_E ∈ [5, 20]
               V_S - E_E ∈ [5, ∞] ∨ E_S - V_E ∈ [0, ∞]
               V_S - TRP = 90
               V_E - V_S = 30
               E_S - TRP ∈ [0, ∞]
               E_E - TRP ∈ [-∞, 360]
               E_E - E_S = 30
```

Figure 1: DTP encoding of the Autominder example.

Figure 1 shows a DTP encoding of this example. The first constraint states that TakeMeds must start 5 to 20 minutes after finishing exercising, the second expresses when exercise can occur relative to the planned visit, the third constrains the starting time of the visit, and so on. Consider the first constraint. We assume that its parameters have been chosen by a medical expert who understands the effects of the medicine in relation to exercise. In the DTP, the interval [5, 20] represents the allowable time difference between the finishing the Exercise action and starting the TakeMeds action. Any other time, say 4, 21, or -30, is not allowed and represents an inconsistency or failure in the DTP. Notice that in a DTP, a difference of -30 is no worse than a difference of 4; both are considered failures.

When specifying bounds on time differences, the expert must reason about a trade-off: a wider set of bounds increases the likelihood that the entire DTP is consistent, while more narrow bounds keeps actions closer to their ideal times. Therefore, specifying good bounds requires knowledge about the underlying relationship *and* how the bounds will affect the rest of the network. This type of situation motivates the need for preferences. The doctor may want to express that the ideal time difference between Exercise and TakeMeds is the interval [5,10); that the intervals [3,5) and [10,20) are almost as good; and that the intervals [0,3) and [20, 25] are acceptable, but not preferable. This could be implemented by attaching a preference function f to the first constraint in the DTP, such that $f([5, 10)) = 3$, $f([3, 4)) = 2$, $f([10, 20)) = 2$, $f([0, 3)) = 1$, and $f([20, 25]) = 1$. Expressing preferences in this way avoids the trade-off mentioned above. The expert can express the local relationship without regard to its affect on the DTP as a whole.

Our example encoding can be improved by adding preferences to the disjunctive (second) constraint as well. The

DTP encoding states that Exercise must finish at least 5 minutes before the visitor arrives *or* anytime after the visitor leaves. We might want to elaborate this to say that finishing at least 15 minutes ahead is most desirable; finishing 10 minutes early is the next preference and is equivalent to the option of starting the exercises after the visitor leaves; and finishing only 5 minutes before the visitor arrives has the lowest preference of all allowed possibilities. To encode this set of preferences, we would use two preference functions. Let f be associated with the first disjunct of the constraint; then $f([15, \infty]) = 3$, $f([10, 15)) = 2$, and $f([5, 10)) = 1$. Similarly, let g be associated with the second disjunct. It assigns a value of 2 to the entire interval for that disjunct: $g([0, \infty]) = 2$. By default, each preference function assigns a value of 0 to all disallowed points.

In this example, the preference functions were step functions, mapping intervals to each preference level. In general, however, any function, discrete or continuous, that maps temporal differences to preference values is acceptable.

Solving DTPs

As noted above, DTPs (and TCSPs) can be viewed as collections of component STPs, and solving a DTP (or TCSP) can thus be achieved by searching for a consistent component STP. This insight underlies the Forward and Go-Back algorithm of Dechter *et al.* (1991). Although this algorithm was originally designed to solve TCSPs, it applies directly to DTPs as well, since it does not exploit the fact that the constraints in a TCSP are binary. In fact, the approach of searching for component STPs is the basis of most modern DTP-solving algorithms (Stergiou & Koubarakis 2000; Oddi & Cesta 2000; Tsamardinos & Pollack 2003). Consequently, in the reminder of the paper, we shall refer only to DTPs; the reader should keep in mind that everything we write also applies to the special case of TCSPs.

The Forward and Go-back algorithm works by searching the space of partial component STPs within a given DTP, where a partial STP consists of single STP constraints extracted from a subset of the DTP constraints. It builds the STPs one constraint at a time, adding a constraint if the current set is consistent, and backtracking otherwise, looking for complete component STPs that are consistent. (A component STP is complete if it includes a disjunct from every constraint in the original DTP). The original version of the algorithm found *all* consistent component STPs, and returned a network consisting of the union of their minimal networks. For many applications, however, it suffices to find the minimal network of a single consistent component STP.

Figure 2 shows the Forward and Go-back algorithm modified in two ways. First, it returns the minimal network of the first consistent STP found. Second, the bookkeeping aspects of the algorithm have been modified to enable it to be extended to an algorithm that solves DTPs with preferences. Because of these bookkeeping modifications, a line-by-line comparison of our version of the algorithm with the original one reveals few similarities, but in fact they search through the space of component STPs in the same order.

The algorithm operates on a DTP represented as a list of n DTPs constraints, each of which is itself a set of STP

Solve-DTP(i)
1. select[i] = select[i] + 1; // Choose next disjunct in constraint i
2. if select[i] > S_i // If true, no more disjuncts in constraint i
3.   begin
4.   if i = 1 return failure // no solution
5.   select[i] = 0; //Reset disjunct in this constraint
6.   Solve-DTP(i-1); // Try next disjunct in previous constraint
7.   end
8. if Consistent-STP(D_1(select[1]), ..., D_i(select[i]))
9.   begin
10.   if i = n return Minimal Network of selected STP
11.   Solve-DTP(i+1); // Add another constraint
12.   end
13. else
14.   Solve-DTP(i); // Try the next interval in this constraint

Figure 2: Solve-DTP, an algorithm modified from the Forward and Go-back algorithm. (Dechter, Meiri, & Pearl 1991)

constraints; by definition S_i is the number of STP constraints (i.e., the number of disjuncts) in the ith DTP constraint. Thus each DTP constraint is defined as $D_i = [\mathrm{STPC}_1, \ldots, \mathrm{STPC}_{S_i}]$, where STPC_k is an STP constraint and i ranges from 1 to n. A vector $select[]$ stores the indices of the STP constraints currently under consideration in the algorithm; if $select[j] = k$, then the kth disjunct of D_j is included in the current partial STP. The vector is initialized to 0's, meaning no disjuncts are selected at the algorithm's start. If $select[] = \{2, 1, 0, 0\}$, it identifies a partial STP composed of the second disjunct in D_1 and the first disjunct in D_2. Constraints D_3 and D_4 are not included.

The algorithm uses one internal function, Consistent-STP, which checks an STP for consistency. The external call to Solve-DTP should pass in the integer '1' for i, indicating that it starts the search with the first DTP constraint.

Line 1 of the algorithm chooses the next disjunct for constraint D_i by incrementing the constraint's index. If the index goes out of bounds (line 2), lines 4-6 remove D_i from the current component STP, and backtrack to try the next disjunct of the previous constraint D_{i-1}.

Line 8 checks the consistency of the partial STP defined by the $select[]$ vector. If it is both consistent and complete (lines 8 and 10), then a solution has been found. If consistent but not yet complete, then the algorithm is called recursively to find a disjunct from the next DTP constraint D_{i+1} (line 11). If it is not consistent, then the algorithm is called again with the current DTP constraint D_i so that its next disjunct can be considered (line 14).

The time complexity of this algorithm depends on the number of component STPs in the DTP, which in turn depends upon the number of DTP constraints and the number of disjuncts in each. If the DTP contains $|X|$ time points and n constraints with a maximum of k disjuncts, there exists $O(k^e)$ STPs to check for consistency. In the worst case, where no consistent STP exists, the total complexity is $O(k^e * |X|^3)$. The number of STP checks can be significantly reduced through pruning techniques that will be discussed briefly in the analysis section of this paper.

Solving STPPs

As noted earlier, recently an algorithm has been developed for finding optimal solutions to STPPs (STPs with preferences) with semi-convex preference functions (Khatib *et al.* 2001; 2003). Like the algorithm for solving DTPs, this algorithm works by searching a space of projected STPs. This time, however, the projected STPs (of the STPP) are defined by the preference functions associated with each STPP constraint. Specifically, given an STPP constraint $< X_j - X_i \in [a, b], f : x \in [a, b] \to \Re >$, for every preference level p we can project a constraint of the form $X_j - X_i \in [c, d]$, where the interval $[c, d] = \{x : f_{ij}(x) \geq p\}$. If f_{ij} is semi-convex, the projected points will always form at most a single interval, resulting in an STP constraint. Moreover, there will be at most a single complete projected STP at level p.

To find a weakest-link optimal solution to an STPP then, one can search for the maximum p such that there exists a consistent projected STP at preference level p. If preference levels are discretized into the set A, a binary search through preference levels will find the consistent STP with the highest preference level in $\log(|A|) * |X|^3$ time.

Extending the algorithm to handle DTPPs

Our approach to finding WLO-optimal solutions to DTPPs (and TCSPPs) combines the idea of projecting networks with hard temporal constraints from those with preferences, with the idea of projecting component STPs from DTPs. At a high level, the algorithm has two phases, as illustrated in Figure 3. In the first, the preference values in the DTPP are discretized and a set of DTPs is projected from the DTPP– one DTP for each preference value. As explained below, this is done in a manner similar to that of the STPP algorithm. In the second phase, an attempt is made to solve each DTP by searching for a consistent component STP. The goal is to find the DTP with highest preference value that is consistent.

To project a DTP at preference level p from a DTPP, we use an approach analogous to that of projecting an STP from an STPP: each *disjunct* $< X_j - X_i \in [a, b], f_{ij} >$ is projected to $X_j - X_i \in [c, d]$ where $[c, d] = \{x : f_{ij}(x) \geq p\}$. Note that just as projecting an STPP constraint to a given preference level produces an STP constraint, projecting a DTPP constraint to a given level produces a set of disjunctive STP constraints, i.e., a DTP constraint. Consequently, a
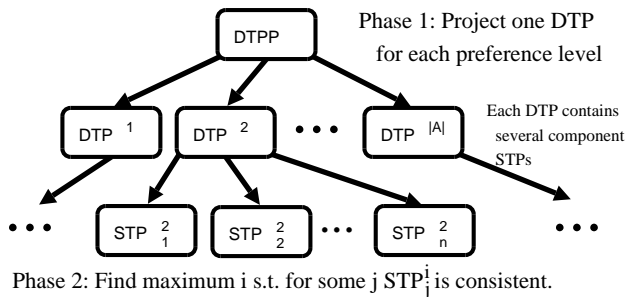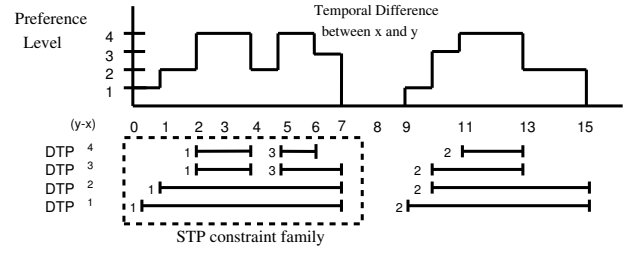

Figure 4: A DTPP (actually a TCSPP) constraint containing two disjuncts projected onto 4 DTP constraints. Each interval is labeled with the index assigned during projection.

projection of a DTPP to preference level p is a single DTP, which we label DTP^p.

Figure 4 provides an example; for ease of illustration, the example is actually a TCSPP constraint, but the process is identical for general DTPP constraints. The set of disjuncts produced by projecting constraint i at preference level p is stored in D_i^p, while the number of disjuncts is stored in S_i^p; the superscript p is necessary because we do not restrict preference functions to be semi-convex, allowing DTPs at different levels to have different numbers of disjuncts (see Figure 4). Whenever a disjunct splits, as in levels 2 and 3 in Figure 4, the projection algorithm must assign one the index of the original interval and assign an unused index to the other.

The result of the projection phase is a set of $|A|$ DTPs, where $|A|$ is the number of discrete preference levels. Notice in Figure 4 that each disjunct of the DTPP constraint produces multiple STP constraints, which we refer to as an STP constraint *family*. Within a family each constraint is *tighter* than members at lower levels, meaning, the set of temporal difference values allowed by the constraint is a subset of those allowed by members at lower levels. We can define these terms for entire STPs as well: STP$_1$ is *tighter* than STP$_2$ if each constraint in STP$_2$ corresponds to a unique tighter constraint in STP$_1$; and STP$_1$ is *related* to STP$_2$ if each constraint in STP$_2$ has a constraint family member in STP$_1$.

These definitions allow us to describe two properties of the projected DTPs that make our algorithm efficient. First, each component STP of DTP^p will be related to and tighter than a component STP of each DTP^q for $q < p$. Because an STP has a solution if any tighter STP does, we obtain the Downward Consistency property:

Property 1 *Downward Consistency. If there is a consistent component STP in DTP^p, then there is a consistent STP for every DTP^q for all $q < p$.*

The converse property is Upward Inconsistency:

Property 2 *Upward Inconsistency. If a component STP of DTP^q is inconsistent, then any related component STP of DTP^p for $p > q$ will also be inconsistent.*

This property holds because the higher-level STP constraints are tighter than the lower-level STP constraints belonging to the same family. If no solution exists in an STP, no tighter version of that STP will contain a solution.

Figure 3: The two phases of solving a DTPP

The upward inconsistency property allows us to prune away a large part of the search space each time an inconsistent STP is found. For example, if the algorithm finds an inconsistent STP at preference level 1, then it can prune away all related higher-level STPs.

The algorithm for the first phase depends on the original form of the preference function and the discrete preference levels chosen. For most reasonable forms of preference functions (e.g. piecewise linear), this step is inexpensive compared to the second step of the algorithm, which involves searching through the projected DTPs to find the one with highest preference value that is consistent. This is done using the technique of searching for a conistent component STP from within the projected DTPs.

The entire algorithm is shown in Figure 5. After projecting DTPs from the input DTPP, it searches through partial component STPs within the lowest level DTP until it finds a consistent STP. Once a solution is found, the search moves to the next preference level. At the new level, the search starts with an STP that is related to the consistent one just found at the level below. At each level, the solution found (the minimal network of the consistent component STP) is stored as 'bestSolution'. The search ends when no solution is found for a given level. There is no need to search higher levels in this case, as the upward inconsistency property ensures that it is not possible for a solution to exist at one.

Solve-DTPP-Main(dtpp)
1. Let D = constraints in dtpp
2 $\forall i, \forall p. [D_i^p, S_i^p]$ = project(D_i, p)
3. bestSolution = Ø;
4. Solve-DTPP(1, 1)

Solve-DTPP(i, p)
1. if p $>$ |A| return bestSolution; // Solution found at highest p-level.
2. select[i] = select[i] + 1; // Choose next disjunct in constraint i
3. if select[i] $> S_i^p$ // If true, there are no more disjuncts in constraint i
4.   begin
5.   if i = 1 return bestSolution; // no solution at this p-level
6.   select[i] = 0;
7.   Solve-DTPP(i-1, p); // Try next interval in previous constraint
8.   end
9. if Consistent-STP(D_1^p(select[1]), ..., D_i^p(select[i]))
10.   if i = n
11.   begin
12.   bestSoln = Minimal Network of selected STP
13.   p = p + 1;
14.   select[i] = select[i] - 1;
15.   Solve-DTPP(i, p); // Try related STP at higher preference level
16.   end
17.   else
18.   Solve-DTPP(i+1, p); // Add another constraint
19. else
20.   Solve-DTPP(i, p);// Try the next interval in this constraint

Figure 5: Algorithm for Solving DTPPs. Bold lines indicate differences between Solve-DTP and Solve-DTPP.

The initial call to `Solve-DTPP` starts the search at the first DTP constraint and at preference level 1. The algorithm behaves exactly like Solve-DTP until it finds the first solution (line 11). At this point it stores the solution, moves up to the next preference level, and tries a related STP at the higher level (lines 12-15). This continues until a solution is found at the highest preference level (line 1) or no more solutions are found (line 5).

When the algorithm moves up to the next preference level, it does not search the entire space of STPs at that level: it "skips" STPs that are related to the STPs already searched at the lower level[2]. Operationally, this skipping is enacted by leaving the $select[]$ indexes unchanged during the shift to the higher level[3]. The upward inconsistency property tells us that skipping these STPs is justified, since they are tighter than STPs already found to be inconsistent at lower levels.

Analysis

A simple algorithm for solving DTPPs is to run Solve-DTP (*not Solve-DTPP*) for the DTP projected at the highest preference level and decrement the level until a solution is found. This algorithm would have an O(|A|* complexity(Solve-DTP)) worst case run time, where complexity(Solve-DTP) is the complexity of solving the largest of the |A| DTPs. A slight improvement can be made by performing a binary search through the preference levels, reducing the worst case complexity to O(($log|A|$) * complexity(Solve-DTP)).

Unlike these two simple algorithms, however, our algorithm does not require multiple calls to the Solve-DTP procedure. It modifies the procedure to achieve a time complexity of O(|A| * complexity(Solve-STP) + complexity(Solve-DTP)); hence the additional cost of preferences over hard DTPs is at most $|A| * |X|^3$.To understand why this is so, notice that only two pieces of code were added to Solve-DTP to produce Solve-DTPP. The first addition (line 1) does not affect the trajectory through the search space; it simply checks for an exit condition. The second addition (lines 11-16) executes only after a valid solution for a level is found. These lines store the solution, increment the preference level, and continue the search with the *related* STP in the search space one preference level higher. Since Solve-DTPP moves to the next preference level each time it finds a consistent STP, it will find at most $|A| - 1$ consistent STPs and will thus make at most $|A| - 1$ additional STP consistency checks, using a $|X|^3$ time algorithm.

Improving Efficiency

The shape of preference functions can greatly influence the additional cost of handling preferences. Consider Figure 6. In this network, consisting of only two constraints, the DTP at level 2 contains 9 STPs, whereas the DTP at level 3 contains only 1. If a solution exists at level 3, then starting at level 1 using Solve-DTPP may require up to 11 STPs to be checked for consistency, where starting at the solution level would only require a single check.

The general issue here is that the number of component STPs in the largest DTP may be much larger than the number in the solution level DTP. Since the algorithm may

---

[2]We should note that *some* related STPs are not skipped, because intervals can split as the preference level increases.

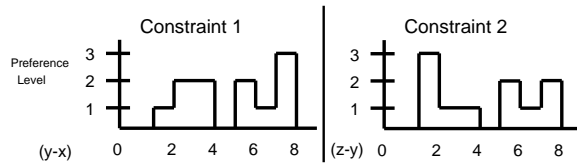[3]Lines 14 and 2 cancel each other out to produce no change.

Figure 6: DTPP constraints whose lower level DTPs have much greater complexity than the upper level DTP.

search the largest DTP completely before reaching the solution level, our claim that cost of adding preferences is at most $|A| * |X|^3$ greater than the time needed for the largest DTP, while true, is misleading.

Fortunately, this issue can be addressed by careful assignment of indices during the projection step of the algorithm. When a disjunct I splits into I_1 and I_2, the one that persists to the highest preference level should be assigned the lower index. This causes the algorithm to search STPs related to the top-level STPs first. Using this strategy, a stronger claim can be made: the cost of adding preferences is at most $|A| * |X|^3$ greater than the time needed for solving the *solution-level* DTP.

A second approach to increasing efficiency takes advantage of the relationship between STPs at adjacent preference levels. If a solution to a DTPP exists at the highest preference level, the algorithm as presented must find solutions at all lower levels before reaching the optimal one. However, by intersecting the constraints of an STP S at level $k+1$ with the minimal network of the unique STP related to it at level k, we obtain information that may allow us to immediately determine whether S is consistent or inconsistent, thereby obviating the need to run the check. If the intersection produces an empty set, then S is guaranteed to be inconsistent. On the other hand, if the intersection returns the minimal network of the STP at level k, then S is guaranteed to be consistent. In both cases, a consistency check is unnecessary. The average savings derived from this technique depends on the shape of the preference functions. For example, a rectangle or steep-sloped rhombus may often produce the second case (immediate determination of consistency).

Further increases in efficiency can be obtained by exploiting techniques that have been developed for pruning the search space of TCSPs and DTPs, including Conflict-Directed Backjumping, Semantic Branching, and Removal of Subsumed Variables, and No-good Recording (Tsamardinos & Pollack 2003). Each of these methods can be incorporated with few modifications into the Solve-DTP algorithm, and, although we do not provide adequate explanation in this paper, they can similarly be adopted into Solve-DTPP.

Finally, note that because it starts its search at the lowest preference level, Solve-DTPP has an anytime quality. In practice, since DTPs at lower levels are less tight, it will be easier to find a solution at a lower level. The algorithm can be interrupted anytime after the first solution is found if available computation time is exhausted. Given that general DTPs are NP-Hard, this is an important property.

Future Work

The Solve-DTPP algorithm, like the STPP algorithm in (Khatib *et al.* 2001), provides solutions with Weakest Link Optimality, meaning that it finds solutions whose lowest preference constraint is maximal. For a given problem, several solutions of equal value can exist, some of which may dominate others, meaning although the lowest preference constraint for each solution is at the same level, all constraints of one is satisfied at equal or greater level than the other. For semi-convex STPPs, an algorithm has been developed to search out and remove all dominated solutions, leaving a set of Pareto optimal solutions for that STPP (Khatib *et al.* 2003). It would be useful to develop a similar algorithm for DTPPs.

An empirical comparison between solving tractable DTP problems and DTPPs randomly generated from them would provide a characterization of the impact of the various efficiency strategies described in the previous section, as well as the usefulness of Solve-DTPP's anytime property.

References

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

Khatib, L.; Morris, P.; Morris, R.; and Rossi, F. 2001. Temporal constraint reasoning with preferences. *17th International Joint Conference on Artificial Intelligence* 1:322–327.

Khatib, L.; Morris, P.; Morris, R.; and Venable, K. B. 2003. Tractable pareto optimal optimization of temporal preferences. *18th International Joint Conference on Artificial Intelligence* 1:1289–1294.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on AI*, 494–499.

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

Pollack, M. E.; Brown, L.; Colbry, D.; McCarthy, C. E.; Orosz, C.; Peintner, B.; Ramakrishnan, S.; and Tsamardinos, I. 2003. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems* 44(3-4):273–282.

Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:81–117.

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

Tsamardinos, I.; Pollack, M. E.; and Ramakrishnan, S. 2003. Assessing the probability of legal execution of plans with temporal uncertainty. In *Proceedings of the ICAPS03 Workshop on Planning under Uncertainty and Incomplete Information*.

Wald, A. 1950. *Statistical Decision Functions*. Wiley, New York.