Solving Over-constrained Disjunctive Temporal Problems with Preferences

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

Abstract

We present an algorithm and pruning techniques for efficiently finding optimal solutions to over-constrained instances of the Disjunctive Temporal Problem with Preferences (DTPP). Our goal is to remove the burden from the knowledge engineer who normally must reason about an inherent trade-off: including more events and tighter constraints in a DTP leads to higher-quality solutions, but decreases the chances that a solution will exist. Our method solves a potentially over-constrained DTPP by searching through the space of induced DTPPs, which are DTPPs that include a subset of the events in the original problem. The method incrementally builds an induced DTPP and uses a known DTPP algorithm to find the value of its optimal solution. Optimality is defined using an objective function that combines the value of a set of included events with the value of a DTPP induced by those events. The key element in our approach is the use of powerful pruning techniques that dramatically lower the time required to find an optimal solution. We present empirical results that show their effectiveness.

Introduction

Several types of Temporal Constraint Satisfaction Problems (TCSPs) (Dechter, Meiri, & Pearl 1991) have been used successfully in recent years as components in planning and scheduling applications, including Autominder (Pollack *et al.* 2002) and NASA's Mars Rover (Muscettola *et al.* 1998). In planning and scheduling domains, the variables in a TCSP represent events to be scheduled or executed by an agent, and the constraints specify allowable times and temporal differences between events. The main task for these applications, when given a TCSP, is to find an assignment of times to all variables that respects all constraints.

One aspect of TCSPs that is often ignored is the task of specifying the events and constraints that compose them. The choice of which events to include and how much to constrain the events has great impact on whether the TCSP has a solution. Typically, the burden is on the knowledge engineer (KE) to reason about whether the addition of an event or the tightening of a constraint will over-constrain the problem. In this paper, we remove the burden from the KE with a technique that combines recent advances in reasoning about preferences in TCSPs with a new approach for choosing the optimal set of events to include in the TCSP.

To understand why specifying TCSPs can be difficult, consider the task of defining a TCSP for a single day in the life of an office worker. The goal (for most people) is to get as much as possible done in that day while retaining at least a minimum level of quality in their work. For example, one task may be to create a presentation, which can be done poorly in a small amount of time, or done well in a larger block of time. Deciding whether or not to include this task in the day's plan depends on which other activities are scheduled. For a busy day, we may want to exclude the task altogether, or include it with a constraint that sets the minimum time allocated for the task to a small value. For a slow day, however, we would want to include the task and set the minimum allowed time to a greater value, ensuring a higher-quality presentation.

Two separate limitations of standard TCSPs appear in this example. First, standard TCSPs require hard constraints, which cannot express sentiments such as "it is better to spend more time on a presentation than less." Because hard constraints often do not reflect reality, a KE specifying bounds on time differences must reason about a trade-off: a wider set of bounds increases the likelihood that the entire set of constraints is consistent, while narrower bounds keeps events closer to their ideal times. Therefore, specifying appropriate bounds for a constraint requires knowledge about the real-world situation it models *and* how the bounds might effect the feasibility of the problem.

Second, TCSPs require all specified events to be included in their solutions. This all-or-nothing requirement forces the KE to be careful about which events to include in the problem. Adding too many events over-constrains the problem and does not allow any solutions; adding too few events results in under-utilized resources. Therefore, knowing which events to include in the problem requires the KE to reason about how constrained the problem is — reasoning that should be left to the TCSP algorithms.

In this paper, we remove the burden from the KE by addressing both limitations. First, we allow the KE to express preferences, that is, to designate some bounds as more or less desirable than others when defining a constraint. This resolves the first trade-off: the expert can define a constraint without regard to its affect on the TCSP as a whole. Sec-

ond, we provide a method for specifying (1) the relative importance of each event, (2) which events must coexist, and (3) the relative importance of getting highly preferable solutions versus getting solutions that include many events. Taken together, this information defines an over-constrained TCSP with Preferences (TCSPP). We define an algorithm that uses this information to find an optimal solution to the over-constrained TCSPP. We focus on Disjunctive Temporal Problems with Preferences (DTPPs) (Peintner & Pollack 2004), but the results carry through to other TCSP subproblems as well. We present a representation and associated algorithm that allow the KE to add as many events as desired into a DTP without worrying about how many will "fit", and allow the KE to specify soft constraints (preferences) without regard to how they affect the rest of the problem.

Our approach is to cast the problem as a multi-objective optimization problem: the first objective is to include as many events as possible (plan capacity), and the second is to choose and tighten constraints in a way that maximizes preference (plan quality). The overall plan value is defined to be some function over these two metrics. The general algorithm itself is simple and obvious: for certain subsets of events in the DTPP, we build an *induced DTPP* that contains only constraints that reference the events in the subset. Then, we solve the induced DTPP to find its optimal value. The subset of events and associated solution that maximize the objective function is chosen as the optimal solution.

The primary contribution of this paper is not this general algorithm, but the pruning techniques and evaluation functions we use to make the process efficient. First, we define an evaluation function for the event set that greatly reduces the number of event combinations that need to be checked. Second, we show how information learned solving the induced DTPP in one iteration can be used to reduce search when solving the induced DTPP in the next.

Example

To motivate the need for optional events and preferences, we describe an example from the domain of Autominder, a system that currently uses DTPs to manage the daily plans of persons with memory impairment (Pollack *et al.* 2003).

Consider the following constraints on an elderly woman's activities in late afternoon (3–5pm). She must exercise for 25 uninterrupted minutes shortly after taking heart medication, either before or after a friend visits from 3:45 to 4:15. If Exercise ends before the visit, it is best that it ends well before to allow her to recover (although it would be even better for Exercise to occur sometime after the visit). The woman would also like to knit for at least half an hour (the yarn is brought by the visitor, so this must occur after the visit). There is a short interview on the radio of interest from 3:00 to 3:15, and also a marathon of Matlock reruns showing from 3pm to 5pm.

The task is to model this scenario with a DTP and use it to determine when to take the medication, when to begin exercising, when to knit (if at all), and whether to listen to the radio interview and/or watch the Matlock marathon. To represent this situation, we use a DTP with twelve time-points, depicted in Table 1: a temporal reference point (TRP), which is used for stating absolute (clock-time) constraints; a time-point representing the TakeMeds event (assumed instantaneous); and time-points representing the start and end of the Exercise, Visit, Knitting, Radio, and Matlock intervals. Along with each time point, we define a weight that indicates the relative importance of the event.

Time-points		Worth
TRP (3pm)	TRP	(∞)
TakeMeds	T	(8)
Exercise Start/End	E_S, E_E	(8)
Visit Start/End	V_S, V_E	(6)
Knit Start/End	K_S, K_E	(4)
Radio Start/End	R_S, R_E	(2)
Matlock Start/End	M_S, M_E	(1)

Table 1: Events in the Autominder example.

Constraints
$C_1 : T - TRP \geq 0$
$C_2 : V_S - TRP = 45$
$C_3 : R_S - TRP = 0$
$C_4 : M_S - TRP \geq 0$
$C_5 : E_E - E_S = 25$
$C_6 : V_E - V_S = 30$
$C_7 : K_E - K_S \geq 30$
$C_8 : R_E - R_S = 15$
$C_9 : M_E - M_S = 120$
$C_{10} : E_S - T \in [5, 20]$
$C_{11} : K_S - V_E \geq 0$
$C_{12} : (M_S - E_E \geq 0) \vee (E_S - M_E \geq 0)$
$C_{13} : (R_S - E_E \geq 0) \vee (E_S - R_E \geq 0)$
$C_{14} : (K_S - E_E \geq 0) \vee (E_S - K_E \geq 0)$
$C_{15} : (V_S - E_E \geq 0) \vee (E_S - V_E \geq 0)^*$
$C_{16} : (V_S - T \geq 0) \vee (T - V_E \geq 0)$

Table 2: DTP encoding of some constraints in the example.

Table 2 shows an DTP encoding of the constraints in this example. The constraints C_1 through C_4 constrain the Visit, TakeMeds, Radio, and Matlock actions to occur after 3pm (the time for event TRP), and C_5 through C_9 constrain the durations of each action. Constraints C_{10} through C_{16} encode some of the more interesting aspects of the problem. For instance, C_{10} represents a doctor's recommendation that the medication should be taken within 5-20 minutes before exercising. C_{11} constrains knitting to occur after the visit. The remaining disjunctive constraints prevent overlap of the Exercise interval and other events in the plan. As an example, C_{13} requires either the Radio action to start after Exercise ends, or the Exercise action to start after Radio ends. C_{16} enforces a similar constraint between the Visit action and TakeMeds, ensuring that the woman does not interrupt her visit to take her medication. Several other constraints exist, expressing additional overlap restrictions and the fact that all activities must finish by 5pm, but we do not explicitly state these in the table.

One of these constraints, C_{15}, is marked with an asterisk because each of the two disjuncts in this constraint has varying degrees of preference, depending on the value of

Solution 1	Solution 2	Solution 3
T[15,15]	T[0,10]	T[75,90]
E_S[20]	E_S[5,15]	E_S[80,95]
E_E[45]	E_E[30,40]	E_E[105,120]
V_S[45]	V_S[45]	V_S[45]
V_E[75]	V_E[75]	V_E[75]
K_S[75,90]	K_S[75,90]	$K_S \rightarrow$ removed
K_E[105,120]	K_E[105,120]	$K_E \rightarrow$ removed
R_S[0]	$R_S \rightarrow$ removed	$R_S \rightarrow$ removed
R_E[15]	$R_E \rightarrow$ removed	$R_E \rightarrow$ removed

Table 3: Possible solutions for the Autominder example.

its respective temporal difference. If the first disjunct is selected, we will tolerate the difference $V_S - E_E$ to be 0, but we would prefer it to be at least 5 minutes (to allow some recovery time before the visitor arrives). We would prefer even more for the second disjunct to be selected instead.

After careful inspection of this example, it is apparent that there is no feasible solution that includes all events. This is because the Matlock marathon covers the entire makespan of the schedule, and thus overlaps with all other actions. Now, suppose we ignore the Matlock marathon and its respective constraints. For this problem, there is a feasible solution, given as Solution 1 in Table 3. Unfortunately, since the plan is fairly compact, it allows no time for the woman to recover between Exercise and the visit. Thus, we satisfy constraint C_{15} at the lowest level. If we remove the Radio action and its constraints, we get Solution 2, where the Exercise interval is moved forward, giving slightly more time before the visitor arrives. To achieve the highest preference level for C_{15}, we would also need to remove the knitting action and its constraints — this allows Solution 3, where Exercise can occur after the visit.

This example demonstrates the inherent trade-off between plan capacity and plan quality. Larger problems that involve an inordinate number of events are at even greater risk of being over-constrained, admitting either no solutions or solutions that meet only a minimal set of standards.

Background

STPs and DTPs

The most restricted subclass of temporal CSPs is the Simple Temporal Problem (STP). An STP is a pair $\langle X, C \rangle$, where the elements $X_i \in X$ designate time-points, and C is a set of binary temporal constraints of the following form:

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

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

A Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis 2000) is a pair $\langle X, C \rangle$, where each element of C is a disjunction of STP constraints as in the following:

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

To satisfy a DTP constraint, only one of its disjuncts needs to be satisfied. An assignment that satisfies at least one disjunct in each constraint is a solution to a DTP.

DTPs are useful in planning and scheduling applications because they can represent the so-called *promotion/-demotion constraints* that frequently occur. In addition, plans and schedules often allow activities to occur at multiple times or allow multiple activities to be assigned to a single time slot; disjunctive constraints model both of these situations well.

Recent work has used partial constraint satisfaction to solve over-constrained DTPs (Moffitt & Pollack 2005). The approach differs from ours in that it reasons about removing only constraints and not events, and that it does not reason about the degree to which a constraint is satisfied.

Meta-CSP formulation of DTP

A key construct used for solving a DTP is a *component STP*, which is created by selecting one disjunct from every DTP constraint. A given DTP has a solution if and only if one of its component STPs does. Therefore, the search for a solution to a DTP can be carried out by searching for a solution in each of its component STPs.

Finding a solution to an STP requires only polynomial time, but a DTP can have an exponential number of component STPs. Fortunately, backtracking search and pruning techniques make this approach practical in many cases: the component STP is built up one disjunct at a time, backtracking if the current set of disjuncts are inconsistent.

The search through a DTP's component STPs is often cast as a search through a meta-CSP derived from the DTP. Each variable in the meta-CSP corresponds a disjunctive constraint in the DTP; each disjunct in the disjunctive constraint is represented by a single value in the meta-CSP variable. Assigning a meta-CSP variable to a particular value is therefore equivalent to choosing a particular disjunct from a constraint to include in a component STP. Consequently, a set of assigned variables in the meta-CSP defines an STP composed of all disjuncts implied by the assignments.

Representing Preferences in DTPs

Allowing preferences to be defined on each DTP constraint gives the KE the ability to avoid the trade-off inherent in defining hard bounded constraints.

To extend a DTP to a DTP with Preferences (DTPP), disjuncts of every constraint are assigned a preference function that maps each value in the allowed interval identified by that disjunct to a *preference value*, a quantitative measure of the value's desirability. Hence the disjuncts in a DTPP constraint take the following form:

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

In our example, we alluded to these preference functions when discussing constraint C_{15}, which precluded the overlap of the Exercise and Visit intervals. For the first disjunct, 0 minutes was acceptable, while 5 minutes or more was better. Hence, its preference function could output the value 1 for any time difference greater than 0 and less than 4, and 2 for any difference greater than or equal to 5. Also, recall that any assignment which chooses the second disjunct should have an even higher preference level. So, the preference function for that difference could be 3 for all values greater than 0. If desired, other constraints could be given their own preference functions as well. For instance, preferences on durations or start times can also be specified.

The addition of preference functions changes the problem from one of finding *any* solution for the problem to finding the *optimal* solution. We evaluate the quality of some solution with some objective function that takes as input the output of each constraint's preference function. In this paper, we use the *maximin* function, which sets the preference value v_P of a solution to the minimum value of all the preference function outputs. The maximin function leads to efficient algorithms for finding optimal solutions to DTPPs (Peintner & Pollack 2004).

Basic Approach

In this section, we provide a means to reason about potentially over-constrained DTPPs, balancing the trade-off between plan capacity (i.e., the set of events included in the solution), and plan quality (i.e., the degree to which each preference function is satisfied in the solution). To manage this trade-off, we need (1) a way to evaluate the value of including a particular set of events, (2) a way to evaluate the induced DTPP for that set of events, and (3) a global objective function that combines the value of both to produce the overall plan value. We will discuss all three in turn.

To determine the value of including a set of events, we first augment the DTPP representation to include a weight vector that specifies the cost of excluding each event x_i, $w(x_i)$. We defined the costs for each event in our example in the last column of Table 1. We then define the value of including a set of events to be the negative of the cost of the most important event excluded. Therefore, if we partition the set of events X into two sets, the included set, E_I, and the omitted set, E_O, then the value of that partition is $v_E = -\max_{x \in E_O} w(x)$. For instance, Solution 3 of our example includes the events $\{T, E_S, E_E, V_S, V_E\}$ and omits events $\{K_S, K_E, R_S, R_E\}$. The most important excluded events are K_S and K_E, each with a cost of 4. Therefore, the value of this partition is -4. The value is negated because our global objective function will maximize value rather than minimize cost.

This choice of evaluation function provides the key benefit and key drawback of our approach. Notice that if we modify the partition just described by moving events R_S and R_E to the included set, the value of the partition does not change (knitting is still the most important excluded action) even though the new partition is clearly superior. In other words, our evaluation function is not guaranteed to identify a Pareto optimal set of events. Of course, other evaluation functions exist — for instance, to calculate the weighted sum of the included events, as is typically done in weighted CSPs (Schiex, Fargier, & Verfaillie 1995), or to apply the leximin operator when comparing solutions (Dubois, Fargier, & Prade 1996b). The reason that we choose the less informative metric is that it provides an advantage we exploit heavily in our algorithm: we do not need to explore all possible combinations of included events — if some event with a weight w is excluded, then all events with weights equal or lower than w will be excluded as well. One side-effect of this requirement is that we can link together events that must co-exist by assigning them the same weight. For example, if the start of the knitting action is included, then the end of knitting should be included as well.

Using this property, we define what constitutes a "legal" set of events to include in a given problem: a partition of events into an included set E_I and omitted set E_O is legal iff there exists an event x_o with weight $w(x_o)$ such that all events in E_I have weight greater than $w(x_o)$, and all members in E_O have weight less than or equal to $w(x_o)$. Again, the value of such a partition is $v_E = -w(x_o)$. This approach bears resemblence to (Dubois, Fargier, & Prade 1996a), where all constraints less than a given threshold are ignored, while those above the threshold are enforced as hard constraints. Although this requirement seems strict, it makes sense for many practical situations; the events in plans, for example, often have dependencies in which the execution of one event is required to enable the execution of other events.

To determine the value of an induced DTPP for a partition, we find the optimal solution value for the DTPP using the DTPP_Maximin algorithm, which is described in the next section. A solution of quality v_P for a DTPP is one in which the output of each preference function has value v_P or greater. DTPP_Maximin finds a solution that maximizes v_P.

Finally, to evaluate the overall plan value, we use a weighted sum of the plan capacity and the plan quality, $w_P v_P + w_E v_E$. The setting of the parameters w_P and w_E depends largely on the relative magnitude of the event and temporal preference value functions, and can be chosen based on the desired trade-off between plan capacity and plan quality. We choose this function for simplicity; any continuous function of the two elements can be used, as long as the function is nondecreasing in the parameters v_P and v_E.

Now that the objective criterion has been defined, we present a simple algorithm, called Grow_DTPP, to find the optimal combination of the value of included events and the value of the corresponding induced DTPP. The basic idea is to iteratively build an induced DTPP one event level at a time, finding the value of its optimal solution during each iteration. The algorithm is given in Figure 1.

As input, the algorithm accepts a possibly over-constrained DTPP, T, a set of weights, W, and a global evaluation function f_G. We initialize a partition of the set of events by setting the included set E_I to be empty and an omitted set E_O containing all events, sorted by weight. The induced DTPP used in the algorithm is denoted by $T(E_I)$, since it is

Algorithm: Grow_DTPP
Input:
$T = \langle E, C \rangle$: A DTPP with $|E|$ events and $|C|$ constraints
$W = \{w(x) : x \in E\}$ A mapping from events to weights.
$f_G : \{v_E, v_P\} \rightarrow \Re$: The global value function.
Output:
An event set and component STP that maximizes f_G.
Initialization:
Let $E_I \leftarrow \emptyset$; // The initial included event set.
Let $E_O \leftarrow E$; // The initial omitted event set sorted w.r.t w.
Let *bestSolution* $\leftarrow \emptyset$;
Let *bestValue* $\leftarrow -\infty$;
Algorithm:
1. while($E_O \neq \emptyset$)
2. newEvents $\leftarrow \{x_o \in E_O : w(x_o) = \max_{x \in E_O} w(x)\}$
3. Let $E_I \leftarrow E_I \cup$ newEvents, $E_O \leftarrow E_O \setminus$ newEvents
4. Let $v_E \leftarrow -\max_{x \in E_O} w(x)$;
5. *currentSolution* $\leftarrow$ **DTPP_Maximin**($T(E_I)$);
6. If currentSolution is null, return bestSolution;
7. Let $v_P \leftarrow$ valueOf(currentSolution); // Get induced DTPP value
8. Let $v_G \leftarrow f_G(v_E, v_P)$;
9. If ($v_G >$ *bestValue*)
10. *bestValue* $\leftarrow v_G$;
11. *bestSolution* $\leftarrow \{E_I,$ *currentSolution*$\}$;
12. end
13. end
14. return bestSolution;

Figure 1: Grow_DTPP: A basic algorithm for solving over-constrained DTPPs.

a subset of T determined by the included events.

The algorithm begins by moving the most heavily weighted events from E_O to E_I (Lines 2 and 3). This step increases the value of the event partition (calculated in Line 4) but further constrains the induced DTPP because new constraints may be added. Next we use the DTPP algorithm DTPP_Maximin (Line 5) to obtain an optimal solution to the induced DTPP $T(E_I)$. If no solution exists, we can safely exit (Line 6), since the remaining steps can only add constraints. Otherwise, the value of this solution is assigned to v_P (Line 7), and the global value is computed (Line 8). If the global value exceeds the value of the current best solution, the best value and solution are updated (Lines 9-12).

Then, the next highest events in E_O are moved to E_I, and the process repeats until all events have been moved into the included set or until the induced DTPP does not have a solution.

The algorithm as presented uses the DTPP_Maximin algorithm as a black box when solving the induced DTPP. This strategy is needlessly wasteful from a constraint satisfaction point of view, since all information learned during search is lost between iterations. Later, we define pruning techniques that store small amounts of information learned in early iterations to reduce search in later iterations. Understanding these techniques, however, first requires a basic understanding of how the DTPP_Maximin algorithm works.

Finding maximin optimal solutions to DTPPs

In this section we describe our two-phase algorithm for finding a set of maximin optimal solutions to DTPPs with unrestricted preference functions (Peintner & Pollack 2004) — the call on Line 3 of the Grow_DTPP algorithm in Figure 1. DTPP_Maximin will be called once for every candidate set of included events chosen by the main loop.

The goal of DTPP_Maximin is to find some component STP of the DTPP that allows assignments with maximum values; the value of an assignment is determined by the disjunct whose preference function produces the minimal value for that assignment. An assignment with maximum value is called a maximin-optimal assignment. DTPP_Maximin searches the space of component STPs to find one that contains an optimal assignment.

Phase 1 of DTPP_Maximin

The first phase of the algorithm reduces the DTPP to a set of hard DTPs, which will be searched for solutions in the second phase. Each DTP in the set represents the DTPP *projected* at a particular preference level. Intuitively, a hard DTP projected at level i consists of only the parts of the DTPP that could participate in a solution with a maximin value of i or greater. Therefore, if a solution of maximin value i exists in the DTP, it can be found by searching the DTP projected at level i. The maximin optimal solution of the DTPP can be found by searching the individual projected DTPs: the highest-level projected DTP for which a solution can be found is the maximin optimal value.

The idea of projecting hard constraints from constraints with preferences is not specific to DTPPs. The idea was introduced for an algorithm that found maximin-optimal (Weakest Link Optimal) solutions to STPs with Preferences (Khatib *et al.* 2001).

The output of this phase is an organized collection of STP constraints called a *preference projection*.

Definition 1 *A* **preference projection** *relative to a* preference value set A *for a DTPP* $T = \langle X, C \rangle$*, is a set of intervals,* $\{p^a_{ck} | a \in A, c \in C, k$ *is a disjunct in* $c\}$*, where* $p^a_{ck} = \{x | f_{ck}(x) \geq a\}$ *and* f_{ck} *is the preference function for disjunct* k *in* c*. A preference value set is an ordered set of positive real numbers.*

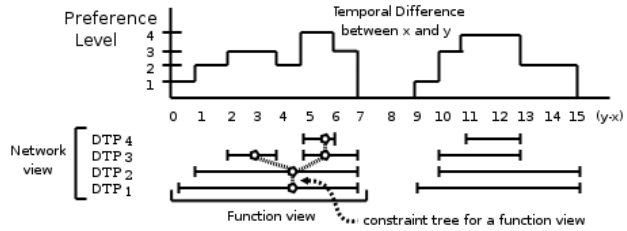

Figure 2: A DTPP constraint containing two disjuncts and its preference projection relative to the set $\{1, 2, 3, 4\}$. The circles and dashed lines define a tree within the function view of the first disjunct. Each interval in the tree is the child of the interval below it.

Each interval in the preference projection represents a hard constraint over the time-points in the disjunct from which it was projected. The intervals can be viewed in two ways: as a collection of disjuncts for every preference level or as a collection of disjuncts for every constraint. The *network view* groups the hard constraints by preference level, resulting in a DTP for each preference level. For example, the network view for a preference projection is a set of $|A|$ DTPs, where $\text{DTP}_i = \{p^i_{ck} | c \in C, k \in c\}$ is the hard DTP projected at preference level i.

The *function view* groups the hard constraints by preference function and represents all constraints originating from a single preference function. $P_c = \{p^a_{ck} | a \in A, k \in c\}$ is the function view for a particular disjunct k of constraint c. We often organize the constraints in a function view into a tree, where a hard constraint at one level is the parent of all hard constraints in the level above it. All standard parent/child relationships exist for this tree, with a parent always one level lower than its children. A child in this tree is always *tighter* than its parent; meaning, the set of temporal difference values allowed by the child is a subset of those allowed by its parent. We can define the parent relation for entire STPs as well: STP_1 is a child of STP_2 if each constraint in STP_2 is a parent to some constraint in STP_1; an STP is always tighter than its parent. The *ancestor* and *descendant* relations among STP constraints and STPs have obvious meanings.

Figure 2 shows an example of a single DTPP constraint and its preference projection relative to the preference value set $\{1, 2, 3, 4\}$. It shows the network view, which consists of four DTPs[1], and the function view for the first disjunct. The tree formed from the intervals in the function view is important for understanding the second phase.

The first phase of DTPP_Maximin is illustrated in the left half of Figure 3, where a single DTPP is projected into $|A|$ hard DTPs.

Phase 2 of DTPP_Maximin

The second phase attempts to solve the set of DTPs by searching for a consistent component STP within them. The phase starts by searching the bottom-level DTP until a consistent component STP is found. Then the search moves up to the DTP at the next level. The goal is to find the DTP with highest preference value that contains a consistent component STP.

One property of the preference projection that leads to an efficient algorithm is called the *upward inconsistency* property; it says that each descendant of an inconsistent component STP is inconsistent as well. This property holds because each STP is tighter than any of its ancestors at lower levels. If no solution exists in one STP, no tighter version of that STP can contain a solution either.

Property 1 Upward Inconsistency. *If a component STP of DTP_q is inconsistent, then any of its descendant component STPs in DTP_p for $p > q$ will also be inconsistent.*

[1]In this case, each DTP has only one constraint.

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 while searching the hard DTP at preference level 1, then it can prune away all descendant STPs at level 2 and above. Therefore, during the search through all projected DTPs, we can avoid checking many of the component STPs for consistency by remembering which component STPs at lower levels are inconsistent. "Remembering" in this case does not require additional storage; rather, it simply requires a jump ahead in the search space. We do not present the details of this search here (see (Peintner & Pollack 2004)). Instead, we sketch the basic steps:

1. Using any standard meta-CSP-based DTP solver (e.g. (Tsamardinos & Pollack 2003)), start searching the hard DTP projected at the lowest preference level.
2. If a consistent component STP S is found, store it as "best" and begin searching the hard DTP for the next highest preference level.
3. When moving to a DTP at a higher level, start the search at the first child of S, skipping all earlier component STPs.
4. When a solution at the highest DTP has been found or some other DTP has been fully searched, return "best".

Using a careful ordering of the search space, Step 3 of this algorithm sketch can be guaranteed to only skip component STPs that are descendants of STPs found to be inconsistent at lower levels (using the upward inconsistency property). The right half of Figure 3 shows a sample Phase 2 search trajectory that illustrates the use of the upward inconsistency property: component STP^1_3 was skipped in the search because its parent STP^1_2 was found to be inconsistent.

Pruning Techniques

The basic Grow_DTPP algorithm can be significantly enhanced with three pruning techniques that use information learned while solving the induced DTPP in one iteration to reduce search in later iterations.

Each of the three pruning techniques depends in some way on a basic property that relates induced DTPPs at different iterations: each induced DTPP is at least as constrained as those at previous iterations. This property is ensured because each iteration adds at least one event that may cause the addition of more constraints. If no constraints are added, the induced DTPP remains unchanged; if one or more constraints are added, the problem becomes more constrained.

Global Upper Bound

The first pruning technique uses the basic property described above to calculate an upper bound for the global objective function in the Grow_DTPP algorithm.

Since we know that the induced DTPP tightens after each iteration, we also know that the value of optimal solutions to the series of induced DTPPs will monotonically decrease. For example, if the induced DTPP at iteration i, T_i, has optimal value v^i_P, we know there is no solution to T_i at level $v^i_P + 1$. Since the difference between T_i and T_{i+1} is a set of

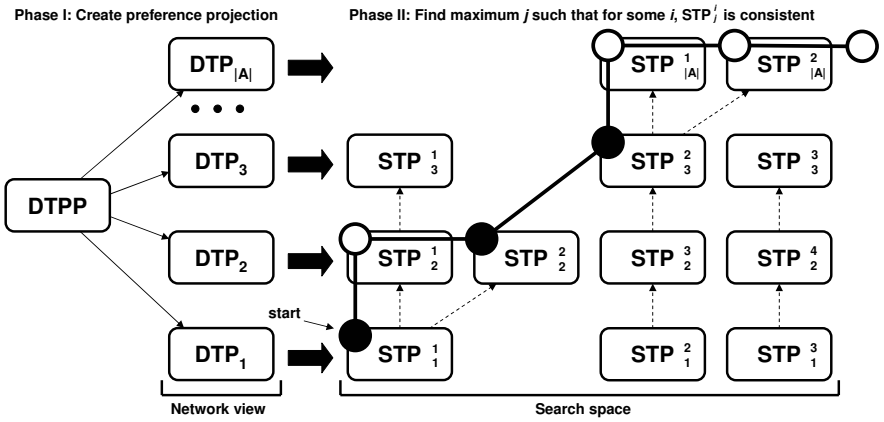

Figure 3: The two phases of solving a DTPP. Phase 1 projects the DTPP onto $|A|$ DTPs, each consisting of many component STPs. Phase 2 searches through the space of component STPs, starting at the bottom level and moving up one level each time a consistent STP is found. The dotted lines between STPs denote a parent/child relationship. A sample search trajectory is shown, with filled in circles representing consistent STPs and empty circles representing inconsistent STPs. The maximin optimal solution is STP^2_3.

additional constraints, then we know that T_{i+1} does not have a solution at level v_P^i+1 either. Therefore, $v_P^i \geq v_P^j : i < j$.

Given this knowledge, we can calculate the upper bound for the global function at iteration $i+1$ to be $f_G(0, v_i)$, where 0 is the value of including all events and v_i is the value of the optimal solution to T_i. We can use this upper bound to determine whether early termination of the algorithm is warranted. For this, we make three changes to the Grow_DTPP algorithm. First, we initialize a *globalUpperBound* variable to an infinite value. Second, we calculate the upper bound by adding the following statement between lines 12 and 13: $globalUpperBound = f_G(0, v_P)$. Finally, we change Line 1 to force termination if the upper bound is not greater than the best value found so far. If the upper bound is not greater, no improvement is possible, so further search is unnecessary.

DTPP Bounds

Where the first pruning technique computes an upper bound for the global value, the second calculates an upper and lower bound that is used by DTPP_Maximin. We already discussed that the value of the induced DTPP solution for iteration i is the upper bound for the value of the induced DTPP solution in iteration $i + 1$. Therefore, we can restrict DTPP_Maximin from searching any projected DTPs at preference levels greater than the upper bound. Thus, if the optimal value for both iterations i and $i + 1$ is k, then the solver can terminate as soon the level k solution is found during iteration $i + 1$. If it did not recognize the upper bound, the solver would have to fully search level $k + 1$.

We can also compute a lower bound for DTPP_Maximin using the global optimality function and the value of the current best solution. Consider a case in which the algorithm has just completed iteration 2, and assume the objective functions produced the values: $v_P = 5$, $v_E = -4$, and $v_G = 1$ (both weights in f_G are 1). Assume that this is currently the best solution. Now, when the third iteration begins, a new event is moved to the include set E_I, increasing the v_E value to -2. When solving the new induced DTPP, a v_P value of 3 or less would not be useful, since the new v_G would not exceed 1, the global value of the best solution. Therefore, a lower bound of 4 can be passed in to the DTPP_Maximin algorithm, allowing it to avoid searching levels 1-3. More generally, the lower bound indicates that any solution to the DTPP with a value less than the lower bound will not be useful, so we can jump directly to the projected DTP whose preference level matches the lower bound in Step 1 of the algorithm sketch.

To compute the lower bound, the global evaluation function must be reversible. For the weighted sum function this is simple: $min\_v_P = (v_G - w_E v_E)/w_P$, where v_G is set to 1 greater than the value of the best solution.

Component STP Skip

Since each induced DTPP is tighter than previous ones, there exists a property that is analogous to the upward inconsistency property: the *persistent inconsistency* property. This property says that any component STP found to be inconsistent in one iteration will remain inconsistent in all later iterations. Using this property, parts of the search space can be pruned on subsequent iterations. Specifically, when searching an induced DTPP, we skip (partial) component STPs that have already been checked.

To implement this skip during the DTPP_Maximin algorithm, we store at each level the component STP that was found to be consistent — the point in the search space where the algorithm moves up to the next preference level. All earlier points are known to be inconsistent.

It is not immediately apparent how storing the search points in one DTPP can help solve another — after all, they contain different constraints. We know, however, that the only difference between an induced DTPP and the one that follows it is a set of newly added constraints. All constraints

in the first still exist unchanged in the second. Therefore, if we simply append the new constraints to the end of the constraint list, the two search spaces will become aligned. As a result, the search trajectory for the first induced DTPP matches the search trajectory for the second — at least up to the stored search point.

The result of the pruning achieved by the upward inconsistency and persistent inconsistency properties is that no inconsistent component STP is ever checked twice. The only component STPs that are checked twice are those found to be consistent during some iteration. The maximum number of iterations in the algorithm is the number of events in the algorithm, $|E|$. The number of consistent component STPs per iteration is bounded by the number of preference levels ($|A|$), since the algorithm moves up one level each time a consistent component STP is found. Therefore, we can bound the complexity of the total search for all calls to DTPP_Maximin to be the sum of (1) the complexity of searching all projected DTPs of the largest induced DTPP tried and (2) $|A| * |E|$ STP consistency checks.

Although this pruning technique is possibly the least powerful (savings average around 10% when applied alone), its analysis tells us that the cost of incrementally growing the DTPP is small: the complexity of the algorithm depends more on the size of the largest DTPP tried than on the number of iterations. In practice, when this technique is combined with the previous two pruning techniques, the time required to iteratively build up a large induced DTPP can be much less than directly solving the induced DTPP for the final iteration.

Experimental Results

We now present empirical results that show (1) the effectiveness of our pruning strategies during the DTPP_Maximin search, (2) how our algorithm compares to an "all-knowing" oracle, and (3) how the algorithm behaves as the number of specified events increases.

Since the goal of Grow_DTPP is to find a set of events that maximize the global objective function, we can determine a lower bound for the run-time of Grow_DTPP by timing DTPP_Maximin on the optimal set of events. In other words, we pretend an oracle delivers the optimal event set in advance, and use DTPP_Maximin to find the solution of the induced DTPP. We refer to this algorithm as *Oracle*.

We report a comparison of Oracle and two versions of the Grow_DTPP algorithm: one with all three pruning strategies, called *With Pruning*; and one with only the first pruning strategy, called *Standard*. We use the Global Upper Bound technique in both algorithms because it functions as a terminiation condition that, if not used, would greatly skew the results. We also show how the run-time of *With Pruning* depends not on the number of events in the original problem, but only on the number of events that "fit" in the DTPP.

Setup

We generated random over-constrained DTPPs using a set of parameters shown in Table 4.

Generating random DTPPs We begin the process of generating a DTPP by creating a set of events and assigning a random weight to each. The number of events is governed by *numEvents*, while the weight is determined by *maxEventWeight*. We fixed *maxEventWeight* at *numEvents*/2, so that on average 2 events share each weight.

Next, we create *numConstraints* constraints. To create a constraint, we first create *numDisjuncts* disjuncts, choosing the events randomly for each, and choosing the upper and lower bounds using values between *minDomain* and *maxDomain*. The result is a hard DTP constraint.

Next, we attach a preference function to each disjunct using the last three parameters in Figure 4. Rather than defining the function for each time difference directly, we instead define the interval(s) that exist at each level, i.e., the function view of the preference projection for each disjunct.

We begin by defining the interval for the lowest level to be the hard bounds of the corresponding disjunct. To determine the interval width of the child constraint at the next level, a random value from the real interval [*reductionFactorLB*, *reductionFactorUB*] is multiplied by the width of interval at the current level. The interval's lower bound is selected by adding a random value from the interval [0, previous width − new width] to the lower bound of the parent's interval.

For example, consider a disjunct whose bounds form the interval $[-3, 12]$; we set the lowest preference level to be this interval. Let the *reductionFactorLB* = .5 and the *reductionFactorUB* = 7. If the randomly chosen reduction factor is 0.6, then the width of the new interval will be 0.6*15 = 9. The new interval's start bound could be placed anywhere in the interval [-3, 3]. We can choose a value in this interval by choosing from the interval $[0, 15 - 9]$ and adding it to -3.

This process continues for each level until the maximum number of levels has been reached or the constraint width at some level reduces to 0. The result is a semi-convex function[2] as in the second disjunct of Figure 2.

Space of DTPPs tested Given the number and range of parameters, it is difficult to sufficiently cover the range of possible DTPPs types. For this experiment, we fix each parameter to a default value and vary one parameter at a time. The defaults are shown in the left column of Table 4.

In addition to varying the parameters of the DTPP, we varied the weights for the event set value w_E and the preference value w_P to see the effect of different global optimality functions on run-time.

We ran four tests, each of which varied the following parameters independently:

Preference function shape The preference function shape determines how many preference levels exist. We tried preference functions that ranged from very shallow slopes (*reductionFactorLB* = [.3,.4]) to very steep slopes (*reductionFactorLB* = [.8,.99]), and functions with highly varied slopes (*reductionFactorLB* = [.5,.99]).

Constraint Density We varied the number of constraints (i.e. the amount by which the problem is over-const-

[2] A semi-convex function is one in which, for all Y, the set $\{X : f(X) \geq Y\}$ forms an interval (Khatib *et al.* 2001).

20	numEvents	The number of events in the DTPP
10	maxEventWeight	The maximum weight that an event can have. Minimum is 0.
40	numConstraints	The number of dual bounded constraints
2	numDisjuncts	The number of disjuncts per constraint
-100	minDomain	Minimum value of any STP interval
500	maxDomain	Maximum value of any STP interval
15	numLevels	The maximum number of preference levels in a constraint
.5	reductionFactorLB	Minimum fraction of an STP interval width at level i that will exist at level $i+1$
.9	reductionFactorUB	Maximum fraction of an STP interval width at level i that will exist at level $i+1$

Table 4: Parameters used to generate random DTPPs and their default values.

rained), using values 20, 30, 40, 50, 60, and 70, which is 1 to 3.5 times the number of events[3].

Weight ratio The ratio of the event set value weight (w_E) and the preference value weight (w_P) was varied using values 15, 10, 5, 2, and 1. We do not report on ratios less than 1 because they tend to create solutions with few events and have run times similar to problems of ratio 1.

Problem size invariance For three different problem sizes, we showed that the running time for problems whose solutions had equal number of events was relatively constant. The three problem sizes were 20 events, 30 events, and 40 events. The constraint density was fixed at 2.

The first three tests averaged the results from 500 randomly generated DTPPs, while the data for the fourth test was extracted from 3000 DTPPs. To analyze the results from the fourth test, we calculated the average running time for each problem size/solution size pair. For example, we found all instances in which problems with 30 specified events had 15 events in their solution and averaged their run times. The intent is to show that the average running times for problems with m events in their solution is not dependent on whether the original problem contained 20, 30 or 40 events.

The algorithms were implemented in Java and the tests were run in a WindowsXP OS using a 3GHz hyper-threaded Pentium 4 processor. Hyper-threading limits the CPU usage of any single process to 50%, so the full speed of the processor was not utilized. Running time for the algorithms was measured using a library that counts only the amount of time the main thread is running, so garbage collection and other system threads do not affect the run-time reported.

Results

We first show the comparison of run-times for *Oracle*, *With Pruning*, and *Standard*. Figure 4 shows the run times for each algorithm on the first three tests (Preference function shape, Constraint Density, and Weight ratio) and for only *With Pruning* on the fourth test (Problem size invariance).

Graphs (a), (b), and (c) in Figure 4 show that the pruning techniques cut run-time regardless of preference function slope, constraint density, or weight ratio. On average, pruning cut run-time by 45-60%, a significant savings. Other tests not shown attributed about a quarter of this savings to the Component STP Skip technique and the rest to the DTPP Bound technique. (Remember that we used the Global Upper Bound technique in all tests.) *Oracle* clearly out-performs the other two algorithms, which is not surprising since the Grow_DTPP algorithms usually solves problems that contain more than the optimal number of events.

We notice in the test for varying weight ratios (Graph (c)) that the run-time decreases as the value of the induced DTPP is weighted more heavily. When the w_P value is higher than w_E (i.e. the ratio is < 1), few iterations occur in the algorithm because the best solutions are always the ones with few events and high valued solutions.

The most important result is found in Graph (d). Recall that our goal is to allow the KE to add as many events and constraints as desired without concern for how it affects the network. It is crucial therefore that the run-time does not increase substantially as the number of specified events increases. Graph (d) shows that we have achieved this goal. For example, for all problems whose solutions contained 25 events, the average running time for problems with 30 specified events equaled the average running time for those with 40 specified events. For other solution sizes, the difference between the three curves are very small.

Conclusion

In this paper, we have described an approach for solving possibly over-constrained DTPs with Preferences. Our goal was to simplify the task for the KE by addressing two limitations of TCSPs and their associated algorithms: the inability to reason about optional events, and the trade-off inherent in defining hard constraints. Our approach allows the KE to prioritize events and to express preferences when defining constraints so that the KE can focus on accurately defining local information without regard to the problem as a whole.

Our method solves an over-constrained DTPP by searching through the space of induced DTPPs, which are DTPPs that include only a subset of the events in the original problem. The method incrementally builds an induced DTPP and uses a known DTPP algorithm to find the value of its optimal solution. Optimality is defined using three objective functions: one that defines the value of a set of included events, one that defines the value of a DTPP induced by the included events, and a third that combines the results of the first two into an overall score. The main contribution is a

[3]The ratios are small compared to other reports because the DTP constraints we describe are dual-bounded, rather than the more common single-bounded variety. To represent a DTP constraint with k dual-bounded disjuncts requires 2^k single-bounded disjuncts.

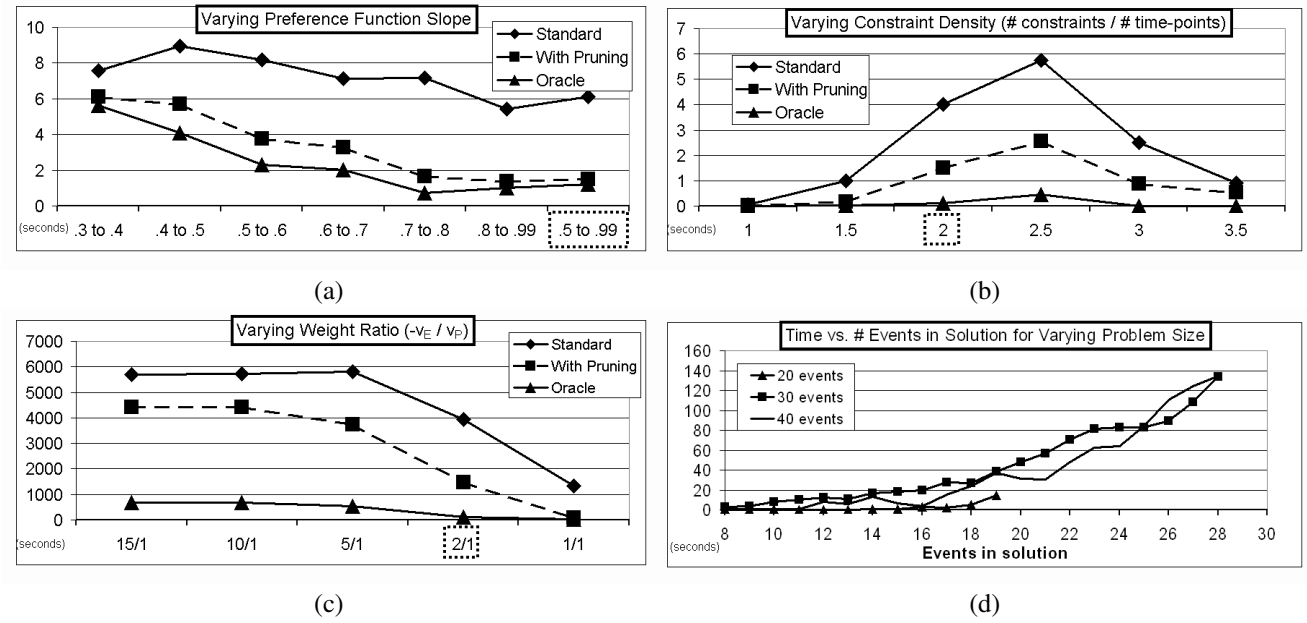

Figure 4: Run time in seconds for each of the four tests. The dotted boxes denote the data points that correspond to our default parameters. Graph (a) shows the run-times for different reduction factor bounds: lower bounds produce shallow slope functions with few preference levels. Graph (b) shows run-times for different constraint densities. Graph (c) shows the run-times for different $-v_E/v_P$ ratios. Graph (d) shows the run-times for problems whose solutions had the same number of events.

set of three pruning techniques that were empirically shown to dramatically reduce the time required to find the optimal solution. The key result is that solving times depend on the size of the solution, not on the size of the original problem.

We believe the main deficiency in this approach is the use of the maximin optimality function for both the event sets and the induced DTPPs. A utilitarian optimality function (maximizing the sum of the values of all elements) would be a better choice for our application. We are currently working on algorithms that will efficiently find high-quality utilitarian solutions to DTPPs.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research, under Contract No. FA9550-04-1-0043 and the Defense Advanced Research Projects Agency (DARPA), through the Dept. of the Interior, NBC, Acquisition Services Division, under Contract No. NBCH-D-03-0010. Any opinions, findings and conclusions or recommendations do not necessarily reflect the view of the United States Air Force, DARPA, or the Department of Interior-National Business Center.

References

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

Dubois, D.; Fargier, H.; and Prade, H. 1996a. Possibility theory in constraint satisfaction problems: handling priority, preference and uncertainty. *Applied Intelligence* 6:287–309.

Dubois, D.; Fargier, H.; and Prade, H. 1996b. Refinements of the maximin approach to decision-making in fuzzy environment. *Fuzzy Sets and Systems* 81:103–122.

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

Moffitt, M. D., and Pollack, M. E. 2005. Partial constraint satisfaction of disjunctive temporal problems. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1-2):5–47.

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*, 723–728.

Pollack, M. E.; McCarthy, C. E.; Ramakrishnan, S.; Tsamardinos, I.; Brown, L.; Carrion, S.; Colbry, D.; Orosz, C.; and Peintner, B. 2002. Autominder: A planning, monitoring, and reminding assistive agent. In *Proceedings of the 7th International Conf. on Intelligent Autonomous Systems* 7.

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.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

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.