Resolving Uncontrollable Conditional Temporal Problems using Continuous Relaxations

Peng Yu and Cheng Fang and Brian Williams
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge, MA 02139
{yupeng,cfang,williams}@mit.edu

Abstract

Uncertainty is commonly encountered in temporal scheduling and planning problems, and can often lead to over-constrained situations. Previous relaxation algorithms for over-constrained temporal problems only work with requirement constraints, whose outcomes can be controlled by the agents. When applied to uncontrollable durations, these algorithms may only satisfy a subset of the random outcomes and hence their relaxations may fail during execution. In this paper, we present a new relaxation algorithm, Conflict-Directed Relaxation with Uncertainty (CDRU), which generates relaxations that restore the controllability of conditional temporal problems with uncontrollable durations. CDRU extends the Best-first Conflict-Directed Relaxation (BCDR) algorithm to uncontrollable temporal problems. It generalizes the conflict-learning process to extract conflicts from strong and dynamic controllability checking algorithms, and resolves the conflicts by both relaxing constraints and tightening uncontrollable durations. Empirical test results on a range of trip scheduling problems show that CDRU is efficient in resolving large scale uncontrollable problems: computing strongly controllable relaxations takes the same order of magnitude in time compared to consistent relaxations that do not account for uncontrollable durations. While computing dynamically controllable relaxations takes two orders of magnitude more time, it provides significant improvements in solution quality when compared to strongly controllable relaxations.

Introduction

Every day, as individuals we miss meetings and deadlines, because we try to do too much, and do not estimate time accurately. These situations can lead to anywhere from a major annoyance, such as missing a conference deadline, to a major catastrophe. These situations can be better handled with decision aids that help individuals to estimate how much time is required in order to compensate for uncertainty, and by providing advice on which goals should be dropped, to achieve a manageable set of goals.

Such over-subscribed situations have often been modeled by *over-constrained temporal problems*. A temporal problem is over-constrained if no schedule (Dechter *et al.* 1991), or execution strategy (Vidal and Fargier 1999) for problems with uncertain durations, can be found that satisfies all its constraints. To solve an over-constrained temporal problem, one has to identify its conflicting constraints and resolve them by relaxing one or more constraints, such that the consistency or controllability of the problem can be restored.

Several methods have been developed to solve over-constrained temporal problems. In (Beaumont *et al.* 2001), partial constraint satisfaction techniques were applied to find a subset of satisfiable constraints. Later, (Moffitt and Pollack 2005; Peintner *et al.* 2005) extended the resolution capability to temporal problems with disjunctions and preferences. In (Yu and Williams 2013), a conflict-directed approach was introduced to resolve conditional temporal problems by continuously relaxing constraints, instead of suspending them completely. However, all prior work focused on temporal problems with only controllable durations. When applied to problems with uncertain durations, their relaxations may fail since they only satisfy a subset of the possible times for the uncontrollable durations.

In this paper, we present our approach for resolving over-constrained conditional temporal problems with uncertain durations, the Conflict-Directed Relaxation with Uncertainty algorithm (CDRU), to address this issue. CDRU enumerates preferred continuous relaxations that restore controllability using a conflict-directed approach. It learns the cause of failure by generalizing the conflict-learning technique in (Williams and Ragno 2002; Yu and Williams 2013) to include strong and dynamic controllability checking and conflict extraction algorithms. CDRU then resolves the conflicts by relaxing constraints and tightening uncontrollable durations.

The CDRU algorithm has been incorporated in a mission plan advisory system and demonstrated for assisting oceanographers to schedule activities in deep-sea exploration expeditions. Its applications also include managing trip plans for users of car-sharing networks. In the following sections, we will use an example from this domain to illustrate the concepts and techniques in the CDRU algorithm.

Motivating Example

To motivate the need for resolving over-constrained temporal problems with uncontrollable durations, and to demon-

strate the capabilities of the CDRU algorithm, we describe an example in the domain of commute trip planning. Planning a daily commute trip may become difficult for a person when there are multiple time constraints, such as deadlines or reservations. The person has to compare different options and adjust the durations of their activities carefully to meet all scheduling constraints. The problem gets even more complex if there is a large uncertainty in the duration of an activity: checking all possible outcomes of the uncertain duration is usually beyond a person's capability.

Consider the following example on John's weekend trip for grocery shopping and lunch. He is planning to leave home at 10am and return before 1pm. John reserved a Zipcar, an hourly-based car-sharing service (Zipcar 2013), for his trip. There are two grocery stores (A and B) and two restaurants (X and Y) nearby. John has a preference over the stores and restaurants. In addition, the preferred stay at each location varies, and the reservation times John can get at two restaurants are different. Finally, driving times between these locations are different too, and there are uncertainties in the estimation of driving durations.

We introduce the Controllable Conditional Temporal Problem with Uncertainty (CCTPU) formalism to model John's trip. It is an extension to the CCTP formulation in (Yu and Williams 2013) with the addition of uncertain durations. The solution to such a problem can be viewed as a strategy for John's trip that includes: which store to visit, which restaurant to dine at, and how much time to spend at each of them. A CCTPU contains two types of components: decision variables and temporal constraints. We define two variables for the decisions that John needs to make: $Store$ and $Lunch$. There are two values in the domain of variable $Store$: A (100) and B (200). Each domain value represents a decision of going to one grocery store, and is associated with a positive reward value that represents John's preference toward it. The other variable, $Lunch$, has two domain values: X (200) and Y (100).

Table 1: Events in John's trip

Trip starts:	S_T	Restaurant X arrive/leave:	X_A,X_L
Trip ends:	R_T	Restaurant Y arrive/leave:	Y_A,Y_L
Store A arrive/leave:	A_A,A_L	Store B arrive/leave:	B_A,B_L

Next, we define events in John's trip, which are designated time points that represent different states. There are ten events in the CCTPU (Table 1): an event that represents the beginning of the trip (S_T); an event for the end of the trip (R_T); and pairs of events that represent the arrival and departure of each location (Store A, B; Restaurant X, Y).

Table 2 shows all the constraints in the CCTPU. They encode the durations of driving, shopping and dining activities in John's trip, as well as his time requirements. Constraints C_1 through C_4 encode John's desired length of stay at each location. Constraints C_5 through C_{12} are highlighted in bold: they are uncontrollable simple temporal constraints that encode the driving times between locations. Their temporal bounds indicate the domain of the random outcomes. Constraints C_{13} and C_{14} represent the lunch reservations that John can get at restaurant X and Y. He has to arrive at the restaurant within the reservation window in order to

Table 2: Temporal constraints in John's trip (in minutes)

C_1(R):A_L-$A_A \in [50, 60]$	$\mathbf{C_5}$:A_A-$S_T \in [\mathbf{45}, \mathbf{65}]$		Store←A
C_2(R):B_L-$B_A \in [45, 60]$	$\mathbf{C_6}$:B_A-$S_T \in [\mathbf{30}, \mathbf{50}]$		Store←B
C_3(R):X_L-$X_A \geq 60$	$\mathbf{C_7}$:R_T-$X_L \in [\mathbf{28}, \mathbf{35}]$		Lunch←X
C_4(R):Y_L-$Y_A \geq 65$	$\mathbf{C_8}$:R_T-$Y_L \in [\mathbf{30}, \mathbf{32}]$		Lunch←Y
$\mathbf{C_9}$	X_A-$A_L[\mathbf{51}, \mathbf{54}]$	Store←A and Lunch←X	
$\mathbf{C_{10}}$	X_A-$B_L[\mathbf{22}, \mathbf{24}]$	Store←B and Lunch←X	
$\mathbf{C_{11}}$	Y_A-$A_L[\mathbf{42}, \mathbf{45}]$	Store←A and Lunch←Y	
$\mathbf{C_{12}}$	Y_A-$B_L[\mathbf{21}, \mathbf{25}]$	Store←B and Lunch←Y	
C_{13}	X_A-$S_T[105, 120]$	Lunch←X	
C_{14}	Y_A-$S_T[90, 105]$	Lunch←Y	
C_{15}(R)	R_T-$S_T \in [0, 180]$		

secure a table. Finally, constraint C_{15} encodes the length of John's Zipcar reservation.

Note that some of the constraints are conditional and labeled by one or more assignments. A conditional constraint is activated if and only if the assignments in its label hold. For example, constraints C_1 and C_5 encode the driving and shopping time for store A. They are considered only if John chooses to shop at A. In addition, the constraints labeled with (R) are relaxable constraints (Yu and Williams 2013). Their lower or upper bounds can be relaxed at a specified cost, in order to restore the feasibility of the problem. In this problem, we assume that the gradients of the cost functions are 3/minute for shortening the shopping time (C_1, C_2), 2/minute for shortening the dining time (C_3, C_4), and 1/minute for extending the Zipcar reservation (C_{15}).

Without any relaxations, these is no solution that can satisfy all requirements in John's trip: the overall trip length is at least 193 minutes, which is 13 minutes more than John's Zipcar reservation. To resolve this over-constraint problem, we first apply the consistency-based BCDR algorithm. It returns Figure 1 as the most preferred relaxation for John.

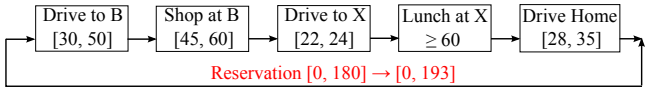

Figure 1: Consistent solution for John's trip

The utility of this solution is 387, which is computed by subtracting the cost of relaxing the reservation constraint (13) from the reward of going to store B and restaurant X (400). However, the solution does not account for the uncontrollable driving times: BCDR only considers the lowest driving times while computing the solution. As a result, the solution may fail during the trip, since it has no margin to absorb any delay in driving. Next, we present two solutions generated by CDRU, which are based on two execution strategies that take the uncertainty into consideration. The first strategy, called *Strong Controllability*, comes up with a schedule of activities before starting the plan, which ensures success for all uncontrolled durations. The second execution strategy, called *Dynamic Controllability*, instead observes these uncertain outcomes along the way, and makes 'more informed' decisions about scheduling each activity.

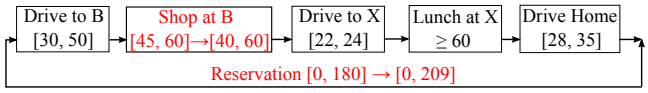

Figure 2: Strongly controllable solution for John's trip

The second solution is computed based on strong controllability (Figure 2). It extends the reservation to 209 minutes, and decrease the lower bound of the shopping time at B to account for the uncertainty in the driving between home and store B. This solution has a lower utility of 356, but enables a schedule that satisfies John's requirements and the uncertain driving durations.

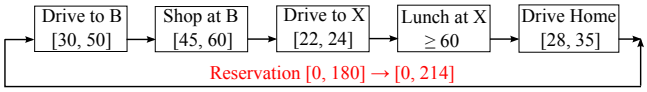

Figure 3: Dynamically controllable solution for John's trip

The third and final solution is computed based on dynamic controllability (Figure 3). Unlike the previous solution, it does not need to decrease the lower bound of shopping time at B to account for the uncertainty in driving, and hence is less conservative than the second solution, while still being safe. The solution has a higher utility of 366, and enables a dynamic schedule on the fly instead of a static schedule beforehand: the times of leaving store B and restaurant X depend on the actual driving times.

This example demonstrates the effectiveness of CDRU in resolving uncontrollable temporal problems. Compared to consistency-based relaxation algorithms, whose solutions may fail during execution, CDRU guarantees that its solutions satisfy all random outcomes of uncertain durations. Depending on the user's needs, CDRU can generate relaxations that enable either a schedule for all events, or a dynamic execution strategy.

Problem Statement

Simple Temporal Networks with Uncertainty (Vidal and Fargier 1999) have been widely used to model temporal problems with uncertain durations. It is an extension to the Simple Temporal Network formalism (Dechter *et al.* 1991) by adding a new class of constraints: *contingent constraints*. The duration of a contingent constraint is a random variable between its lower and upper bounds and cannot be freely assigned. The solution to a STNU is an execution strategy that satisfies all requirement constraints regardless of the outcomes of contingent constraints. The existence of such a strategy is characterized by the controllability, instead of consistency, of the STNU. Controllability concerns about the outcomes of contingent constraints and the time points when the outcomes become available to the agent.

There are three types of controllability (Vidal and Fargier 1999): *Strong*, *Dynamic* and *Weak*. Each type has a different assumption about the time when the outcomes of contingent constraints become available. In this paper, we focus on the first two types, strong and dynamic controllability, which assume that no outcome is known prior to the execution. Strong controllability requires a predetermined schedule which satisfies all constraints regardless of the outcomes of the uncertain durations, whereas dynamic controllability requires a policy for scheduling as observations of uncertain durations become available. Intuitively, dynamic controllability is more flexible as it make use of information gained during execution.

In addition, the STNU formalism has been extended with disjunctions and conditional constraints to handle more real-world scheduling and planning problems. In (Venable and Yorke-Smith 2005; Peintner *et al.* 2007), the *Disjunctive Temporal Problem with Uncertainty* (DTPU) formalism was introduced to permit non-convex and non-binary constraints. DTPU can be viewed as an extension to the deterministic Disjunctive Temporal Problem (DTP) (Stergiou and Koubarakis 1998; Tsamardinos and Pollack 2003) formalism with contingent constraints. It allows the expression of disjunctive constraints, and enables the agent to choose between alternatives.

In this paper, we define a new formalism that extends DTPUs to permit the description of more general trip planning and relaxation problems, the *Controllable Conditional Temporal Problem with Uncertainty* (CCTPU). The new formalism is also an extension to *Controllable Conditional Temporal Problem* (CCTP, (Yu and Williams 2013)) with the addition of contingent constraints. We first repeat the definition of CCTP here, then present the additions introduced by CCTPU.

Definition 1. *A CCTP is an 9-tuple* $\langle V, E, RE, L_v, L_p, P, Q, f_v, f_e \rangle$ *where:*

- P *is a set of controllable finite domain discrete variables;*
- Q *is the collection of domain assignments to* P*;*
- V *is a set of events representing designated time points;*
- E *is a set of temporal constraints between pairs of events* $v_i \in V$*;*
- $RE \subseteq E$ *is a set of relaxable temporal constraints whose bounds can be relaxed;*
- $L_v : V \rightarrow Q$ *is a function that attaches conjunctions of assignments to* P*,* $q_i \in Q$*, to some events* $v_i \in V$*;*
- $L_p : P \rightarrow Q$ *is a function that attaches conjunctions of assignments to* P*,* $q_i \in Q$*, to some variables* $p_i \in P$*;*
- $f_p : Q \rightarrow \mathcal{R}^+$ *is a function that maps each assignment to every controllable discrete variable,* $q_{ij} : p_i \leftarrow value_j$*, to a positive* ***reward****;*
- $f_e : (e_i, e_i') \rightarrow r \in \mathcal{R}^+$ *is a function that maps the relaxation to one relaxable temporal constraint* $e_i \in RE$*, from* e_i *to* e_i'*, to a positive* ***cost****.*

Definition 2. *A CCTPU contains all elements in a CCTP, plus* E_u *and* RE_u*, where:*

- $E_u \subseteq E$ *is a set of contingent constraints between pairs of events.* $E \backslash E_u$ *is the set of all requirement constraints;*
- $RE_u \subseteq E_u$ *is a set of relaxable contingent constraints whose bounds can be tightened, and* $RE_u \subseteq RE$*.*

The cost function is generalized to include both requirement and contingent constraints: $f_e : (e_i, e_i') \rightarrow r \in \mathcal{R}^+$ *is a function that maps the following to a* ***non-negative cost****.*

- *the* ***relaxation*** *to a requirement constraint,* $e_i \rightarrow e_i', e_i \in RE \backslash RE_u$*;*
- *or the* ***tightening*** *to a contingent constraint,* $e_i \rightarrow e_i', e_i \in RE_u$*;*

We generalize the concept of relaxations to include contingent constraints. To resolve a conflict by relaxing requirement constraints, we will either increase its upper bound or reduce its lower bound. On the other hand, we can shrink the uncertainty for contingent constraints in a conflict: we may resolve the conflict by increasing the lower bound or decreasing the upper bound of its contingent constraints. Usually, contingent constraints are used to reserve some flexibility for the agents or the environment in executing their tasks. A tighter duration means less flexibility for them, but also imposes less restriction on the solution to the temporal problem. We will give more insights into the relation between contingent constraints and conflict resolutions in the algorithm section.

The solution to a CCTPU is a 3-tuple $\langle A, R_e, R_u \rangle$, where:

- A is a complete set of assignments to some variables in P that leaves no variable unassigned.
- R_e is a set of relaxed bounds of some relaxable requirement constraints in $RE \backslash RE_u$.
- R_u is a set of tightened bounds of some relaxable contingent constraints in RE_u.

A feasible solution provides a grounded and controllable CCTPU. We separate the solutions into two categories: *strongly controllable* and *dynamically controllable*. This is based on the type of execution strategies a solution can enable.

- A strongly controllable solution makes the CCTPU strongly controllable. That is, the relaxation enables an execution strategy with a firm schedule for all events.
- A dynamically controllable solution makes the CCTPU dynamically controllable. Due to the flexibility of dynamic controllability relative to strong controllability, there is usually a greater solution space to explore.

Note that a strongly controllable solution is also a dynamically controllable solution, since strong controllability is more restrictive than dynamic controllability. In this paper, we are only concerned about controllable variables that are not dependent on observation events in CCTPUs. This makes it simpler to solve compared to the Conditional Temporal Problems with Uncertainty (CTPUs) (Hunsberger *et al.* 2012) in that those tasks may require the enumeration of all scenarios that are dependent on uncontrollable variables.

Approach

In this section, we present the Conflict-Directed Relaxation with Uncertainty algorithm that enumerates strongly and dynamically controllable relaxations for CCTPUs. CDRU can be viewed as an extension to the BCDR algorithm (Yu and Williams 2013). It extends the conflicts learning and resolution process to account for uncontrollable durations. To enumerate relaxations for CCTPUs, CDRU needs to extract conflicts from strong and dynamic controllability checking algorithms. The conflict could be a mixed set of requirement and contingent constraints. The resolution of conflicts involves both relaxing requirement constraints and tightening contingent constraints. We will first give an overview of the CDRU algorithm, and then discuss the conflict learning and resolution process in detail.

The CDRU Algorithm

CDRU uses a conflict-directed approach to enumerate candidate relaxations and prune infeasible search space. This approach was first introduced by Conflict-directed A* (Williams and Ragno 2002) on discrete domain variables and later extended to continuous variables and constraints by BCDR. The key of this conflict-directed approach is to explore the search space using two types of expansions: expand on unassigned variables and on unresolved conflicts. The first expansion guides the search into unexplored regions, and the second expansion keeps the search away from known infeasible regions in the search space.

CDRU can be implemented with different search orders, such as best-first, depth-first and branch&bound, to meet the needs of different applications. In this section, we will use best-first order to demonstrate the concepts and techniques in it. The pseudo code of CDRU is given in Algorithm 1.

Input: A CCTPU
$T = \langle V, E, E_u, RE, RE_u, L_v, L_p, P, Q, f_v, f_e \rangle$.
Output: A relaxation $\langle A, R_e, R_u \rangle$ that maximizes $f_v - f_e$.
Initialization:
1 $Cand \leftarrow \langle A, R_e, R_u, C_r, C_{cont} \rangle$; the first candidate;
2 $Q \leftarrow \{Cand\}$; a priority queue of candidates;
3 $C \leftarrow \{\}$; the set of all known conflicts to be checked;
4 $U \leftarrow V$; the list of unassigned controllable variables;
Algorithm:
5 **while** $Q \neq \emptyset$ **do**
6 $Cand \leftarrow$Dequeue(Q);
7 $currCFT \leftarrow$UNRESOLVEDCONFLICTS$(Cand, C)$;
8 **if** $currCFT == null$ **then**
9 **if** *isComplete?*$(Cand, U)$ **then**
10 $newCFT \leftarrow$CONTROLLABILITYCHECK$(Cand)$;
11 **if** $newCFT == null$ **then**
12 `return` $Cand$;
13 **else**
14 $C \leftarrow C \cup \{newCFT\}$;
15 $Q \leftarrow Q \cup \{Cand\}$;
16 **endif**
17 **else**
18 $Q \leftarrow Q \cup$EXPANDONVARIABLE$\{Cand, U\}$;
19 **endif**
20 **else**
21 $Q \leftarrow Q \cup$EXPANDONCONFLICT$\{Cand, currCFT\}$;
22 **endif**
23 **end**
24 `return` $null$;

Algorithm 1: The CDRU algorithm

A candidate in CDRU is a 5-tuple $\langle A, R_e, R_u, C_r, C_{cont} \rangle$:

- A: a set of assignments to variables;
- R_e: a set of relaxations to requirement constraints;
- R_u: a set of tightening to contingent constraints;
- C_r: a set of conflicts resolved by this candidate;
- $C_{cont} \subseteq C_r$: a set of continuously resolved conflicts.

All these fields are empty sets for the first candidate. CDRU starts with a loop and continues until a candidate is

found that makes the CCTPU controllable (Line 10). Depending on the type of solutions required, function CONTROLLABILITYCHECK checks either strong or dynamic controllability, and returns a conflict if the candidate failed the test. If CDRU does not find a controllable relaxation before the queue exhausts, it returns null indicating that no solution exists for this CCTPU (Line 24).

Within each loop, CDRU first dequeues the best candidate (Line 6) and checks if it resolves all known conflicts (Line 7). If not, an unresolved conflict $currCFT$ will be returned by function UNRESOLVEDCONFLICTS. This unresolved is used by function EXPANDONCONFLICT (Line 21) to expand the current candidate, $Cand$. The child candidates of $Cand$ will then be queued.

If $Cand$ resolves all known conflicts, CDRU then proceeds to check if it is complete by comparing its assignments and unassigned variables in the CDRU (Line 9). If $Cand$ is incomplete, CDRU will expand it using function EXPANDONVARIABLE (Line 18), which creates child candidates using the domain values of an unassigned variable. If $Cand$ is complete and resolves all known conflicts, CDRU will check its controllability using function CONTROLLABILITYCHECK (Line 10). If a new conflict is returned by the function, it will be added to the list of known conflicts for expanding candidates. CDRU will also put $Cand$ back to the queue for future expansion since it now has an unresolved conflict (Line 15).

Conflict Learning For Strong Controllability

For CCTPs, a conflict is an inconsistent set of requirement constraints. It can be detected by negative loop detection algorithms: a negative cycle in the equivalent distance graph of a grounded CCTP can be mapped to a set of conflicting constraints. This is because of the one-to-one mapping between the distance edges and the lower/upper temporal bounds of constraints. However, this method does not apply to controllability checking algorithms. Due to the reduction procedures in both strong and dynamic controllability checking algorithms, the one-to-one mapping property is not preserved: during reductions, new distance edges are created and added to the graph, and the weights of some edges are modified. We cannot extract the sets of conflicting constraints from the negative loops in reduced graphs directly.

The key to solve this issue is to understand what constraints contributed to each distance edge in the reduced graph. We name these constraints the *supporting constraints*. The supporting constraints for an edge include the source constraint and the constraints that modify the weight of the edge during reduction. We extend the polynomial time algorithm in (Vidal and Fargier 1999) with additional procedures for recording supporting constraints during reductions (Algorithm 2). This extension enables the algorithm to extract a conflict from a negative loop in the reduced graph. The input to it is a grounded CCTPU without any unassigned variables. There are three major steps in this algorithm:

- Map the grounded CCTPU to its equivalent distance graph (Line 1) and record the supporting constraint of each distance edge in the graph with its source, which is either an upper or lower bound of a temporal constraint.
- Reduce all non-contingent edges that start (Line 14) or end (Line 5) at an uncontrollable node using the triangular reduction rule. If constraint A is reduced to C through B, the supporting constraints of C will be updated to the union of the supporting constraints of A and B (Line 11, 20).
- After the reductions, we run the Bellman-Ford algorithm on the reduced graph (Line 24). If a negative loop is detected, we collect the supporting constraints of all its edges into a set (Line 25) and return it as a conflict that makes the problem uncontrollable. Otherwise, the function returns null to indicate that the problem is strongly controllable.

Input: A grounded CCTPU $T = \langle V, E, E_u, L_v, L_p \rangle$.
Output: A conflict $\langle A, E' \rangle$ that makes T uncontrollable
Algorithm:

```
1  DG ←GETDISTANCEGRAPH(T);
2  ReductionQ ←NONCONTINGENTEDGES(DG);
3  while ReductionQ ≠ ∅ do
4      A ←Dequeue(ReductionQ);
5      if END(A) is uncontrollable then
6          B ← CONTINGENTEDGEENDAT(END(A));
7          A' ← REDUCE(A, B);
8          C ←GETEDGE(START(A),START(B));
9          if WEIGHT(A') <WEIGHT(C) then
10             C ← A';
11             SUPPORTS(C) ← SUPPORTS(A, B);
12             ReductionQ ← ReductionQ ∪ C;
13         endif
14     else if START(A) is uncontrollable then
15         B ← CONTINGENTEDGEBEGINAT(START(A));
16         A' ← REDUCE(A, B);
17         C ←GETEDGE(END(B),END(A));
18         if WEIGHT(A') <WEIGHT(E) then
19             C ← A';
20             SUPPORTS(C) ← SUPPORTS(A, B);
21         endif
22     endif
23 end
24 NCycle ←BELLMAN-FORD(DG);
25 return GETSUPPORTS(NCycle);
```

Algorithm 2: Strong controllability checking algorithm

We demonstrate this process using a temporal network with four constraints (Figure 4.1): A and B are contingent constraints with uncontrollable durations; C and D are requirement constraints. First, we map the network to its equivalent distance graph (Figure 4.2). Each distance edge in the graph is labeled with its weight and supporting constraints. The subscript after the constraint name, either U or L, specifies if the distance edge is generated from the upper or lower bound of the constraint.

There are two non-contingent edges in the graph, S2-S1 and E1-E2, and E1-E2 starts and ends at uncontrollable nodes (denoted by squares in the graph). We first reduce it using edge S2-E2, a contingent edge that shares the same end node with E1-E2. The result is a new edge E1-S2 with weight -2 and supporting constraints C_L, B_U, which are the union of the supporting constraints of E1-E2 and S2-E2 (Figure 4.3). Since E1-S2 starts at an uncontrollable node, we can further reduce it using E1-S1. The result is edge S1-S2 with weight 3 and supporting constraints C_L, B_U, A_L (Figure 4.4).

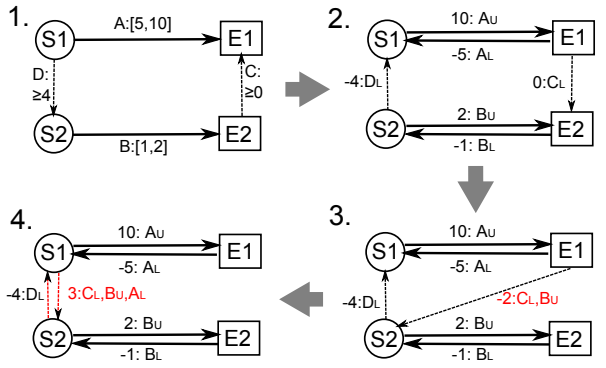

Figure 4: Supports recording during triangular reductions

It can be seen from the reduced graph that there is a negative cycle of two edges: S1-S2 and S2-S1. The negative cycle indicates that the original STNU is not strongly controllable, and the supporting constraints of these two edges, $\{A_L, B_U, C_L, D_L\}$, are in conflict and cause the failure. The negative value of the conflict is -1.

Using this algorithm for checking strong controllability and extracting conflicts does not add much overhead: it takes the same order of magnitude in time compared to consistency checking algorithms. Given a graph with V nodes and E edges, there will be at most $2E$ reductions and support constraint recordings. The time complexity of strong controllability is thus the same order of magnitude as consistency checking. Both are dominated by the $O(VE)$ negative cycle detection.

Conflict Learning For Dynamic Controllability

Our approach for learning conflicts from dynamic controllability checking algorithm is similar to that for strong controllability. We extend the fastDCcheck algorithm in (Morris 2006) with an additional step in its reduction procedures to record the supporting constraints of reduced edges. Its pseudo code is presented in Algorithm 3.

An STNU is dynamically controllable if and only if it does not have a semi-reducible negative cycle (Morris 2006). The fastDCcheck algorithm is designed based on this theorem. It converts the STNU to an equivalent distance graph of normal form (Line 1) and identifies all negative paths that start with a lower-case edge, called *moat paths*, through propagations (Line 6). The input STNU is determined to be dynamically controllable if none of these negative paths leads to a semi-reducible negative cycle (Line 3, 17).

During the reduction of moat paths, we record the supporting constraints for each reduced edge (Line 9). If the ALLMAXCONSISTENT function captures a negative cycle in the reduced graph, it will return a conflict that collects the supporting constraints of all edges in the cycle. There are five types of reductions in this procedure (Morris and Muscettola 2005; Morris 2006). We demonstrate the supports recording process for each of them in Figure 5.

Checking dynamic controllability and extracting conflicts is significantly harder than that for strong controllability. The fastDCcheck algorithm is currently the fastest DC checking algorithm with a complexity of $O(N^4)$, which is

Input: A grounded CCTPU $T = \langle V, E, E_u, L_v, L_p \rangle$.
Output: A conflict $\langle A, E' \rangle$ that makes T uncontrollable
Algorithm:

```
1  DG ←GETNORMALDISTANCEGRAPH(T);
2  for 1 to K do
3      NCycle ←ALLMAXCONSISTENT(DG);
4      if NCycle == null then
5          for E in LOWERCASEEDGES(DG) do
6              moatPaths ←PROPAGATE(E);
7              for Path in moatPaths do
8                  E' ←REDUCE(E,Path);
9                  SUPPORTS(E') ← SUPPORTS(E, Path);
10                 ADDTOGRAPH(E', DG)
11             end
12         end
13     else
14         return GETSUPPORTS(NCycle);
15     endif
16 end
17 NCycle ←ALLMAXCONSISTENT(DG);
18 return GETSUPPORTS(NCycle);
```

Algorithm 3: Dynamic controllability checking algorithm

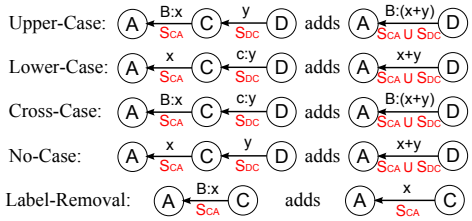

Figure 5: Supports recording in fastDCcheck reductions

two orders of magnitude higher than checking strong controllability. Recording supports during reductions does not increase the overall complexity of the algorithm. To improve efficiency, the algorithm can be terminated and return true after a no-reduction iteration. This is similar to the implementation in (Morris and Muscettola 2005) and will not affect the correctness of the results.

Resolving Conflicts with Contingent Constraints

CDRU uses the resolutions to unresolved conflicts to expand the search tree. There are three options for resolving a conflict that includes both requirement and contingent constraints:

1. Changing assignments to deactivate constraints in the conflict.
2. Relaxing the temporal bounds of requirement constraints.
3. Tightening the temporal bounds of contingent constraints.

The conflict resolution process, implemented as the EXPANDONCONFLICT function, is separated into two stages. The first stage implements the first option: resolving conflicts by changing assignments. This is identical to that in BCDR: we look for alternative assignments that can deactivate one or more constraints in the conflict, and use them to generate new candidates.

The second stage implements option 2 and 3. We compute the continuous relaxations to the relaxable constraints in the conflict. In the reduction process of both strong and dynamic controllability checking algorithms, the weights of supporting constraints are combined linearly: addition for requirement constraints and subtraction for contingent constraints.

In addition, only constraints that modifies the weights of reduced edges are recorded as supports. Therefore, the change to any constraint in a conflict will have equal effect on its negative cycle. If the amount of modification we applied to the constraints in a conflict exceeds the negation of its negative value, its negative cycle will be eliminated and the conflict is then resolved. This allows us to formulate an optimization problem with linear constraints to compute preferred continuous relaxations. The variables in this problem are the modifications applied to the temporal bounds of relaxable constraints. They are non-negative and their sum must compensate the negative value of the conflict.

$$\sum_{i \in Conflict}(\Delta UB_i + \Delta LB_i) \geq -1 \times NegativeValue.$$

For a contingent constraint e_u, ΔLB_u and ΔUB_u represent the amount of tightening. That is, the lower bound will increase ΔLB_u while the upper bound will decrease ΔUB_u in the relaxation. We add an additional linear constraint for each contingent constraint in the conflict to prevent over-tightening: $\Delta LB_u + \Delta UB_u \leq UB_u - LB_u$. For requirement constraints, these variables represent the amount of relaxations, and will make the lower time bounds lower and the upper time bounds higher. In addition, the semi-convex objective function of the problem is defined over these variables and minimizes the total cost of relaxing and tightening constraints:

$$min \sum_{i \in Conflict} f_{eiu}(\Delta UB_i) + f_{eil}(\Delta LB_i)$$

For example, the conflict in Figure 4 contains two contingent constraints and two requirement constraints. The negative value for this conflict is -1. We can construct the following optimization problem to compute its continuous relaxations. The second and third constraints prevent over-tightening to contingent constraints A and B.

$$min(f_A(\Delta A_L) + f_B(\Delta B_U) + f_C(\Delta C_L) + f_D(\Delta D_L)).$$
$$\textbf{s.t. } \Delta A_L + \Delta B_U + \Delta C_L + \Delta D_L \geq 1;$$
$$\Delta A_L \leq 5; \Delta B_U \leq 1;$$

CDRU constructs a new candidate using the best continuous relaxation, if one exists. This candidate and the candidates generated in the first stage will be returned by function EXPANDONCONFLICT to expand the search tree.

Application and Experimental Results

The CDRU algorithm has been incorporated within a mission advisory system, called *Uhura*, for helping oceanographers plan activities in their expeditions. During expeditions, the lead oceanographer has a list of prioritized scientific goals, which require the use of multiple deep-sea vehicles and sensors for sampling and mapping. However, there is uncertainty in the durations of their operations. Due to unexpected weather changes and unfamiliar terrain, predicting the times for completing activities and goals is infeasible. It is difficult for the oceanographer to determine which goals are feasible and to make a plan that accounts for each of the possible uncertain outcomes.

Uhura significantly reduced their workload and improved the planning process. It can quickly check the feasibility of expedition plans with uncertain activities, and make optimal decisions between alternatives. If a plan is over-subscribed, Uhura can expose its conflicts to the user and suggest preferred alternatives for them to make more informed decisions. With its assistance, the oceanographers can make sure that all critical goals can be achieved in uncertain situations.

In the rest of this section, we present empirical results that compare the performance of the following algorithms.

- BCDR: the consistency-based relaxation algorithm.
- CDRU-SC: CDRU for strongly controllable relaxations.
- CDRU-DC: CDRU for dynamically controllable relaxations.

As presented before, a controllable relaxation that accounts for uncertain durations will be much less risky compared to a relaxation that only restores consistency. However, such a relaxation may be of lower utility, and computing it may take much more time. Determining which approach to use for a given problem is a trade-off between efficiency and solution quality. We would like to provide some insights into the selection of approaches with the experiments.

Experimental Setup

We generate the test cases using a car-sharing network similar to (Yu and Williams 2013). The uncertain driving times between locations are modeled by contingent constraints. The stay at each location and the length of car reservation remain controllable. All constraints are relaxable and are associated with linear cost functions with different slopes. This makes it easy for us to compare the quality of solutions generated by the algorithms, since all test cases are solvable by three algorithms. We use the following parameters to characterize a test case:

$\mathbf{N_u}$: number of users per car, $1 \leq N_u \leq 12$;
$\mathbf{N_c}$: number of cars available, $1 \leq N_c \leq 10$;
$\mathbf{N_{act}}$: number of activities per reservation, $1 \leq N_{act} \leq 10$;
$\mathbf{N_{opt}}$: number of alternatives per activity, $2 \leq N_{opt} \leq 10$.

Each test case has $N_u \times N_c \times N_{act}$ discrete variables and $N_u \times N_c \times N_{act} \times 4N_{opt}$ constraints. 50% of the constraints are contingent constraints used for modeling driving times between locations. The locations in the tests are randomly sampled from a Boston map. The driving times between locations are computed using two different driving speeds: V_H:[20kph, 30kph] for express ways, and V_L:[7kph, 15kph] for city roads. Finally, the cost function of each relaxable constraint is linear with a gradient between [0,2] per minute. The rewards for choices of locations are in [0,500].

We generated two groups of test cases, each contains 2400 CCTPUs. The timeout for each test is 20 seconds, which is usually the maximum time a user would wait for a trip planner result. The first group is used to evaluate the efficiency of three algorithms in finding feasible solutions. The CCTPUs in this group have both choices and relaxable constraints. The second group is used to compare the solution quality of the three algorithms: the best solutions found by them may have different utility due to their different conflict detection and relaxation procedures. N_{opt} is set to 1 to remove choices from tests in this group.

Results

First, we evaluate the runtime performance of the three algorithms using the first group of tests. Since choices and al-

ternative destinations are allowed, there are many more constraints in these tests than the tests in group 2. We run each algorithm twice with different configurations. The first configuration uses a priority queue (Figure 6a). It returns the best solution and is useful when the user needs a solution of the highest quality. The second configuration does depth-first enumeration (Figure 6b), which is usually much faster but might return poor quality solutions. The x-axis represents the number of constraints in the test, and each dot in the graphs represents the averaged runtime across all tests in that category.

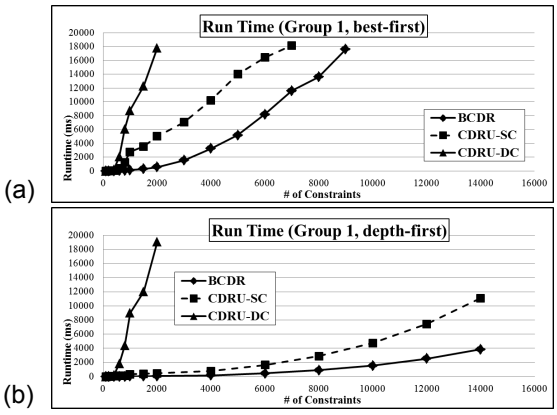

Figure 6: Benchmark results for test group 1

In both depth-first and best-first tests, CDRU-DC takes much more time than the other two algorithms. This is due to the slow reduction process of dynamic controllability checking and conflict extraction, whose complexity is two orders of magnitude higher than consistency and strong controllability checking. Compared to CDRU-DC, generating strongly controllable relaxations using CDRU-SC takes much less time. In theory, checking strong controllability and extracting conflicts is up to two times slower than checking consistency. However, the actual difference in time is bigger, since CDRU-SC has to resolve more conflicts than BCDR. This is demonstrated in the second group of tests.

Next, we compare the quality of solutions generated by the three algorithms using the second group of test cases. The results are presented in Figure 7. Again, CDRU-DC takes more time than the other two algorithms and times out on problems with more than 500 constraints. We identified 628 out of the 2400 test cases that are solved by all three algorithms within the time limit, and only present their results in the graph for comparison (Figure 7a). We measure the quality of a solution by its utility, which is computed by subtracting the costs of its relaxations from the rewards of its assignments. For easier comparison, we present the results using two differences between the solution utilities: BCDR over CDRU-DC and CDRU-DC over CDRU-SC.

The solutions generated by BCDR have the highest utility, followed by CDRU-DC. The solutions generated by CDRU-SC have the lowest average utility. In order to satisfy the uncontrollable outcomes of contingent constraints and restore the controllability of an over-constrained CCTPU, CDRU-SC and CDRU-DC add or tighten constraints during the reduction process. The solution space of their problems is

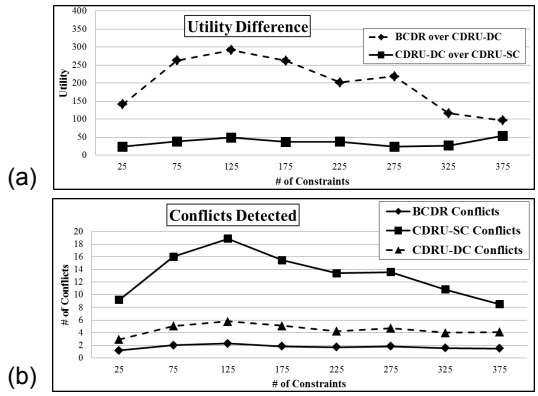

Figure 7: Benchmark results for test group 2

more restricted compared to BCDR. As a result, their best solutions are usually of lower utility.

Due to the less conservative execution strategy of dynamic controllability, CDRU-DC imposes less constraints than CDRU-SC and allows larger solution space. This can be measured by the number of conflicts detected and resolved by each algorithm before returning the solution (Figure 7b). The number of conflicts determines how constrained the optimization problem is while computing continuous relaxations: the more conflicts need to be resolved, the more constraints in the optimization, which require more relaxations to satisfy. This is why the solutions of CDRU-SC are of lower quality: on average, it has to resolve three times more conflicts than CDRU-DC before returning the solution.

In summary, both CDRU-SC and CDRU-DC can resolve over-constrained CCTPUs and enumerate relaxations to restore controllability. CDRU-SC runs significantly faster due to its simpler reduction process, while the quality of the relaxations generated by CDRU-DC is higher. If both static and dynamic execution strategies are acceptable, the decision on which algorithm to use is then determined by the trade-off between time and solution quality. In addition, best-first enumeration can be very time-consuming without an efficient heuristic function for guiding the search process. For large-scale problems, it is better to start with depth-first expansion and find a feasible solution first, then continue to improve it using a branch & bound approach.

Contributions

In this paper, we presented the Conflict-Directed Relaxation with Uncertainty algorithm, the first approach that resolves over-constrained conditional temporal problems with uncontrollable durations. Compared to previous relaxation algorithms, which only restore temporal consistency, CDRU generates continuous relaxations restoring both strong and dynamic controllability. It extends the conflict learning and resolution process in previous relaxation algorithms to account for contingent constraints, and incorporates this new capability into a conflict-directed framework for efficient enumeration of solutions. CDRU has been incorporated in a mission advisory system for activity planning and scheduling in oceanographic expeditions. In addition, experimental results have demonstrated its effectiveness in resolving large and highly constrained real-world problems.

Acknowledgments

Thanks to David Wang, Enrique Gonzalez, Julie Shah, Scott Smith and Ronald Provine for their support. This project is funded by the Boeing Company under grant MIT-BA-GTA-1. Additional support was provided by the DARPA meta program, under contract number 6923548.

References

Matthew Beaumont, Abdul Sattar, Michael Maher, and John Thornton. Solving overconstrained temporal reasoning problems. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI-2001)*, pages 37–49, 2001.

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

Luke Hunsberger, Roberto Posenato, and Carlo Combi. The dynamic controllability of conditional stns with uncertainty. In *Proceedings of the Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) Workshop*, pages 121–128, 2012.

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

Paul Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005*, pages 1193–1198. AAAI Press / The MIT Press, 2005.

Paul Morris. A structural characterization of temporal dynamic controllability. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP-2006)*, pages 375–389, 2006.

Bart Peintner, Michael D. Moffitt, and Martha E. Pollack. Solving over-constrained disjunctive temporal problems with preferences. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 2005.

Bart Peintner, Kristen Brent Venable, and Neil Yorke-Smith. Strong controllability of disjunctive temporal problems with uncertainty. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-2007)*, pages 856–863, 2007.

Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:248–253, 1998.

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

K. Brent Venable and Neil Yorke-Smith. Disjunctive temporal planning with uncertainty. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1721–1722, 2005.

Thierry Vidal and Helene Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:23–45, 1999.

Brian C. Williams and Robert J. Ragno. Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*, 155(12):1562–1595, 2002.

Peng Yu and Brian Williams. Continuously relaxing over-constrained conditional temporal problems through generalized conflict learning and resolution. In *Proceedings of the 23th International Joint Conference on Artificial Intelligence (IJCAI13)*, pages 2429–2436, 2013.

Zipcar. An overview of zipcar. `http://www.zipcar.com/about`, 2013. Accessed: 2013-04-07.