Continuously Relaxing Over-constrained Conditional Temporal Problems through Generalized Conflict Learning and Resolution

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

Abstract

Over-constrained temporal problems are commonly encountered while operating autonomous and decision support systems. An intelligent system must learn a human's preference over a problem in order to generate preferred resolutions that minimize perturbation. We present the Best-first Conflict-Directed Relaxation (BCDR) algorithm for enumerating the best continuous relaxation for an over-constrained conditional temporal problem with controllable choices. BCDR reformulates such a problem by making its temporal constraints relaxable and solves the problem using a conflict-directed approach. It extends the Conflict-Directed A* (CD-A*) algorithm to conditional temporal problems, by first generalizing the conflict learning process to include all discrete variable assignments and continuous temporal constraints, and then by guiding the forward search away from known infeasible regions using conflict resolution. When evaluated empirically on a range of coordinated car sharing network problems, BCDR demonstrates a substantial improvement in performance and solution quality compared to previous conflict-directed approaches.

1 Introduction

Temporal constraint networks [Dechter *et al.*, 1991] have been widely used to model planning and scheduling problems in daily life. They have been used to describe and reason over conditional and uncertain situations with multiple alternative plans. However, a solution to a temporal problem does not always exist. For example, in a car sharing network scenario, a user needs four hours to complete his shopping trip but only has three hours of reservation time. It is not enough for a scheduling program to just signal a failure. Instead, it should explain the situation and propose alternative plans for the user so that he can make a more informed decision, either to extend the reservation or to drop goals. Such a scenario is usually framed as an over-constrained temporal problem, and the goal is to find one or a set of preferred relaxations to the temporal constraints in the problem so that a consistent schedule can be found.

Prior work on over-constrained temporal problems starts with [Beaumont *et al.*, 2001], in which Partial Constraint Satisfaction techniques [Freuder and Wallace, 1992] are implemented to find the subset of temporal constraints that can be satisfied. Later, disjunctive constraints and optimality were added in the context of over-constrained Disjunctive Temporal Problems with Preferences (DTPPs) [Peintner *et al.*, 2005]. In a DTPP, the disjuncts of every constraint are assigned a preference function that maps the temporal constraint to a cost value. The optimal partial solution is obtained by enumerating consistent subproblems using Branch & Bound, as well as other optimization techniques introduced in [Khatib *et al.*, 2001]. Most of the prior work has focused on restoring consistency through complete suspension of constraints, however, in real-world scenarios, the user often wants to preserve as much of the schedule as possible to minimize the perturbation.

In this paper, we present our continuous relaxation approach, the Best-first Conflict-Directed Relaxation algorithm (BCDR), to address this issue. BCDR efficiently resolves over-constrained conditional temporal problems with controllable variables. It reformulates an over-constrained temporal problem by identifying its continuously relaxable temporal constraints, whose bounds can be partially relaxed to restore consistency. BCDR uses a conflict-directed strategy similar to Conflict-Directed A* [Williams and Ragno, 2002] to enumerate continuous relaxations in best-first order: it learns conflicts between constraints and variable assignments, and uses the resolutions to these conflicts to guide the search away from infeasible regions.

Note that this paper is not concerned about the dynamic or weak consistency of Conditional Temporal Problems with uncontrollable discrete variables (CTPs and CTPPs,[Tsamardinos *et al.*, 2003; Falda *et al.*, 2010]). We are only concerned about controllable variables that are not dependent on observation events. Solving such a problem is simpler than determining the dynamic/weak consistency of a CTP in that those tasks may require the enumeration of all possible scenarios.

2 Example

To motivate the need for continuously relaxing over-constrained temporal problems, we describe an example in the domain of a coordinated car sharing network, such as Zipcar [Zipcar, 2013]. Such a network provides an hourly rental service to its members: a rental car may be used by multiple members in a day. Each member must time their usage well so that the car can be returned on time. Otherwise, the next reservation will be affected and a penalty fee will be applied.

Consider the following example on John's trip for grocery shopping and lunch. He has reserved a car from 11am to 2pm, and is planning to go to one of the two grocery stores nearby: A or B. John has a preference for each store and their shopping times vary from 35 minutes to 50 minutes. After grocery shopping, John would like to have lunch at a restaurant, either X, Y or Z, before going home. Lunch takes a different amount of time for each restaurant. Finally, driving times to these locations are different, and John has to return his car back home in three hours (11am to 2pm) so that the next person can start his/her trip on time.

We develop the Controllable Conditional Temporal Problem (CCTP) formalism and use it to model John's trip and determine the best strategy for him that includes: which grocery store to visit, which restaurant to dine at, how much time to spend at each location and whether to extend his reservation. We start by defining two variables for the decisions he needs to make: GS (Grocery store) and RT (Restaurant). GS has two options in its domain: A (40) and B (100). Each option is associated with a positive reward value that represents John's preferences towards it, the larger the better. The other variable RT has three options: X (70), Y (80) and Z (30).

Next, we define twelve events as time points for the problem (Table 1): a reference point in time (S_T) that represents the beginning of the trip at 11am; a time point that indicates the end of the trip (R_T); and time points representing the arrival and departure of each locations (store A and B, restaurant X, Y and Z).

Events with Time Points			
Trip starts	S_T	Store A Arrive/Leave	A_A,A_L
Trip ends	R_T	Store B Arrive/Leave	B_A,B_L
Restaurant X Arrive/Leave			X_A,X_L
Restaurant Y Arrive/Leave			Y_A,Y_L
Restaurant Z Arrive/Leave			Z_A,Z_L

Table 1: Events in John's trip

Constraints (in minutes)		
$\mathbf{C_1}$:$\mathbf{A_L}$-$\mathbf{A_A} \geq \mathbf{40}$	C_6:A_A-$S_T \in [35, 50]$	$GS \leftarrow A$
$\mathbf{C_2}$:$\mathbf{B_L}$-$\mathbf{B_A} \geq \mathbf{35}$	C_7:B_A-$S_T \in [35, 40]$	$GS \leftarrow B$
$\mathbf{C_3}$:$\mathbf{X_L}$-$\mathbf{X_A} \geq \mathbf{50}$	C_8:R_T-$X_L \in [45, 50]$	$RT \leftarrow X$
$\mathbf{C_4}$:$\mathbf{Y_L}$-$\mathbf{Y_A} \geq \mathbf{75}$	C_9:R_T-$Y_L \in [40, 50]$	$RT \leftarrow Y$
$\mathbf{C_5}$:$\mathbf{Z_L}$-$\mathbf{Z_A} \geq \mathbf{100}$	C_{10}:R_T-$Z_L \in [50, 60]$	$RT \leftarrow Z$
C_{11}	X_A-$A_L[30, 40]$	$GS \leftarrow A$ and $RT \leftarrow X$
C_{12}	Y_A-$A_L[25, 30]$	$GS \leftarrow A$ and $RT \leftarrow Y$
C_{13}	Z_A-$A_L[20, 25]$	$GS \leftarrow A$ and $RT \leftarrow Z$
C_{14}	X_A-$B_L[35, 40]$	$GS \leftarrow B$ and $RT \leftarrow X$
C_{15}	Y_A-$B_L[25, 40]$	$GS \leftarrow B$ and $RT \leftarrow Y$
C_{16}	Z_A-$B_L[30, 35]$	$GS \leftarrow B$ and $RT \leftarrow Z$
$\mathbf{C_{17}}$	$\mathbf{R_T}$-$\mathbf{S_T} \in [\mathbf{0}, \mathbf{180}]$	

Table 2: Conditional Temporal Constraints in the CCTP

Table 2 shows all the conditional temporal constraints in the CCTP that encode the temporal relaxations between events. Constraints C_1 through C_5 are linear constraints that represent John's desired length of stay at five locations. For example, B_L–$B_A \geq 35$ indicates that John would like to spend at least 35 minutes at store B. These constraints are labeled by the assignments made to the decision variables: a constraint is activated only if its label assignment is made. For example, C_2 will be considered only if John chooses to shop at B, as shown in the right side of Table 2. Constraints C_6 through C_{16} are simple temporal constraints that encode the driving time required between locations. They are conditioned on assignments made to either GS or RT, or both (C_{11} through C_{16}). Finally, C_{17} constrains the duration of John's trip to three hours.

Some of the constraints highlighted in bold (C_1 through C_5 and C_{17}) are relaxable temporal constraints. They can be relaxed in order to restore the consistency of the problem, if necessary. Each relaxable constraint comes with one or two cost functions that describe John's preferences towards the relaxations for the upper and lower bounds. These functions map the relaxation from LB to LB', or from UB to UB', to a positive cost value, as seen in Figure 1. If the upper bound of C_{17} is relaxed from 180 minutes to 200 minutes, meaning that John delays his return by 20 minutes, the cost will be 40. On the other hand, if he shortens his lunch time by relaxing the lower bound of C_3 to 30, the cost would be 100. In this example, we assume that all other relaxable constraints have linear cost functions with gradient 1 for simplicity.

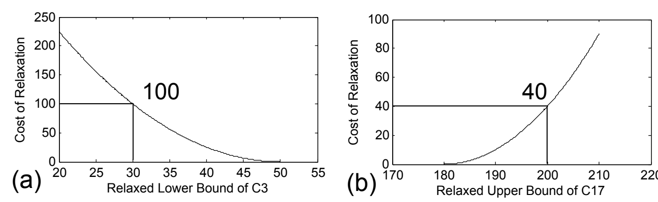


Figure 1: Preference functions for C_3 and C_{17}

Relaxation 1	**Relaxation 2**	**Relaxation 3**
$GS \leftarrow B$	$GS \leftarrow B$	$GS \leftarrow B$
$RT \leftarrow Y$	$RT \leftarrow X$	$RT \leftarrow Y$
C_4 to ≥ 50	C_3 to ≥ 48	C_4 to ≥ 55
C_{17} to $\in [0, 185]$	C_2 to ≥ 17	C_2 to ≥ 25
Utility: 152.5	Utility: 151	Utility: 150
↘**Do not relax** C_{17} ↗		↘ C_2 **is at least 25**↗

Table 3: Three preferred relaxations to the CCTP

Before relaxing any constraints, there is no consistent solution to the problem. The cause of failure is that three hours is not enough for John to complete both shopping and dining tasks: driving to the nearest grocery store and restaurant will consume at least 100 minutes, which brings the minimum trip duration to 200 minutes. Therefore, one or more temporal constraints need to be relaxed. Table 3 shows three consistent relaxations for the CCTP ranked in best-first order.

Relaxation 1, which is first presented to John, suggests shopping at B and having lunch in Y. The lunch time should be reduced to 50 minutes and the reservation should be extended by 5 minutes. The utility of the relaxation is 152.5, which is computed by summing up the reward of two assignments, $GS \leftarrow B$ and $RT \leftarrow Y$, and subtracting the cost of relaxing C_4 and C_{17}. If John changes his mind and decides not to relax C_{17}, Relaxation 2 will be generated which incorporates this new requirement. It takes John to X for lunch, shortens the lunch time to 48 minutes and reduces the shopping time to 17 minutes. If John is still unsatisfied, he may add an additional requirement that shopping time should be no less than 25 minutes. BCDR will continue the search and present Relaxation 3, which respects both newly added requirements.

This example demonstrates the advantage of continuous relaxation: it minimizes perturbation to the original problem. Compared to discrete relaxations, which may ask John not to shop or have lunch, continuous relaxations preserve more of the original problem while restoring consistency. In addition, the conflict-directed search technique used by BCDR enables it to adapt to newly added constraints and enumerate relaxations accordingly.

3 Problem Statement

Temporal problems with choices are usually modeled using Conditional Temporal Problems (CTPs,[Tsamardinos *et al.*, 2003]). It is a generalization of the restricted problem class of Simple Temporal Problems (STPs,[Dechter *et al.*, 1991]) by adding uncontrollable discrete choices and by conditioning the occurrence of events and simple temporal constraints on the outcomes of these choices. CTPs are capable of modeling conditional plans and uncertainty during executions.

Definition 1. *A CTP is a 6-tuple $\langle V, E, L, OV, O, P\rangle$ where:*

- *P is a set of Boolean atomic propositions;*
- *V is a set of events representing designated time points;*
- *E is a set of simple temporal constraints that restricts the time points in V, and are of the form $l_{ij} \leq v_j - v_i \leq u_{ij}, l_{ij}, u_{ij} \in \mathcal{R}$;*
- *Q is a set of literals of P;*
- *$L : V \rightarrow Q$ is a function that attaches conjunctions of literals, $q_i \in Q$, to each event $v_i \in V$;*
- *$OV \subseteq V$ is a set of observation events that provides the truth value for $p_i \in P$ through function $O : P \rightarrow OV$.*

In a CTP, each event is associated with a conjunctive set of literals, called a *label*. If the label of an event is evaluated to be true, the event is said to be activated and needs to be scheduled. Otherwise, the event and its associated temporal constraints can be ignored.

The solution to a CTP is a schedule that assigns a time point to each event in the CTP and is consistent with the temporal constraints. There are three notions of CTP consistency: Strong, Dynamic and Weak consistency, depending on the assumptions made over the outcomes of observation variables [Tsamardinos *et al.*, 2003]. Conditional Temporal Problems with Preferences (CTPPs,[Falda *et al.*, 2010]) extend CTPs by allowing fuzzy temporal constraints and fuzzy atomic propositions. This allows the user to specify preferences over the execution time of each event $v_i \in V$, and compare two schedules T_1 and T_2 using a preference function that maps a schedule to a utility value $f : T \rightarrow \mathcal{R}^+$ [Khatib *et al.*, 2001].

The problems that BCDR addresses, Controllable Conditional Temporal Problems (CCTPs), are closely related to CTPs; however, there are two important differences. First, CCTPs assume that all variables are controllable. Consequently, to determine the consistency of a CCTP, it is sufficient to find one consistent set of discrete variable assignments. Second, a CCTP extends the domains of discrete variables from binary to any finite domains, and allows the discrete variables to be conditioned on assignments to other variables. Compared to the Temporal Constraint Satisfaction Problems (TCSPs) formulation [Dechter *et al.*, 1991], whose constraints are disjunctions of possible simple temporal constraints, CCTP is more expressive in that it allows a sequence of temporal constraints to be conditioned on choices.

Definition 2. *A CCTP is an 8-tuple $\langle V, E, RE, L_v, L_p, P, f_v, f_e\rangle$ where:*

- *P is a set of controllable finite domain discrete variables;*
- *V is a set of events representing designated time points;*
- *E is a set of temporal constraints between pairs of events $v_i \in V$;*
- *$RE \in E$ 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**.*

To allow the relaxation for an over-constrained temporal problem, we include relaxable temporal constraints in the definition of CCTP, similar to the soft constraints in a Simple Temporal Problem with Preferences (STPP,[Rossi *et al.*, 2002]). We do not use a disjunctive set of temporal bounds for soft constraints. Instead, the constraint is soft in that its lower or upper bounds can be relaxed at the price of increasing cost. The cost is defined over the degree of relaxation made to the lower and upper bounds.

There are two preference functions, f_p and f_e. f_p is a reward function over the assignments to controllable discrete variables $p_i \in P$. Each assignment is mapped to a positive reward value, such as $RT \leftarrow X : 50$. The larger the number is, the more preferred the choice will be. f_e is a positive cost function defined over relaxable constraints. The cost of relaxing an upper bound constraint $E_{ij} : v_j - v_i \leq u_{ij}$ from u_{ij} to u'_{ij} is $f_{eij}(u'_{ij} - u_{ij})$. Figure 1b shows an example function defined over $u'_{ij} - u_{ij}$.

The cost function for temporal constraints that restrict the lower bounds between two events is $f_{eij}(l_{ij} - l'_{ij})$. This is illustrated in Figure 1a. We assume that the user always prefers smaller relaxations. Therefore, all f_e functions must be monotonically increasing, and equal to 0 when there is no relaxation. f_e can be viewed as a semi-convex [Khatib *et al.*, 2001] function with a segment of zero cost when there is no relaxation. This assumption simplifies our relaxation process, as the tightest relaxation will always result in the lowest cost. For relaxable simple temporal constraints, two separate cost functions are required for the lower and upper bounds.

We define the solution to a CCTP as a pair $\langle A, R \rangle$, where:

- A is a complete set of assignments to some discrete variables in P that leaves no variable unassigned.
- R is a set of relaxed bounds of some relaxable constraints in RE.

such that the CCTP is temporally consistent. The utility of a relaxation is computed by subtracting the relaxation cost from the assignment reward: $\sum_{p_i} f_{p_i}(p_i \leftarrow value_i) - \sum_{e_i} f_{e_i}(e_i \rightarrow e'_i)$. The most preferred relaxation to a CCTP is the one with the highest utility value according to f_p and f_e.

Note that CCTP is similar, though different in notations, to the Optimal Conditional Simple Temporal Problem (OCSTP) formulation introduced by [Effinger, 2006]. OCSTP and CCTP are equally expressive for consistency problems. OCSTP encodes temporal constraints as the domain values of discrete variables, and its relaxations are represented by additional domain values. This makes it difficult to encode the relaxable temporal constraints using an OCSTP formulation. We chose CCTP for relaxation problems because of its compact representation of constraint relaxations: consistency can be restored by relaxing the lower or upper bounds of relaxable temporal constraints.

The OCSTP solver introduced in [Effinger, 2006] was designed to solve consistency problems only. It uses a depth-first strategy to find a set of variable assignments that activates a consistent set of temporal constraints. Unlike BCDR, the OCSTP solver cannot relax temporal constraints to restore consistency; the solver can only signal failure given an over-constrained problem.

4 Approach

In this section, we present the Best-first Conflict-Directed Relaxation algorithm that enumerates the relaxations to a CCTP in best-first order. This can be viewed as an extension to the Conflict-Directed A* algorithm [Williams and Ragno, 2002] by generalizing the conflicts learning and resolution capability. CD-A* enumerates likely solutions to discrete domain CSPs with conflicts learned from inconsistent sets of assignments. Once detected, a conflict is used to prune the search space by extending each partial candidate with its resolutions. To resolve a CCTP using the conflict-directed strategy, we have to first generalize the conflicts to include conditional and temporal constraints, and then generate both discrete and continuous constituent relaxations to the conflict. We will first give an overview of the BCDR algorithm, and then discuss the conflict learning and resolution in detail.

4.1 The BCDR algorithm

BCDR takes an A* search strategy by evaluating each partial candidate using an admissible heuristic function and expanding the search tree in best-first order. The first relaxation found is guaranteed to be the best one. It uses two types of expansions to explore the search space, *Expand on an unassigned variable* and *Expand on an unresolved conflict*, which differentiates BCDR from previous relaxation algorithms. The pseudo code of BCDR is given in Algorithm 1.

Input: A CCTP $T = \langle V, E, RE, L_v, L_p, P, f_v, f_e \rangle$.
Output: A relaxation $\langle A, R \rangle$ that maximizes $f_v - f_e$.
Initialization:
1 $Cand \leftarrow \langle A, R, C_r, C_{cont} \rangle$; the first candidate;
2 $Q \leftarrow \{Cand\}$; a priority queue that records candidates;
3 $C \leftarrow \{\}$; the set of all known conflicts;
4 $U \leftarrow V$; the list of unassigned controllable variables;
Algorithm:
5 **while** $Q \neq \emptyset$ **do**
6   $Cand \leftarrow$Dequeue(Q);
7   $currCFT \leftarrow$ResolveKnownConflicts?($Cand, C$);
8   **if** $currCFT == null$ **then**
9   **if** *isComplete?*($Cand, U$) **then**
10       $newCFT \leftarrow$ConsistencyCheck($cand$);
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 BCDR algorithm

BCDR starts with an empty candidate in the queue (Line 1). A candidate is a 4-tuple $\langle A, R, C_r, C_{cont} \rangle$ with assignments A, relaxations R, resolved conflicts C_r and continuously resolved conflicts C_{cont}, all being empty lists in the first candidate. BCDR continues looping until the first relaxation is found that makes the CCTP consistent (Line 11). If BCDR does not find a consistent relaxation until the queue is exhausted, it returns null indicating that no relaxation exists for the input CCTP (Line 24).

Within each loop, BCDR first dequeues the best partial candidate (Line 6). It checks if $Cand$ resolves all known conflicts (Line 7). If not, an unresolved conflict $currCFT$ will be returned by function ResolveKnownConflicts?, which compares the resolved conflicts C_r in $Cand$ with all known conflicts C. $currCFT$ is then used for expanding

$Cand$ by function EXPANDONCONFLICT (Line 21). The child candidates of $Cand$ will then be enqueued.

If $Cand$ resolves all known conflicts, BCDR then checks if it is complete by comparing its assignments and all unassigned variables in the CCTP (Line 9). If $Cand$ is incomplete, BCDR will expand it using the assignments to one unassigned variable through function EXPANDONVARIABLE (Line 18). For example, assume that we need to expand a partial candidate $\{GS{=}A, RT{=}X\}$ with variable $FD{:}\{Steak, Salmon\}$, we simply create two child candidates that extends the partial candidate using two possible assignments of FD (Figure 2a). The expanded candidates will be added back to Q.

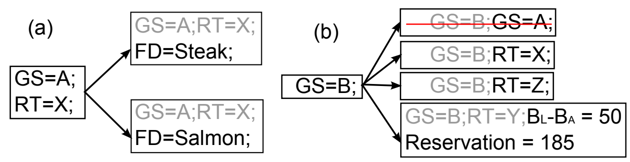


Figure 2: Example of expanding on candidate and conflict

If $Cand$ is complete, BCDR proceeds to check its consistency using function CONSISTENCYCHECK (Line 10). If no conflict is returned, $Cand$ will be returned as the best relaxation (Line 12). If a new conflict, $newCFT$, is detected by CONSISTENCYCHECK, BCDR will record it and put $Cand$ back to the queue for future expansions (Line 14,15).

4.2 Learning Conflicts through Negative Cycles

Given a complete candidate that assigns all active discrete variables, function CONSISTENCYCHECK checks the consistency of all activated temporal constraints. BCDR implements the Incremental Temporal Consistency algorithm [hsiang Shu *et al.*, 2005] for checking temporal consistency. If the set of temporal constraints is inconsistent, ITC will return a simple negative cycle as the cause of failure. We can extract the minimal inconsistent set of temporal constraints, also called minimal conflict [Liffiton *et al.*, 2005], using this simple negative cycle. For example, Figure 3 shows a simple negative cycle detected in John's trip: the reservation time is too tight for activities at B and Y.

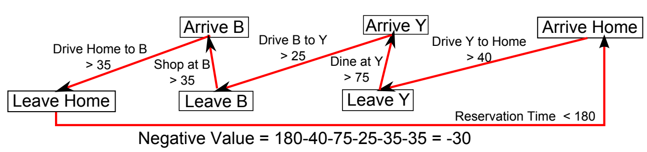


Figure 3: A negative cycle in John's trip

Previous approaches [Effinger and Williams, 2005; Li and Williams, 2005] only extract the discrete variable assignments as conflict, which is $\{GS = B; RT = Y\}$ in this case. Since we are looking for relaxations to temporal constraints, we should include them in the conflict as well, and use their relaxations to resolve the conflict. In addition, because a temporal constraint may depend on one or more assignments, its label must be included in the conflict as well. In short, BCDR learns a conflict from a simple negative cycle. A conflict is composed of the temporal constraints involved in the cycle and the assignments required to activate them. For example, the generalized conflict we can learn from Figure 3 is:

Assignments: $GS{=}B$; $RT{=}Y$;
Constraints: $R_T\text{-}S_T \in [0, 180]$;
$GS{=}B \rightarrow B_A\text{-}S_T \in [35, 40]$; $GS{=}B \rightarrow B_L\text{-}B_A \geq 35$;
$GS{=}B \wedge RT{=}Y \rightarrow Y_A\text{-}B_L \in [25, 40]$;
$RT{=}Y \rightarrow Y_L\text{-}Y_A \geq 75$; $RT{=}Y \rightarrow R_T\text{-}Y_L \in [40, 50]$;

4.3 Generalized Conflict Resolutions

Given a minimal conflict, we can compute their resolutions and use them to expand existing candidates so that future expansions of the candidates will not enter the infeasible region represented by this conflict again. This is the core principle behind conflict-directed search. Previous approaches generate the resolutions, which are called constituent relaxations, by either flipping the assignments to the discrete variables [Williams and Ragno, 2002; Effinger and Williams, 2005] or suspending temporal constraints [Moffitt and Pollack, 2005]. BCDR generalizes the conflict resolution to include both discrete assignments and temporal constraint relaxations: the more we can learn from a conflict, the larger infeasible region we may avoid in the forward search. In addition, we would like to relax the temporal constraints continuously to the minimal extent, instead of completely suspending them, in order to minimize the perturbations.

Input: A candidate to expand $Cand\langle A, R, C_r, C_{cont}\rangle$ and a minimal conflict $currCFT$.
Output: A set of expanded candidates $newCands$.
Initialization:
$newCands \leftarrow \{\}$;
$CFTs \leftarrow C_{cont} \cup \{currCFT\}$; conflicts to be resolved continuously;
Algorithm:
for $a \in A$ **do**
  $A_{alter} = A_{alter} \cup$GETALTERNATIVES$(a)$;
  $A_{alter} = A_{alter} \cup$GETALTERNATIVES$(label(a))$;
end
for $a_{extend} \in A_{alter}$ **do**
  if NOTCOMPETING(A, a_{extend}) **then**
    $Cand_{new} \leftarrow \langle A \cup \{a_{extend}\}, R, C_r, C_{cont}\rangle$;
    $newCands \leftarrow newCands \cup Cand_{new}$;
  end
end
$\langle E_{relax}, N_{value}\rangle \leftarrow$EXTRACTCONSTRAINTS$(CFTs)$;
$f_{obj} \leftarrow \sum_{e \in E_{relax}} f_e(\Delta e)$;
$R_{new} \leftarrow$OPTIMIZE$(f_{obj}, \langle E_{relax}, N_{value}\rangle)$;
if $R_{new} \neq null$ **then**
  $Cand_{new} \leftarrow \langle A, R_new, C_r, C_{cont}\rangle$;
  $newCands \leftarrow newCands \cup Cand_{new}$;
end
return $newCands$;

Algorithm 2: Function EXPANDONCONFLICT

Function EXPANDONCONFLICT is presented in Algorithm

2. The resolution is separated into two stages: First, we generate constituent relaxations by negating variable assignments (Line 3-12). If a variable v_i is conditioned on other assignments, in addition to flipping the assignment to v_i, we can also negate its label. This deactivates the variable and resolves the conflict. For example, for a conflict that involves assignment $FD_Y = Steak$, if we know that variable FD_Y has label $RT = Y$, we can resolve the conflict by flipping the assignment to either FD_Y or RT: FD_Y=$Salmon$, RT=X or RT=Z.

In the second stage, we compute the optimal continuous relaxation to the relaxable temporal constraints that can resolve the conflict (Line 13-19). We formulate the relaxation as an optimization problem with linear constraints (Line 13) and semi-convex objective function (Line 14). The objective function is the minimization over the sum of the relaxation costs of all relaxable constraints. The variables in this optimization problem are ΔLB_is and ΔUB_is, which are the relaxations applied to each relaxable temporal constraint. They are non-negative and their sum must compensate for the negative value of the conflict (Line 15). The optimal relaxation will not over-relax any constraints, due to the semi-convex assumption over cost functions. It is sufficient to relax the relaxable constraints to the extent that just eliminates the negative cycle, that is:

$$min \sum_{i \in conflict} (f_{eij}(u'_{ij} - u_{ij}) + f_{eij}(l_{ij} - l'_{ij}))$$
$$\text{s.t.} \sum_{i \in conflict} (e'_{ij} - e_{ij}) = -1 \times N_{value}$$

For example, the conflict in (Figure 3) involves six constraints. The negative value for this conflict is -30. Among the six constraints, three of them are relaxable constraints whose bounds can be relaxed: $Reservation \in [0, 180]$, B_L-$B_A \geq 35$ and Y_L-$Y_A \geq 75$. We can define the following optimization problem for computing the continuous relaxation:

$$min(f(\Delta(B_L\text{-}B_A)) + f(\Delta(Y_L\text{-}Y_A)) + f(\Delta(R_T\text{-}S_T)));$$
$$\text{s.t. } \Delta(B_L\text{-}B_A) + \Delta(Y_L\text{-}Y_A) + \Delta(R_T\text{-}S_T) = 30;$$

The solution to the above optimization problem is a set of relaxed bounds of the relaxable temporal constraints that resolves the conflict and minimizes the cost. In this case, the best relaxation is: Relax Y_L-Y_A to 50 and $Reservation$ to 185. The cost is 27.5. In fact, this problem can also be viewed as a Simple Temporal Problem with Preferences. [Khatib *et al.*, 2001] demonstrates that finding the optimal solution to a STPP with semi-convex preferences is tractable. In real world applications, we may substitute different optimization algorithms, depending on the preference functions, to improve efficiency.

In total, BCDR generates four constituent relaxations: three new assignments derived from negating assignments and one continuous relaxation. They are used to extend the partial candidates so that future extensions will not run into the same conflict again, as demonstrated in Figure 2b. Note that one extension is removed due to its conflicting assignments.

4.4 Reactive BCDR

Finally, we demonstrate the reactive implementation of BCDR that can continuously enumerate best relaxations to a given CCTP based on users' responses (Algorithm 3). This is achieved through a slight modification to BCDR: the algorithm keeps tracking the search queue and known conflicts even after a solution is returned (Line 4). If the user rejects the current solution, BCDR will record his/her inputs as a conflict and add it to the known conflicts list C (Line 11). This procedure guarantees that all candidates expanded in the future will satisfy this newly added requirement. BCDR then starts searching again for the next solution that resolves all conflicts while maximizing the utility value.

Input: A CCTP $T = \langle V, E, RE, L_v, L_p, P, f_v, f_e \rangle$.
Output: A relaxation $\langle A, R \rangle$ that maximizes $f_v - f_e$.
Initialization:
1 $Sol \leftarrow \langle A, R, C_r, C_{cont} \rangle$; a solution to T;
2 $C \leftarrow \{\}$; the set of all known conflicts;
Algorithm:
3 **while** $true$ **do**
4 $\quad (Sol, C) \leftarrow$BCDR$(T, C)$;
5 $\quad$ **if** $Sol == null$ **then**
6 $\quad\quad$ `return` $null$;
7 $\quad$ **else**
8 $\quad\quad$ **if** $Accepted?(Sol)$ **then**
9 $\quad\quad\quad$ `return` Sol;
10 $\quad\quad$ **else**
11 $\quad\quad\quad$ $C \leftarrow C \cup$ GETREQUIREMENT$\{UserInputs\}$
12 $\quad\quad$ **endif**
13 $\quad$ **endif**
14 **end**

Algorithm 3: Reactive BCDR

5 Experimental Results

To demonstrate the effectiveness of our approach, we present empirical results to compare two BCDR implementations: BCDR-GC (generalized conflict learning and resolution) and BCDR-DC (discrete conflict resolution only). BCDR-DC implements the discrete conflict resolution technique [Li and Williams, 2005]. It learns conflicts that can only be resolved discretely and computes continuous relaxations once a complete candidate is generated. In our experiments, both implementations are set to find the best continuous relaxation.

In addition, we compare BCDR-GC to DFS-GC, a depth-first implementation with the generalized conflict resolution technique. The only difference is that DFS-GC uses a Last-In-First-Out queue to store candidates (Line 6, Algorithm 1). This implementation is faster in finding feasible solutions, but would not guarantee to find the highest-utility solution. As a time-critical alternative to BCDR-GC, we demonstrate the improvements of DFS-GC in run-time performance and the loss in solution quality.

5.1 Setup

We generated random CCTPs using a simulated car sharing network similar to Section 2. To make it more complex, we extended the problem to allow multiple users and cars: there is always another user waiting for the shared car following each reservation; and there are multiple cars that can be reserved in parallel. In addition, two users using different cars may want to meet during their reservations. We use the following control parameters in the problem generator to control the complexity of a test case:

- N_u: number of users per car. $1 \leq N_u \leq 10$.
- N_c: number of cars available. $1 \leq N_c \leq 12$.
- N_{act}: number of activities per reservation. $1 \leq N_{act} \leq 8$.
- N_{opt}: number of alternatives per activity. $2 \leq N_{opt} \leq 10$.

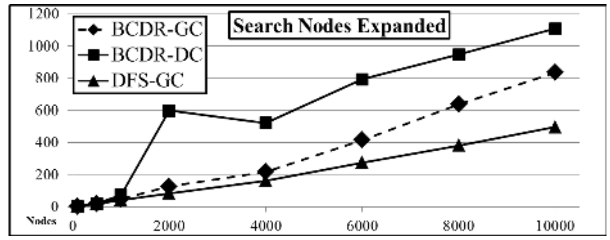
The total number of discrete variables in a test case is $N_u \times N_c \times N_{act}$, and the domain size of each variable is determined by its N_{opt}. We use a map of Boston and randomly sample locations on the map. The driving time is computed using an average speed randomly selected between 30 and 50 mph. The duration at each location and the reservation time, T_{act} and T_{res}, are randomly sampled in $[0, 90]$ and $[0, 360]$ (minutes), respectively. These durations are encoded as relaxable temporal constraints. We define linear preference functions over these relaxable constraints with costs sampled between 0 and 10. The reward for each variable assignment, denoting a location selection for each activity, ranges from 0 to 1000. Finally, arbitrary temporal constraints are added between cars to simulate a meeting between two users. We use LPSolve as the linear optimizer for all three algorithms [Berkelaar *et al.*, 2008]. Note that the generalized conflict resolution works with both linear and non-linear preference functions. We use linear functions and LPSolve only for the purpose of benchmark.

In total, 2400 test cases were generated with the number of constraints ranging from 50 to 10000. The time out for each test is 20 seconds, which is usually the maximum time a user is willing to wait in a reservation system.

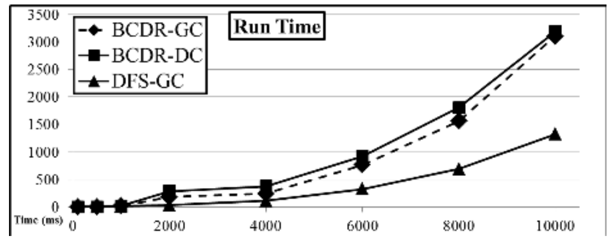
5.2 Results

The results are presented in Figure 4. Each dot in the graph represents the averaged results computed across all test cases in that category. The x-axis in each graph represents the number of constraints in each test problem. As can be seen in Figure 4a, the number of search nodes expanded by BCDR-GC before returning the best relaxation is significantly smaller than that expanded by BCDR-DC. This difference is because the DC procedure is more conservative at pruning search space: a conflict will be learned and used for splitting only if it can be resolved by discretely flipping assignments but not by continuously relaxing constraints. Therefore, the number of search nodes checked by BCDR-GC is no more than the number checked by BCDR-DC before returning the best solution.

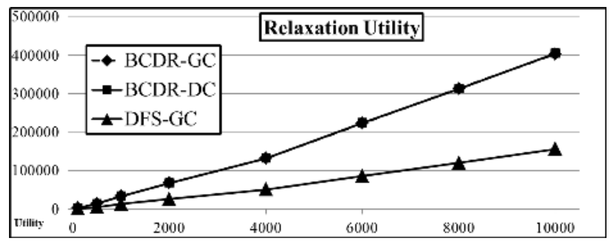
The generalization of conflict learning and resolution to continuous relaxation efficiently prunes the inconsistent regions in the search domain and avoids nearly 30% of unnecessary node expansions, compared to discrete conflict resolution. The reduced number of candidate expansions helps BCDR-GC achieve higher run-time performance compared to BCDR-DC: the average savings is approximately 10%-15% (Figure 4b). We believe that the run-time performance of BCDR-GC can be further improved if we implement the continuous conflict resolution in an incremental manner, since



(a) Number of nodes expanded



(b) Run time



(c) The utility value of the first solution

Figure 4: Benchmark results of each algorithm

the EXPANDONCONFLICT function keeps solving optimization problems with a growing set of constraints.

Next, if the user wants a quick solution, DFS-GC is a good alternative to BCDR-GC. Figure 4a shows that DFS-GC expands 50% fewer nodes than BCDR-GC when the first relaxation is returned, and cuts the run-time by half (Figure 4b). However, the faster result comes at the cost of decreased solution quality: the utility of the first solution returned by DFS-GC is 70% lower when compared to that returned by BCDR-GC (Figure 4c). If time permits, the users may continue running DFS-GC after obtaining the first relaxation with a Branch & Bound approach, or they may use BCDR-GC, which is guaranteed to return the best relaxation.

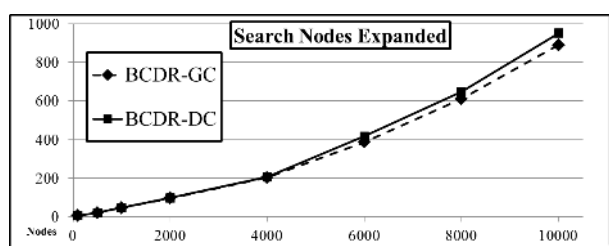


Figure 5: Performance of BCDR-GC and BCDR-DC on problems of low relaxation costs

Despite the promising results, we recognize a limitation of the BCDR-GC algorithm: when the cost of relaxing constraints is orders of magnitude lower than reward, the GC procedure may not provide a significant improvement in performance compared to DC. For example, we generated an additional set of tests with reduced cost functions: the range of gradients is changed from $[0, 10]$ in Section 5.1 to $[0, 0.1]$. As can be seen in Figure 5, BCDR-GC and BCDR-DC expand nearly the same number of search nodes before returning the best solution. These similar outcomes are due to the nature of the best-first search strategy: when the costs are much lower than the rewards, GC will apply the continuous relaxation only close to the leaves of the search tree, which reduces its effectiveness at pruning the search space and its advantage over BCDR-DC. However, in all cases, BCDR-GC will perform at least as fast as BCDR-DC.

6 Contributions

In this paper, we presented the Best-first Conflict-Directed Relaxation algorithm, the first approach that continuously relaxes over-constrained conditional temporal problems. Compared to previous relaxation algorithms, which restore consistency by suspending constraints, BCDR minimizes the perturbation by continuously relaxing temporal constraints to the minimal extent. It reformulates these problems as Controllable Conditional Temporal Problems, which allow relaxable temporal constraints. With the implementation of generalized conflict learning and resolution, BCDR is more efficient at enumerating the best relaxations when compared to previous conflict-directed approaches. Experimental results have demonstrated its effectiveness in resolving large and highly constrained real-world problems.

7 Acknowledgments

Thanks to David Wang, Masahiro Ono, Eric Timmons, 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

[Beaumont *et al.*, 2001] 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.

[Berkelaar *et al.*, 2008] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. *lpsolve : Open source (Mixed-Integer) Linear Programming system*, 2008.

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

[Effinger and Williams, 2005] Robert Effinger and Brian Williams. Conflict-directed search through disjunctive temporal plan networks. *CSAIL Research Abstracts - 2005*, 2005.

[Effinger, 2006] Robert T. Effinger. Optimal temporal planning at reactive time scales via dynamic backtracking branch and bound. Master's thesis, Massachusetts Institute of Technology, 2006.

[Falda *et al.*, 2010] Marco Falda, Francesca Rossi, and K. Brent Venable. Dynamic consistency of fuzzy conditional temporal problems. *Journal of Intelligent Manufacturing*, 21:75–88, 2010.

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

[hsiang Shu *et al.*, 2005] I hsiang Shu, Robert Effinger, and Brian C. Williams. Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 05)*, pages 252–261, 2005.

[Khatib *et al.*, 2001] Lina Khatib, Robert Morris, Robert Morris, and Francesca Rossi. Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 322–327, 2001.

[Li and Williams, 2005] Hui Li and Brian Williams. Generalized conflict learning for hybrid discrete/linear optimization. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pages 415–429, 2005.

[Liffiton *et al.*, 2005] M.H. Liffiton, M.D. Moffitt, M.E. Pollack, and K.A. Sakallah. Identifying conflicts in overconstrained temporal problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 205–211, 2005.

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

[Peintner *et al.*, 2005] 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.

[Rossi *et al.*, 2002] Francesca Rossi, Alessandro Sperduti, Kristen Brent Venable, Lina Khatib, Paul H. Morris, and Robert A. Morris. Learning and solving soft temporal constraints: An experimental study. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 249–263, 2002.

[Tsamardinos *et al.*, 2003] Ioannis Tsamardinos, Thierry Vidal, and Martha Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8:365–388, 2003.

[Williams and Ragno, 2002] 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.

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