Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)

# Dynamic Controllability of Controllable Conditional Temporal Problems with Uncertainty

**Jing Cui, Patrik Haslum**
ANU & DATA61
Canberra, Australia
{cui.jing—patrik.haslum}@anu.edu.au

## Abstract

Dynamic Controllability (DC) of a Simple Temporal Problem with Uncertainty (STPU) uses a dynamic decision strategy, rather than a fixed schedule, to tackle temporal uncertainty. We extend this concept to the Controllable Conditional Temporal Problem with Uncertainty (CCTPU), which extends the STPU by conditioning temporal constraints on the assignment of controllable discrete variables. We define dynamic controllability of a CCTPU as the existence of a strategy that decides on both the values of discrete choice variables and the scheduling of controllable time points dynamically. This contrasts with previous work, which made a static assignment of choice variables and dynamic decisions over time points only. We propose an algorithm to find such a fully dynamic strategy. The algorithm computes the "envelope" of outcomes of temporal uncertainty in which a particular assignment of discrete variables is feasible, and aggregates these over all choices. When an aggregated envelope covers all uncertain situations of the CCTPU, the problem is dynamically controllable. However, the algorithm is not complete. Experiments on an existing set of CCTPU benchmarks show that there are cases in which making both discrete and temporal decisions dynamically it is feasible to satisfy the problem constraints, while assigning the discrete variables statically it is not.

## Introduction

Vidal and Fargier (1999) introduced Dynamic Controllability (DC) of Simple Temporal Problems with Uncertainty (STPU), which postpones decisions on time points to enable a dynamic strategy to deal with temporal uncertainty rather than a fixed schedule. Yu, Fang, and Williams (2014) extended the STPU to the Controllable Conditional Temporal Problem with Uncertainty (CCTPU) by considering controllable choices as discrete variables. However, their notion of dynamic controllability of an CCTPU makes a fixed assignment of values to discrete variables, reducing it to an STPU that is dynamically controllable. To implement the original intent of dynamic control, in this paper, we extend it to the discrete variables of the CCTPU.

The CCTPU builds on the from Conditional Temporal Problem (CTP) (Tsamardino, Vidal, and Pollack 2003) and he Simple Temporal Problem with Uncertainty (STPU) (Vidal and Fargier 1999). CTP is an extension of temporal constraint-satisfaction problems adding observation nodes and labels to all non-observation nodes in the network. The label of a node in a CTP represents the situations in which the node will be executed. The STPU model of uncertain events in temporal problems and three levels of controllability were introduced by Vidal & Fargier (1999), among which dynamic controllability (DC) is the most useful one in real situations. Yu et al. (2014) introduced a relaxation method to solve over-constrained CCTPUs, by making a static assignment of choice variables and relaxing time constraints to produce a dynamically controllable STPU. From them, we borrow the idea of using STPU DC checking algorithms (Morris, Muscettola, and Vidal 2001; Morris 2014) to find conflicts, which represent the reason why an STPU is not DC, but extend it to extract all conflicts.

In this paper, we introduce a definition of dynamic controllability of a CCTPU that considers making assignments of both time points and discrete choices dynamically. To implement the DC checking process, some assumptions are needed: (1) each discrete choice is made at one time point, which *prior* to any other time points related to the choice; (2) only the uncontrollable events *definitely* completed before the time point to make choice can be treated as an observable condition; and (3) each discrete choice is made following the observation of one, or a sequence of, uncertain time points. These assumptions are more conservative than the original concept of dynamic controllability. Our DC algorithm is sound, but incomplete in that it finds only dynamic strategies under these assumptions.

Last but not least, a similar problem – dynamic controllability of Conditional Simple Temporal Networks with Uncertainty (CSTNU) – has been studied in (Hunsberger, Posenato, and Combi 2012; Combi, Hunsberger, and Posenato 2014), which consider conditions as uncontrollable and observable propositions. But we only discuss controllable discrete variables that do not depend on observations.

## An Illustrative Example

To illustrate the motivation for dynamic controllability of a CCTPU, we take the example of Mr. P's travel plan after work. After leaving work at 5pm, Mr. P is going grocery shopping before having dinner, then catching a bus home. Shopping may take 30 to 50 minutes, depending on how crowded it is. For dinner, Mr. P has two options: He can have

a quick dinner at KFC, which only takes 20-30 minutes, or go for his favourite steak. This takes longer, 40-60 minutes, but the restaurant is closer to the bus stop. The bus leaves at 6.50pm. Mr. P neither wants to miss this bus, which will make him wait an hour for the next bus, nor arrive at bus stop before than 6.40pm, to avoid staying out in the cold weather. Mr. P needs to decide his schedule for these two hours.

The CCTPU in Figure 1 models Mr. P's problem. The discrete variable $c_1$ models the choice of dinner ($c_1 = K$ for KFC and $c_1 = S$ for steak). Contingent links (dashed lines: $E1 \rightarrow E1'$, $E2 \rightarrow E2'$ and $E3 \rightarrow E3'$) represent uncertainty, such as the time it takes to shop and have dinner. Those durations are not decided by Mr. P but depend on factors outside his control. The other links express constraints on the solution, in the form of bounds on the difference between two time points. Some links have labels, which are the assignments of the discrete choice variable that active those links. For example, constraints $E1' \rightarrow E3$, $E3 \rightarrow E3'$ and $E3' \rightarrow E$ only need to be met if $c_1 = K$.

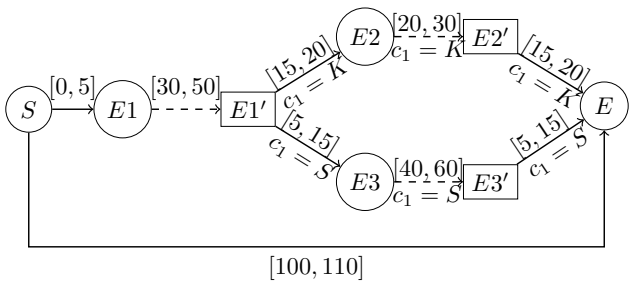



Figure 1: The CCTPU of Mr. P's travel problem.

Unfortunately, because of the uncertainty, neither of the two dinner options, if chosen in advance, lead to a schedule that is guaranteed to satisfy Mr. P's requirements. If he goes to KFC and the uncontrollable links C1 and C2 take their lower bounds, the total duration from S to E is a maximum of 95 minutes, breaking the lower bound of the S–E requirement link. If he goes for steak and links C1 and C3 take their upper bounds, the minimum time from S to E is 2 hours, breaking the upper bound instead. But Mr. P does not have to make the decision about dinner when he leaves work. He only needs to make it after buying groceries. If he spent more than 40 minutes shopping, he can choose KFC; otherwise he can enjoy the steak and catch the bus on time.

Although artificial, the example shows the benefit of postponing decisions, including decisions about discrete choices, when faced with uncertainty. (Indeed, that is the motivation for the concept of dynamic controllability originally proposed for the STPU.) There may be no fixed assignment of discrete variables that is feasible, while making those assignments during execution based on observations of uncontrollable events may provide a feasible strategy.

## Problem Statement

The Controllable Conditional Temporal Problem with Uncertainty (CCTPU) extends the STPU with controllable discrete choices. It was introduced by Yu, Fang, and Williams (2014). We adopt their definition, but omit the reward and cost functions since we consider feasibility only.

**Definition 1.** A **Controllable Conditional Temporal Problem with Uncertainty (CCTPU)** is a 5-tuple $< V, E, C, D, \ell_E >$, where

- $V$ is the set of time points, where $V = VC \cup VU$ and $VU = V \setminus VC$. $VC$ is the set of controllable time points, $VU$ is the set of uncontrollable time points which can be observed,
- $E$ is the set of constraints of form $l_{ij} \leq v_j - v_i \leq u_{ij}$, where $E = EC \cup EU$ and $EU = E \setminus EC$. $EC$ is the set of controllable constraints between pairs of time points, $EU$ is a set of uncontrollable constraints, denoted as contingent links, the exact duration of $eu_{ij}$ is not controllable but within the range $[l_{ij}, u_{ij}]$,
- $C$ is a set of controllable discrete variables,
- $D(c)$ is the domain of variable $c \in C$,
- $\ell_E$ is a mapping that attaches to each link in $E$ a (possibly empty) conjunction of assignments to variables in $C$.

The Controllable Conditional Temporal Problem (CCTP) is a special case of CCTPU where $VU = \emptyset$. Note that when $C = \emptyset$ the CCTPU reduces to an STPU.

The following definitions extend concepts from the theory of controllability of an STPU (Vidal and Fargier 1999; Morris, Muscettola, and Vidal 2001; Hunsberger 2009) to the CCTPU.

**Definition 2.** A **schedule** $S$ for a CCTPU is a tuple $\langle A, T \rangle$. $A$ is an assignment of each discrete variable $c$ to a value in its domain, i.e., $A(c) \in D(c), \forall c \in C$. A link $e \in E$ is **activated** if $A \models \ell_E(e)$. $T$ is a mapping $T : V \rightarrow \Re$, where $T(v)$ is the scheduled time of time point $v \in V$.

A schedule is **consistent** if it satisfies all the constraints of links activated by the assignments of discrete variables.

**Definition 3.** A **projection** $p$ of a CCTPU is constructed by replacing every uncontrollable link $eu_i = [l_i, u_i]$ in $EU$ by the singleton $eu_i = [p_i, p_i]$, where $p_i \in [l_i, u_i]$.

Each projection of a CCTPU is a possible outcome of uncertainties that may occur, and it is a classical CCTP.

**Definition 4.** An **execution strategy** for a CCTPU is a tuple $\langle DT, ES \rangle$, where $DT : C \rightarrow V$ maps each discrete variable $c$ to the time point $DT(c)$ at which the choice for $c$ will be made, and $ES : P \rightarrow S$ is a mapping from the set $P$ of all projections of the CCTPU to the set $S$ of schedules.

An STPU is a special CCTPU without discrete variables, and its *execution strategy* $ES$ is **viable** iff $ES(p)$ is consistent for every projection $p \in P$. Based on the above, Vidal and Fargier (1999) introduced three levels of controllability for the STPU: weak, strong and dynamic. Extending strong controllability to the CCTPU is straightforward:

**Definition 5.** A CCTPU is **strongly controllable** when there is execution strategy $\langle DT, ES \rangle$ such that $DT(c) = 0$ for all $c \in C$, $ES$ is viable, and satisfying $ES(p_1)(x) = ES(p_2)(x)$ and $ES(p_1)(c) = ES(p_2)(c)$ for each controllable time point $x$, discrete variable $c$ and any two projections $p_1$ and $p_2$.

Strong controllability means there is a universal schedule which satisfies all constraints in every projection of the problem. This means the schedule can be made before execution.

Yu, Fang, and Williams (2014) define a dynamically controllable solution of a relaxed CCTPU as a fixed assignment of the discrete variables such that the resulting STPU is dynamically controllable. Specifically, there is a viable execution strategy $\langle DT, ES\rangle$ such that $DT(c) = 0$ for all $c \in C$, and for any two projections $p_1$ and $p_2$, $ES(p_1)\{\prec t\} = ES(p_2)\{\prec t\} \Rightarrow ES(p_1)(x) = ES(p_2)(x)$ for each controllable time point $x$, $t = ES(p_1)(x)$ (Hunsberger 2013) and $ES(p_1)\{c\} = ES(p_2)\{c\}$ for each $c \in C$. That is, decisions on discrete variables are strongly controllable.

## Dynamic Controllability of the CCTPU and Conservative Assumptions

Dynamic controllability means there is a viable execution strategy whose decisions depends only on observations made before the decision. Before we present our definition of dynamic controllability of a CCTPU, we discuss some assumptions we make about the execution strategy.

**Assumption 1.** Assignment $A(c)$ is made once, at the time point $DT(c)$, which must occur no later than any link $e \in E$ such that $\ell_E(e)$ mentions $c$.

It is conservative because if there are more than two possible values for $c$, separate decisions that $c \neq dc_i$ can be made at different time points. For example, if we have the choice of performing a task today, tomorrow, or the day after, we could decide now to not do it today without committing to which of the other two days it will be done. By adopting assumption 1, we may give up some execution flexibility. To some extent, this can be recovered by remodelling the problem. In the example above, the choice between today or any of later days can be represented by one binary variable, and the choice between the other two days by another.

Based on assumption 1, we define a CCTPU execution strategy $\langle DT, ES\rangle$ as **viable** iff (1) $DT(c)$ precedes the start of every link $e \in E$ such that $c$ appears in $\ell_E(e)$ and (2) $ES(p)$ is consistent for every projection $p \in P$.

For dynamic controllability of an STPU, the *observed situation* (Vidal and Fargier 1999), or *prehistory* (Morris, Muscettola, and Vidal 2001; Hunsberger 2009), at any time consists of the observed durations of contingent links that have finished before that time. Given a schedule $S$ of STPU, the prehistory of a time point $x$ is

$$S\{\prec x\} = \{p_{ij} | S(v_i) + p_{ij} \leq S(x)\},$$

where $p_{ij}$ is the observed duration of contingent link $e_{ij}$. However, we restrict the prehistory of discrete choice variables to only those contingent links that must always finish before the variable's decision time point.

**Definition 6.** For any pair of time points $v_i, v_j \in V$, $v_i$ *precedes* $v_j$, $v_i \preceq v_j$, iff $S(v_i) \leq S(v_j)$, for every consistent schedule $S$.

**Assumption 2.** Given a projection $p$, the **prehistory** of a discrete variable $c$ is the observed durations of contingent links which **must** finish before or at $DT(c)$ in every execution, denoted $P_p\{\preceq c\} = \{p_{ij} | v_j \preceq DT(c)\}$.

Furthermore, the **prehistory** of a set of discrete variables is the union of the prehistories of the variables in the set,

$$P_p\{\prec C_s\} = \bigcup_{c \in C_s} P_p\{\prec c\}.$$

Assumption 2 is conservative because contingent links that start, but have not yet ended, before the decision time point of a discrete variable are also observations that could be used in making the decision. In a dynamic strategy for an STPUs, they appear as "wait" constraints (Morris, Muscettola, and Vidal 2001). We now define dynamic controllability of a CCTPU as follows.

**Definition 7.** A CCTPU is **dynamically controllable** if there is a viable execution strategy $\langle DT, ES\rangle$ such that for any two projections $p_1$ and $p_2$, $ES(p_1)\{\prec t\} = ES(p_2)\{\prec t\} \Rightarrow ES(p_1)(x) = ES(p_2)(x)$, where $t = ES(p_1)(x)$, for each controllable time point $x$, and $P_{p_1}\{\prec c\} = P_{p_2}\{\prec c\} \Rightarrow ES(p_1)(c) = ES(p_2)(c)$ for each each discrete variable $c$.

Finally, we make one more assumption about the dynamic execution strategy:

**Assumption 3.** For each discrete variable $c$, $DT(c)$ is end point of a contingent link.

This assumption implies that the observation made before choice $c$ is a single contingent link, or a sequence of contingent links. If two time points are separated by a requirement link, like $t_1$ and $r_1$ in Figure 2, making the decision at $t_1$ enables scheduling different durations for $t_1 \rightarrow r_1$ according to the choice for $c$. Making the decision at $t_2$, on the other hand, while allowing the observed duration of the contingent link(s) after $r_2$ to be used in choosing the value of $c$, also means the preceding controllable events ($r_1$, $r_2$, etc.) must be scheduled the same whichever choice is made for $c$. However, Assumption 3 does rule out strategies that wait for parallel contingent links to finish before making a decision.

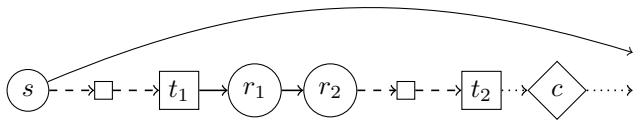



Figure 2: Alternatives for $DT(c)$. Squares are uncontrollable time points, circles controllable time points and the diamond is the latest decision time of $c$.

**Relaxing the Assumptions** Assumptions 2 and 3 are essentially restrictions on the dynamic execution strategies that can be found by the algorithm we present in the next section. Removing them from the definition of dynamic controllability is straightforward (making the prehistory of a discrete variable $S\{\prec DT(c)\}$). Assumption 1 is not as easy. A dynamic strategy has to ensure that discrete choices are made so that there is no ambiguity about the activation of a link when the starting time point of the link is scheduled.

## Approach

Central to our algorithm for finding a dynamic execution strategy is the notion of the "envelope" of a partial assignment of the discrete variables.

**Definition 8.** Given a partial assignment to a subset of discrete variables, $C_{Ass} \subseteq C$, the **dynamically controllable envelope** of an unassigned variable, $c \in (C - C_{Ass})$, is the set of prehistories of $c$ for which there exists a viable dynamic execution strategy.

In other words, the envelope of a decision, given a partial assignment, is the subset of possible outcomes of earlier contingent links for which a viable strategy exists. It is similar to the notion of a relaxation of an over-constrained CCTPU (Yu, Fang, and Williams 2014), which also allows tightening contingent links, but captures all ways of making the subproblem dynamically controllable. An example of a DC envelope, for the problem in Figure 4, is given by equation 8 at the end of the Approach section.

The DC envelope of a set of discrete variables is a combination of the variables' envelopes. For contingent link in the prehistory of two or more variables, all conditions on the link apply, so that the envelope is the intersection. Where two variables have different links in their prehistory, the envelope is defined over the union of their prehistories.

A CCTPU is dynamically controllable if the DC envelope of any partial assignment covers all possible outcomes of uncertainties in the problem. This means the partial assignment can be made statically, and there exists a dynamic strategy for the future (discrete and scheduling) choices. Hence, our approach (1) builds a search tree by expanding on variables, (2) extracts the dynamically controllable envelopes of STPUs at leaves and (3) aggregates those envelopes as the dynamically controllable envelopes of non-leaf nodes. If the DC envelope of any node covers all uncertainty, a dynamic execution strategy can be extracted from this node.

Next we describe the general idea of the approach; the details are introduced in the following four subsections.

### Dynamic Controllability Checking for CCTPU

Our dynamic controllability checking algorithm for a CCTPU (Algorithm 1) is a recursive tree search. Each leaf node is the STPU obtained from a full assignment to discrete variables, while interior nodes are CCTPUs with partial assignments. The root is the original CCTPU. Every other node has one parent that eliminates the assignment to the "latest" variable. The chronological order of variables is defined in the next subsection.

The algorithm traverses the tree depth-first, assigning variables in chronological order. *NextUnassignedVariable* (line 1) returns the next variable to be assigned. If every variable has been assigned, the current node is a leaf (lines 2 – 6); in this case, we extract the conflict resolution constraints that must be satisfied to make the leaf dynamically controllable and record those in $Node.S$ as its DC envelope. *DCEnvelopeSTPU* is a variation of Morris's (2014) DC checking algorithm, which identifies a non-DC STPU by finding a semi-reducible negative cycle, we will call it negative cycle in this paper, during propagation according to dynamic controllability reduction rules. *DCEnvelopeSTPU* expands it by propagating through non-shortest paths and recording all negative cycles. A non-leaf node (lines 7 –13) is expanded by assigning values to the next variable and exploring those nodes recursively. After the child branches of a node have been explored, their DC envelopes are combined and recorded as the DC envelope of current node (line 11). If the envelope of a node is dynamically controllable, then so is the CCTPU and the algorithm returns successfully.

The following subsections explain the three key subroutines: *DCEnvelopeSTPU*, *Union* (line 11), which combines envelopes from branches of the same node, and *isDC* (line 12), which checks if the current solution includes a dynamic strategy for the original problem.

**Input:** A $Node =< N, A, S >$ includes a CCTPU $N$, a vector of assignments $A$ and the solution of the current node $S$.
**Output:** TRUE/FALSE
**Algorithm:** TreeSearch

```
1  c ← NextUnassignedVariable(Node)
2  if isNull(c) then
3  |  Node.S ←DCEnvelopeSTPU(Node)
4  |  if isDC(Node.S) then
5  |  |  return TRUE
6  |  return FALSE
7  for dc_i in D(c) do
8  |  NewNode ← Assign(Node, c, dc_i)
9  |  if TreeSearch(NewNode) then
10 |  |  return TRUE
11 |  Node.S ← Union(Node.S, NewNode.S)
12 |  if isDC(Node.S) then
13 |  |  return TRUE
14 return FALSE
```

**Algorithm 1:** Checking dynamic controllability of a CCTPU.

### Branching Rule

The order in which the algorithm assigns variables obeys the execution process. During execution, a decision can observe and depend on previous assignments. Even when some variables are decided in parallel, the branching rule assigns them in a sequence that respects to this order.

To find the chronological order, we build a dependency tree among variables. If a variable's decision time point is after any links that may be activated by another discrete variable, there is a **dependency** from the earlier to the later variable. For example, in Figure 4 the choice between $c_2 = 1$ and $c_2 = 2$ depends on the choice for $c_1$. Dependencies among discrete variables form a directed tree. A topological sort of this tree provides a **chronological order**, in which every variable does not depend on any variable after it.

### Computing the DC Envelope of an STPU

The DC envelope of an STPU (leaf node) is the observable condition under which the STPU is dynamically controllable. Finding the DC envelope of an STPU is to find the subset of prehistories in which all conflicts can be avoided. This is similar to relaxing an over-constrained (non-DC) problem. Yu, Fang, and Williams (2014) formulate the relaxation

problem as a linear program with a set of constraints derived from the *conflicts* of the STPU, which represent the reasons why it is not dynamically controllable. A conflict is a negative cycle in the network after applying DC reduction rules. They can be represented as follows:

$$\sum_{i \in conf_j} x_i < 0, \tag{1}$$

where $x_i$ are the original bounds ($l_i$ or $u_i$) of links $e_i$ in the conflict. One conflict resolution constraint is

$$\sum_{i' \in conf_j \cap ER} x'_{i'} + \sum_{i \in conf_j \setminus ER} x_i \geq 0 \tag{2}$$

where $ER$ is the set of relaxable links and $x'_{i'}$ are variables for the relaxed bounds. Another way to resolve conflicts with connected lower- and upper-case labels of the same node is to break the reduction since the cross-case reduction rule has a label condition (Morris and Muscettola 2005):

(CROSS-CASE REDUCTION) If $x \leq 0, B \neq C$, $A \xleftarrow{B:x} C \xleftarrow{c:y} D$ adds $A \xleftarrow{B:(x+y)} D$.

For example, in equation 3, the conflict can be resolved by increasing $C \leftarrow D$ to $-9$, which will make back propagation of $C \xleftarrow{-10} D$ stop at $B$ so that the back propagation of $A \xleftarrow{-10} B$ has to stop before $B \xleftarrow{b:1} A$ because it does not satisfy the label condition.

$$A \xleftarrow{B:-10} B \xleftarrow{1} (C \xleftarrow{-10} D \xleftarrow{9} B \xleftarrow{b:1} A) \tag{3}$$

Therefore, the resolution of a single conflict may be a disjunction with one linear constraint of the form 2 and other linear constraints over the links whose relaxations can break the reductions:

$$\bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} + \sum_{i \in res_{jk} \setminus ER} x_i \geq 0 \tag{4}$$

where $res_j$ is the set of conflict resolutions of $conf_j$.

Yu, Fang, and Williams solve an LP over constraints of the form (2) to find a single relaxation. However, provided we find *all* conflicts in the STPU, the set of conflict resolution constraints of the form 4 represents the space of all relaxations, which is the same as the DC envelope. Extracting all conflicts can be done by adapting current DC checking methods (Morris 2006; 2014; Hunsberger 2013; Shah et al. 2007; Nilsson, Kvarnström, and Doherty 2013) and keeping a record of the conflicts.

Our algorithm to find the DC envelope of an STPU builds on the current fastest DC checking algorithm (Morris 2014). In order to find all negative cycles, the Dijkstra propagation is replaced by DFS. It is summarised in Algorithms 2, 3 and 4. Algorithm 4 and Lines 5 – 8 of Algorithm 3 are our DFS process, while the rest is the same as Morris's method.

Algorithm 2 calls $BackPropagation$ once on every negative link. $BackPropagation$ (Algorithm 3) terminates if the current node recursively reached itself through a negative path (lines 1 – 2) or if propagation from current node has already been completed (lines 3 – 4). $dis[i]$ is the distance from $i$ to the end of $source$ in the current round of propagation. In DFS (Algorithm 4), lines 2 – 5 search for negative cycles containing the negative link ending in $n$ in an ancestor call. This link will cause a termination because of being called, which will prevent finding the negative cycle. $NegPathEnds$ keeps a record of negative paths to negative links being called. If a path $e$ in $NegPathEnds[n]$ starts from $source$, a negative cycle consists of $source \xrightarrow{e} n$ and $n \xrightarrow{dis[n]} source$ is added (Line 5). Lines 7 – 8 prevent cross-case reductions that do not satisfy the label condition. As proven by Morris (2014), the algorithm only propagates through non-negative links ending in $n$ (Line 9 – 17). If the propagated path is positive, a new edge is added (Line 12), else, propagation continues to call DFS (Line 16). When encountering a negative link, the algorithm will call $BackPropagation$ again (Line 19). $bReturn$ marks the existence of a recursively called negative link. If the following propagation causes a recursive call, the current path $n \xrightarrow{dis[n]} source$ may cause negative cycles finding which is prevented by termination. Therefore, this negative path and added negative paths ($n \xrightarrow{dis[n]} source + source \xrightarrow{e1} x$) can be used when tracing back to the previous existence of the ancestor call (Line 26).

**Algorithm:** DCEnvelopeSTPU($N$)
**Input:** An STPU $N$.
**Output:** A set of conflicts $NegCycles$

```
1 NegCycles = ∅
2 for e in negative links do
3 |   BackPropagation(e, NegCycles)
4 return NegCycles
```

**Algorithm 2:** Extracting the DC envelope of an STPU.

**Algorithm:** BackPropagation($srcLink, NegCycles$)
**Input:** A negative link $srcLink$, the set of negative cycles $NegCycles$.
**Output:** Return the propagation result and an updated $NegCycles$

```
1 if ancestor call with same srcLink then
2 |   return False
3 if prior terminated call with srcLink then
4 |   return True
5 dis = {infinity}
6 dis[srcLink.start] = srcLink.weight
7 if DFS(srcLink.start, srcLink.end, NegCycles) then
8 |   return True
9 return False
```

**Algorithm 3:** Extracting the DC envelope of an STPU.

The algorithm extracts all conflicts caused by the bounds of the CCTPU with a specific assignment. The solution space of relaxed STPUs that avoid those conflicts is the DC envelope, defined by $x'_{i'}$ satisfying the following constraints:

$$\bigwedge_{j \in Conf} \bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} + \sum_{i \in res_{jk} \setminus ER} x_i \geq 0 \tag{5}$$

The complexity of our conflict extraction method is $O(E^3)$ in the worst case, when all links are negative and

**Algorithm:** DFS($n, source, NegCycles$)
**Input:** Current node $n$, the source of back-propagation $source$ and the set of conflicts $NegCycles$.
**Output:** Return DFS result and an updated $NegCycles$

```
bReturn = True
if n is the end of a negative link in ancestor call then
    for e in NegPathEnds[n] do
        if e.start == source then
            NegCycle.add(e + dis[n])
for e ends with n do
    if e is unusable then
        continue
    if e.weight ≥ 0 then
        NewE = e + dis[n]
        if NewE.weight ≥ 0 then
            addEdge(NewE)
        else
            TmpDis = dis[e.start]
            dis[e.start] = NewE
            bReturn& =
              DFS(e.start, source, NegCycles)
            dis[e.start] = TmpDis
    else
        if !BackPropagation(e, NegCycles) then
            bReturn = False
            NegPathEnds[n].add(dis[n])
            for e1 in NegPathEnds[source] do
                if e1 starts at n then
                    continue
                else
                    NegPathEnds[n].add(e1 + dis[n])
return bReturn
```

**Algorithm 4:** Extracting the DC envelope of an STPU.

the network is fully connected. It is slower than the state-of-art DC checking methods and CDRU. However, it returns the complete set of conflict resolution constraints, not only whether conflicts exist, or one relaxation.

## Combining DC Envelopes

The combining process aims to answer under which condition an assignment of the current discrete variable, $c$, can be made at its decision time point $DT(c)$ such that the future part of the problem is dynamically controllable. This condition is the DC envelope of the current node.

The envelope of an STPU is a conjunction of conflicts, as in equation (7). In combining envelopes, we may split observable uncertainty, which is before $DT(c)$, so that each child branch only tackles part of it. This may resolve conflicts, or at least relax them, meaning less strict constraints on the prehistory may replace the original conflict resolution. Therefore, a conflict can be resolved through several combining processes.

Ideally, dynamic choices should be based on the whole prehistory before $DT(c)$. Conflicts with different paths have different prehistories before $DT(c)$. Recall our Assumption 3, that $DT(c)$ is the end point of a contingent link. Since no more than one contingent link can finish at a node, this means we only consider one contingent link (or the sum of sequence of contingent links) as the latest observation in each combining process.

The envelope of the node of assigning discrete variable $c$ is a disjunction of child nodes' envelopes:

$$\Phi = \bigvee_{dc_l \in D(c)} \bigwedge_{j \in Conf} \bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl}, \quad (6)$$

where $ER = \{e_i \in EU \cap P_p(\preceq c)\}$ and $a_{jkl} = -\sum_{i \in res_{jk} \setminus ER} x_i$ is the sum of bounds of links after $DT(c)$.

The envelope can be expanded into a conjunction of disjunctions, as shown in equation (7). Each conjunct takes one conflict resolution (which is a disjunction of linear constraints) from every child node branch in equation (6), so the number of conjuncts can be the product of the number of constraints in each child node's envelope. Transforming the envelope into this form, same as that of equation (5), makes the combining process uniform at all levels of the tree.

$$\Phi = \bigwedge \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl} \quad (7)$$

For each branch of the conjunction (7), either the contingent link observed just before the decision time point is part of each constraint in the disjunction, or it is not. In the latter case, the branch can not be combined by splitting uncertainty between the disjuncts. We just keep those constraints to the next combining process. In the former case, however, the uncertainty of the common contingent links can be shared by different child node envelopes. We try the end node of every contingent link that precedes the latest decision time of $c$ as $DT(c)$, replacing the variables representing its lower and upper bounds by a single variable $x_{ij}$. Then, the envelope answers under which condition for all $x_{ij} \in [l^c_{ij}, u^c_{ij}]$ there exists a choice $dc_i$ such that $x_{ij}$ satisfies constraints in the envelope of the child node with $c = dc_i$.

A branch can be removed from the conjunction when it covers the whole uncertainty, which means its negation is infeasible within the original bounds of the problem. As the negation is a conjunction of linear constraints, we can use an LP solver to perform this test.

Following our assumptions, the DC envelope (1) is represented by constraints on bounds of links prior to the decision time point of the variable, (2) enables different assignments to the variable, which also implies different following schedules, and (3) contains a dynamic strategy over the prehistory that is consistent for different assignments.

The reduction of a disjunction of constraints with a common contingent link is illustrated in in Figure 3. Figure 3(a) shows an easy situation, where $E^I_c = \{[-Inf, a]\}$, $E^{II}_c = \{[b, Inf]\}$ and $a \geq b$, so the union of envelopes I and II covers the whole space. This means that for all situations in the prehistory, no matter what is observed, there will always be a feasible option that leads to a dynamic strategy. Figure 3(b) illustrates a general and feasible situation. Envelopes I and II covers the area from the lower bound of $E^I_c$ to the upper bound of $E^{II}_c$. The intersections of the observation bounds $[l_c, u_c]$ with the edges are $p_1$ and $p_2$. The combined envelope indicates that if the prehistory is within

$[l_p, u_p]$, there will be a feasible choice for $c$. Figure 3(c) illustrates an infeasible situation. The uncertainty is so large that the combined envelope cannot provide a viable range for the prehistory before the observation.

## DC Checking for the Combined Envelope

This subsection aims to answer what kind of combined envelopes means the problem is solved or unsolvable.

A node is dynamically controllable if its DC envelope covers all uncertain situations implied by the problem. If the negation of an envelope is infeasible within the original bounds of the problem, which means every uncertain situation can be solved within one disjunctive branch, the problem is dynamically controllable. The envelope is disjunctive, a separate check is done for each disjunct.

If the problem is not solved during the combining process at any interior node of the search tree and the DC envelope of the root is still not dynamically controllable, then a dynamic strategy satisfying our assumptions does not exist.

We explain the combining process and node checking with an example with two discrete variables, shown in Figure 4. This problem cannot be solved with a fixed assignment, because the STPU in each leaf node contains conflicts.

The algorithm first arrives at the leaf $\{c_1 = 1, c_2 = 1\}$ and extracts the conflict

$$A \xrightarrow{b:1} B \xrightarrow{1} C \xrightarrow{d_1:1} D_1 \xrightarrow{1} E \xrightarrow{f_1:1} F_1 \xrightarrow{1} G \xrightarrow{-10} A$$

Then in leaf $\{c_1 = 1, c_2 = 2\}$, the conflicts are

$$\begin{aligned}
(1)\quad & A \xleftarrow{B:-10} B \xleftarrow{0} C \xleftarrow{D_1:-10} D_1 \xleftarrow{0} E \xleftarrow{F_2:-40} F_2 \\
& \xleftarrow{0} G \xleftarrow{50} A \\
(2)\quad & A \xrightarrow{b:1} B \xrightarrow{1} C \xrightarrow{d_1:1} D_1 \xrightarrow{1} E \xrightarrow{f_2:2} F_2 \\
& \xrightarrow{1} G \xrightarrow{-10} A
\end{aligned}$$

Potential $DT(c_2)$ is $B$ or $D_1$. Because making decision of $c_2$ at $B$ will never satisfy the branch $\{c_1 = 1, c_2 = 2\}$, we illustrate the solution of making decision at $D_1$. The envelope at node $\{c_1 = 1\}$ is the solution space of the following constraints:

$$\vee \begin{cases} \begin{array}{rl} l'_{AB} + u'_{BC} + l'_{CD_1} + u'_{D_1E} & \\ \quad + l_{EF_1} + u_{F_1G} - l_{AG} & \geq 0 \end{array} \\ \begin{array}{rl} -u'_{AB} - l'_{BC} - u'_{CD_1} - l'_{D_1E} & \\ \quad - u_{EF_2} - l_{F_2G} + u_{AG} & \geq 0 \end{array} \\ \begin{array}{rl} l'_{AB} + u'_{BC} + l'_{CD_1} + u'_{D_1E} & \\ \quad + l_{EF_2} + u_{F_2G} - l_{AG} & \geq 0 \end{array} \end{cases} \tag{8}$$

With variables $x$ representing observations and $p$ representing preconditions, the constraints are expanded as

$$\wedge \begin{cases} p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 & (c_2 = 1) \\ \vee \quad p'_{AB} + p'_{BC} + x'_{CD_1} \leq 10 & (c_2 = 2) \\ p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 & (c_2 = 1) \\ \vee \quad p'_{AB} + p'_{BC} + x'_{CD_1} \geq 6 & (c_2 = 2) \end{cases} \tag{9}$$

The first branch in equation 9 can be cut, because its negation causes infeasibility. The other branch's negation is still feasible, which means it does not cover all uncertainty in its prehistory. Recovering variables to their original bounds, the left constraint $l'_{AB} + u'_{BC} + l_{CD_1} \geq 6$ must be kept to the next combining process. In the next step, our algorithm explores node $\{c_1 = 2\}$. Using the same process, the envelope of node $\{c_1 = 2\}$ is $\{u'_{AB} + l'_{BC} + u_{CD_2} \leq 20\}$. The decision time point is $DT(c_1) = B$, the observation is $A \xrightarrow{[1,10]} B$ and the prehistory before the observation is empty. The combining process results

$$\begin{array}{rlll} & p_\emptyset + x_{AB} & \geq 4 & (c_1 = 1) \\ \vee & p_\emptyset + x_{AB} & \leq 4 & (c_1 = 2) \end{array}$$

whose negation is infeasible. The problem is dynamically controllable.

The dynamic strategy is: (1) if $p_{AB} \geq 4$ at $B$, then make assignment $c_1 = 1$, otherwise $c_1 = 2$; (2) after choosing $c_1 = 1$, $C$ is scheduled based on $p_{AB}$, so that $AD_1$ will always be longer than 6. At $D_1$ if $p_{AD_1} \geq 7$, then choose $c_2 = 1$, if $p_{AD_1} \in [6, 10]$, choose $c_2 = 2$, (3) after choosing $c_1 = 2$, $C$ is scheduled immediately after $B$, and $c_2 = 1$ at $D_2$, (4) the scheduling of other controllable time points are inferred by the reduction rules of dynamic control.

**Soundness and Completeness** If the algorithm finds a node whose envelope passes the DC check, then the CCTPU is dynamically controllable. The strategy is to make choices according to the partial assignment statically, and following the observation of one of the contingent links for the remaining variables. It is valid within the bounds on the prehistory, which are verified by the final DC check to contain all potential outcomes uncertain links. However, the algorithm is not complete, in the sense that it will find only strategies that meet Assumptions 2 and 3.

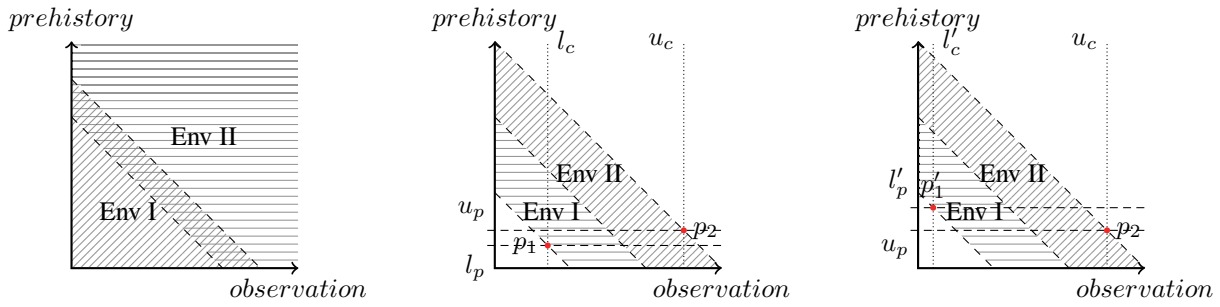
# Experimental Results

We illustrate DC checking for CCTPU by comparing implementations of DC checking methods for the CCTPU with and without dynamic discrete choices. The DC checking method for the CCTPU which does not make assignments to discrete variables dynamically only considers scheduling time points dynamically. Our implementation of this method checks every leaf node (full set of assignments) and considers the CCTPU dynamically controllable if there is one leaf that induces a dynamically controllable STPU .

## Experimental Setup

We use the benchmark generator (Yu 2016) based on Zipcar problems (Yu and Williams 2013; Yu, Fang, and Williams 2014). Its application background is a car-sharing network. Each test case consists of missions with temporal requirements, each mission has a sequence of activities and each activity can be done by choosing one option. An option contains controllable and uncontrollable links. All links are represented by their lower and upper bounds.

We use discrete variables to represent the choices for activities and attach assignments as labels to the links of each option. All temporal links are randomly generated except for the requirement on the overall duration of the mis-




(a) An feasible example (b) A possibly feasible example (c) An infeasible example

Figure 3: Combined Envelope

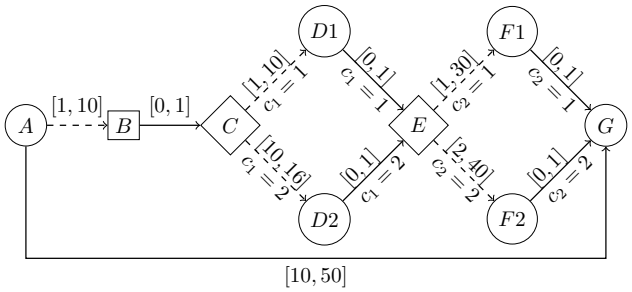



Figure 4: An example of CCTPU with two discrete variables

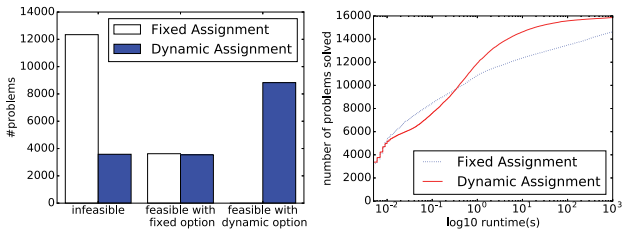



Figure 5: Results

sions, which randomly deviates by $\pm 20\%$ from the estimated bounds of the sequence of activities.

There are 16000 test cases ranging from 1–8 discrete variables with 1–10 options for each variable. In terms of the size of networks, the number of nodes, links and contingent links are varying from 11–170, 11–330 and 4–162, respectively.

## Results

The result is shown in Figure 5. The tests are grouped by DC checking results in the a chart. The white bars represent the result of implementation with fixed assignment. $77.1\%$ of the test cases are infeasible when using only fixed assignment. But only $22.4\%$ are still infeasible while assigning discrete variables dynamically. The number of feasible results with fixed assignment is slightly fewer with implementation of dynamic assignment, because a combined solution can be found earlier in some test cases which contains a dynamically controllable STPU with fixed assignment.

The runtime difference between two implementations is not obvious. It shows that making assignment dynamically does not cost much extra runtime. Furthermore, the implementation with dynamic assignment solves slightly more problems in time limitation above 1 second, which can be inferred that test cases that have dynamic strategy with dynamic assignment terminate before exploring the whole search tree.

## Conclusion and Future Work

In this paper, we extend dynamic controllability to CCTPU with making assignment to discrete variables dynamically. Comparing to the previous work of making assignment statically, some test cases in the current CCTPU benchmarks are dynamically controllable with dynamic decisions on discrete variables but not dynamically controllable with fixed assignment.

In future work, we will try to remove Assumptions 2 and 3, which limit the dynamic strategies that our algorithm can find. Another extension is to see how much improvement can be made in solving optimisation problems over a CCTPU when considering making choices dynamically.

## References

Combi, C.; Hunsberger, L.; and Posenato, R. 2014. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In *Proc. 5th International Conference on Agents and Artificial Intelligence (ICAART)*, 314–331.

Hunsberger, L.; Posenato, R.; and Combi, C. 2012. The dynamic controllability of conditional STNs with uncertainty. In *Proc. Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) Workshop*, 2–4.

Hunsberger, L. 2009. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *Proc. 16th International Symposium on Temporal Representation and Reasoning (TIME)*, 155–162.

Hunsberger, L. 2013. Magic loops in simple temporal networks with uncertainty. In *Proceedings of the Fifth International Conference on Agents and Artificial Intelligence (ICAART)*, 332–350.

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

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proc. 17th International Conference on Artificial Intelligence (IJCAI)*, 494–499.

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

Morris, P. 2014. Dynamic controllability and dispatchability relationships. In *Proc. 11th Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, 464–479.

Nilsson, M.; Kvarnström, J.; and Doherty, P. 2013. Incremental dynamic controllability revisited. In *Proceedings of the 13th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, 337–341.

Shah, J. A.; Stedl, J.; Williams, B. C.; and Robertson, P. 2007. A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, 296–303.

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

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

Yu, P., and Williams, B. 2013. Continuously relaxing overconstrained conditional temporal problems through generalized conflict learning and resolution. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2429–2436.

Yu, P.; Fang, C.; and Williams, B. C. 2014. Resolving uncontrollable conditional temporal problems using continuous relaxations. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 341–349.

Yu, P. 2016. BCDR Test Generator. https://github.com/yupeng/BCDRTestGenerator.