Journal of Artificial Intelligence Research 64 (2019) 445–495 Submitted 03/18; published 02/19

Dynamic Controllability of Controllable Conditional Temporal Problems with Uncertainty

Jing Cui CUI.JING@ANU.EDU.AU
Patrik Haslum PATRIK.HASLUM@ANU.EDU.AU
Research School of Computer Science,
Australian National University
& Decision Sciences Programs, DATA61, CSIRO
108 North Rd, Acton ACT 2601, Australia

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 complete only under certain assumptions. 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.

1. Introduction

For an autonomous system to operate in real problems, its execution has to be flexible to deal with uncertain situations. In the research of temporal planning and scheduling, flexible solutions such as Partial Order Plans (Weld, 1994; Muise, Beck, & McIlraith, 2016) and Partial Order Schedules (Policella, Cesta, Oddi, & Smith, 2007) instead of fixed schedules or plans enables dynamic execution to deal with the uncertain situations (Muise, 2014).

On behalf of providing flexible solutions, adding uncertainty into temporal reasoning models such as Simple Temporal Problems (Dechter, Meiri, & Pearl, 1991) enables us to study uncertainty directly. Vidal and Fargier (1999) introduced Simple Temporal Problems with Uncertainty (STPU) that assume the durations of uncontrollable events can be represented by intervals. During execution, the exact durations may be any value within their intervals. When the durations of the uncertain events are not observable, it requires a universal solution that can deal with any uncertain situations. If such a solution exists, the problem is strongly controllable (SC). Additionally, if the uncertain durations are observable after they have finished, this enables a dynamic strategy to adjust for different past situations. If such a dynamic strategy exists, the problem is dynamically controllable (DC). A dynamic control strategy of an STPU postpones decisions on time points to allow more flexibility to deal with temporal uncertainty than a fixed schedule does.

Scheduling and planning problems contain more decisions than deciding when to schedule each time point, so different extensions of the STPU have been introduced for different application purposes. The pSTN (Tsamardinos, 2002) extends the STPU by representing uncertainties as probabilistic distributions instead of intervals. The Disjunctive Temporal Network with Uncertainty (DTNU) (Venable & Yorke-Smith, 2005; Peintner, Venable, & Yorke-Smith, 2007) builds on the DTN (Stergiou & Koubarakis, 2000) and STPU, and represents each *free constraint* as a disjunction of any controllable constraints and each *contingent link* as a disjunction of intervals for the same pair of controllable and uncontrollable nodes. From the Conditional Temporal Problem (CTP) (Tsamardinos, Vidal, & Pollack, 2003), which attaches labels to nodes to represent the situations in which the node will be executed, the Conditional STNU (Hunsberger, Posenato, & Combi, 2012) and Controllable Conditional Temporal Problem with Uncertainty (CCTPU) (Yu, Fang, & Williams, 2014) were introduced by adding uncertainty. The Conditional STNU (CSTNU) extends to uncertain conditions that are not controllable but observable after a specific observation time point; after the observation, certain links attached to observed conditions are activated. Yu et al. (2014) extended the STPU to the Controllable Conditional Temporal Problem with Uncertainty (CCTPU) by considering controllable choices as discrete variables. In a CCTPU, links with a label, which is a partial assignment of the discrete variables, are activated by the conjunction of those assignments. Barták and Čepek (2007) introduced Temporal Networks with Alternatives (TNAs) which can also represent controllable discrete choices, but not temporal uncertainty. TNAs, however, are restricted and can not represent the same range of choice structures as the CCTPU. Yu, Fang & Williams (2014) introduced a relaxation method to solve over-constrained CCTPUs, by making a static assignment of the discrete choice variables and relaxing time constraints to produce a dynamically controllable STPU. However, their notion of dynamic controllability of a CCTPU makes a fixed assignment of values to discrete variables, reducing it to an STPU that is dynamically controllable.

In this paper, to implement the original intent of dynamic control, we introduce a definition of dynamic controllability of a CCTPU that considers making assignments of both time points and discrete choices dynamically by extending it to the discrete variables of the CCTPU. From Yu et al. (2014), we borrow the idea of using dynamic controllability checking algorithms for the STPU (see Section 4.1) to find conflicts, which represent the reason why an STPU is not dynamically controllable, but modify their iterative process which can find a single best solution but not the whole solution space. From Morris (2014), we borrow the procedure of propagating negative links backwards to find conflicts, but instead of stopping when finding one conflict, we extend it to extract a complete set of conflicts.

When implementing the dynamic controllability checking process, some assumptions are needed: (1) each discrete choice is made at one time point, which is *no later than* any other time point that depends on the choice; (2) only the uncontrollable events *definitely* completed before the time point to make a choice can be treated as observable conditions; and (3) each discrete choice is made following the observation of one, or a sequence of, uncertain events. These assumptions are more conservative than the original concept of dynamic controllability. Thus, our dynamic controllability checking algorithm for CCTPUs is sound but only complete with respect to these assumptions. It is guaranteed to a find dynamic strategy under these assumptions.

1.1 Contributions

The following are the main contributions of this paper:

- We present a definition of dynamic controllability of both temporal and discrete choices; this definition is essentially the same as the one we have previously proposed (Cui & Haslum, 2017) but reformulated to make it clearer and more rigorous.
- We provide a sound and complete algorithm to extract the dynamically controllable envelope of an STPU. We do this by modifying Morris's cubic algorithm to extract a complete set of DC conflicts, such that the solution space of the conflict constraints is the dynamically controllable envelope.
- We propose a sound algorithm to find dynamically controllable envelopes by aggregating the DC envelopes of fully or partially assigned CCTPUs that represent different choices for the controllable discrete variables. If the DC envelope of any fully or partially assigned CCTPU covers its prehistory, the problem is dynamically controllable. Additionally, the algorithm is complete with respect to our assumptions.
- We demonstrate that making dynamic assignments enables more flexibility than fixed assignments using a set of benchmark problems due to Yu et al. (2014).
- We also test a simple optimisation method on CCTPUs under dynamic controllability and further show that dynamic assignments enable better objective value than fixed assignments. We do this by adding a binary search on the objective value over the DC checking algorithm.

The work in this paper extends the conference publication by Cui and Haslum (2017). We have refined the definition of dynamic controllability of the CCTPU, added proofs of correctness of the algorithms and an optimisation method, and expanded the discussion of previous work on related temporal reasoning problems.

1.2 Organisation

We start with motivations in Section 2 by showing a simple but illustrative example. In Section 3 we present the problem statement, which includes the definitions of the CCTPU, dynamic assignment of discrete variables, our assumptions, and dynamic controllability of the CCTPU. We briefly review the dynamic controllability checking algorithms and the conflict resolution of STPU in Sections 4.1 and 4.2, which form the basis of our algorithm that extracts a complete set of conflicts (Section 4.3) in order to find the dynamically controllable envelope of an STPU (Section 4.4). In Section 5, we show how to check dynamic controllability of a CCTPU by aggregating and checking its dynamically controllable envelopes. Section 6 shows the correctness and completeness of the algorithms. Section 7, 8 and 9 present experiments, related work and conclusions, respectively.

2. Motivation

In this paper, we focus on solving CCTPU, which is a temporal problem with uncertainty and controllable choices. This model has been used in several applications, for example trip planning for Zipcar ride sharing (Yu et al., 2014), control of Autonomous Underwater Vehicles (AUV) (Timmons & Williams, 2015), and controlling traffic on the Red Line subway in Boston (Yu, Williams, Fang, Cui, & Haslum, 2017). In this section, we present a simple example, based on a new application, to show the motivation for dynamic controllability of a CCTPU.

2.1 Illustrative Example – Evacuation Planning

The evacuation planning problem (Even, Pillac, & Van Hentenryck, 2014) studies how to find a feasible or optimal evacuation plan, when an area faces a serious disaster (major flood or fires). Figure 1 shows a small example: Each cube represents a unit (vehicle) to evacuate before the flood approaching from the northeast corner of the city cuts off the roads. A successful plan must evacuate all or most units to safe places before roads are blocked or destroyed by the disaster. Evacuation is done by areas (such as suburbs). An evacuation plan assigns each area a specific route, and a scheduled time to begin evacuation. Vehicles from the area will form a traffic flow along their assigned route. Due to the limited road capacity and the limited number of alternative routes, traffic flows from different areas may need to be scheduled to follow one another along overlapping routes. How to schedule the evacuation times so that traffic flows efficiently is a challenging problem.

After generating different routes, evacuation planning can be represented as a temporal network with route options. In the temporal network, time points represent events, such as when to begin evacuation of each region, when the flow of vehicles from each region enter and finish passing a road segment and when each road will be made impassable by the approaching flood. The durations among pairs of the events are modeled by temporal constraints.

In the evacuation planning problem shown in Figure 1, units in Area A are going to evacuate through road G and units in Area B have two options, either evacuate through G after the flow from A has left it, or to evacuate via a longer route G'. To the people who organise the evacuation, it is uncontrollable when exactly the flood will block each road and how fast the traffic will flow. However, the duration of these uncontrollable events can be estimated within some intervals, in other words, they can be modelled by contingent links with lower and upper bounds. During execution, the exact duration may be any value within the bounds. Therefore, we can study whether there exists solutions that can deal with the uncertain durations.

Figure 2 shows a temporal network with uncertainty and options for the example in Figure 1:

- the controllable nodes (ellipses) are Start, Evacuate A and Evacuate B;
- the uncontrollable nodes (rectangles) represent when the traffic flow from area A/B reaches and leaves road G/G', and when G/G' are blocked;
- the uncontrollable events (dashed lines) are the travel time of traffic flows and the flood;

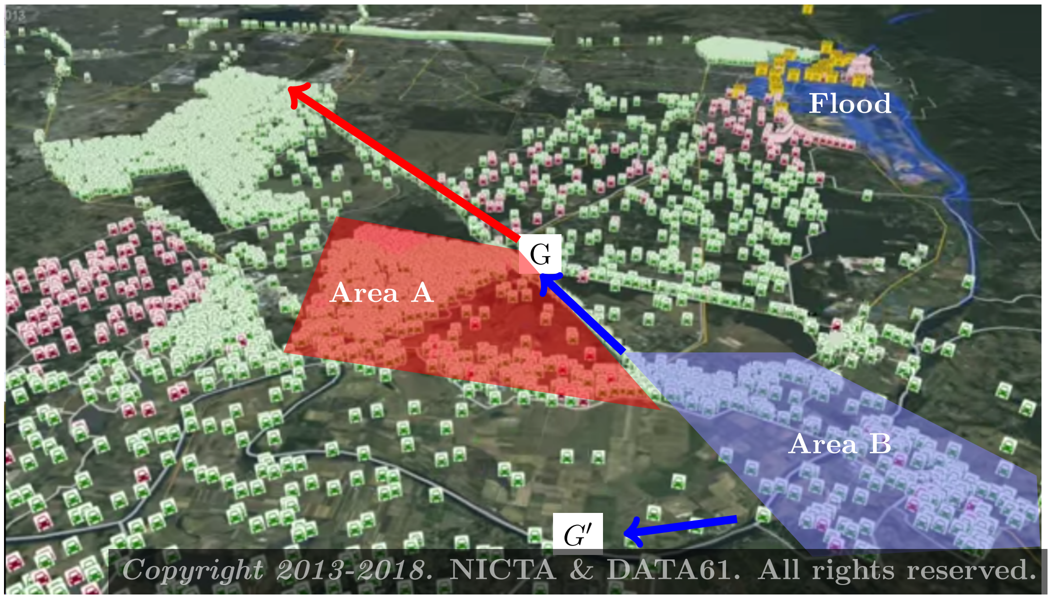


Figure 1: An evacuation situation for flood, generated by the NICTA Evacuation Planning Simulator (Even et al., 2014). (Image from https://research.csiro.au/data61/nicta-evacuation-planner/)

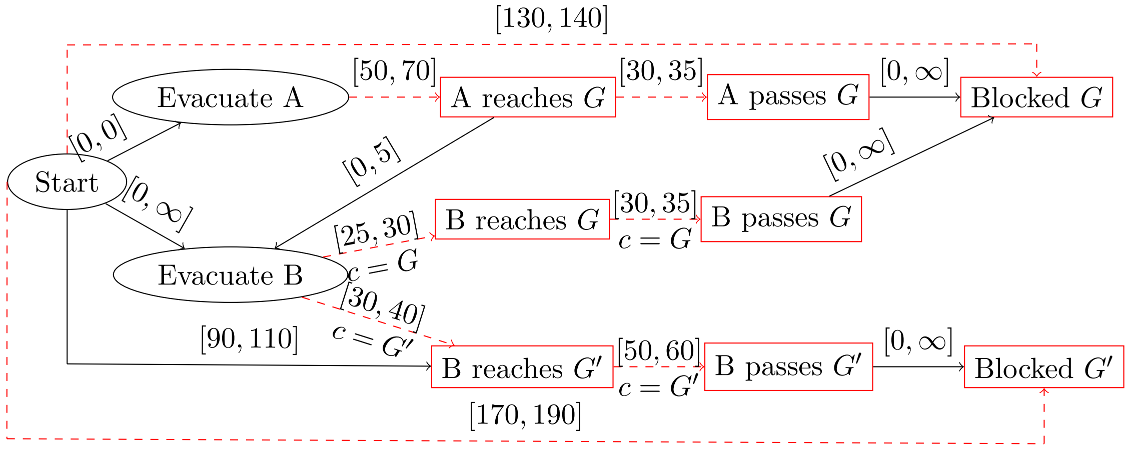


Figure 2: A CCTPU representing the Evacuation Planning example.

- the choice for evacuating B through road G or G' is represented by the discrete variable $c \in \{G, G'\}$.

- To coordinate with the evacuation of other suburbs, we have the extra constraints that (1) area B will begin evacuation after the flow from A reaches G so that the traffic flows won't cause a congestion at G, but with a waiting time less than 5 minutes, and

(2) the time window for the flow from B to arrive at G', if taking that route, is 90 to 110 minutes after the start of the evacuation plan.

- The other controllable constraints (solid lines) are precedence constraints.

If the choice of route for evacuating Area B has to be made before the start of execution, neither option can satisfy all constraints. If we assign B to go through G, units from Area B may still be on road G during $[130, 135]$ minutes after the start of the evacuation when this road is cut off, if the worst case happens. Figure 3 shows the subnetwork for this option, with the conflicting temporal constraints highlighted.

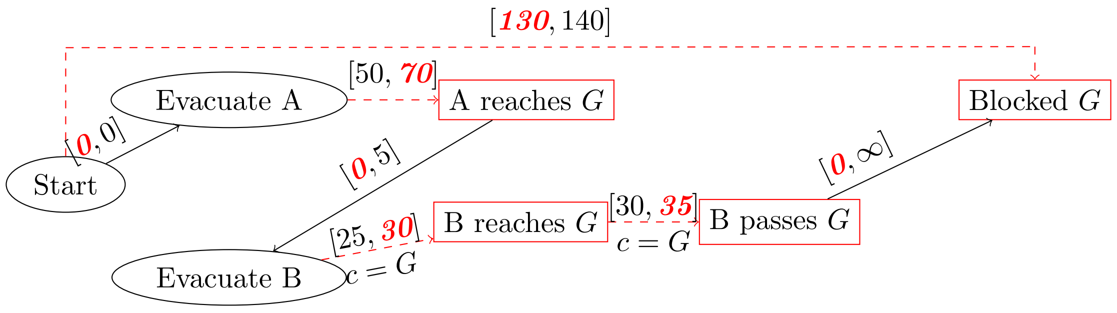


Figure 3: The subnetwork representing the option of Area B evacuating through route G, with the conflicting temporal constraints highlighted in red.

Otherwise, if we assign B to go through G', the traffic flow from B may arrive at G' in the interval $[85, 90]$ minutes after the start of the evacuation and cause a traffic congestion at G' since it arrives too early. This case is highlighted in Figure 4.

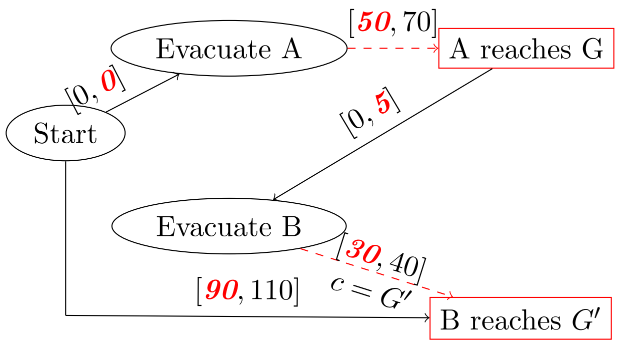


Figure 4: The conflict occurring in the subnetwork for the option of Area B going through route G'.

However, making a dynamic decision for the discrete variable c after observing the *key event* "A reaches G" can avoid both infeasible cases because the two options share the uncertainty in their common *prehistory*. When the duration of the contingent link from "Evacuate A" to "A reaches G" is no more than 65, the constraint violation shown in

Figure 3 won't occur, so that we can assign B to go via G. When the duration of the link is 55 or greater, the conflict shown in Figure 4 is avoided, so that B can go through G'. Furthermore, the conditions "no more than 65" and "no less than 55", which make each option feasible, cover the whole interval of uncertainty for the link "Evacuate A"–"A reaches G"; in other words, no matter when the key event occurs, a valid option will always exist to be chosen after it finishes. This example, although simple, shows that a CCTPU may have a feasible solution with dynamic choice of options for the discrete variables also when it does not have a feasible solution with a static choice of assignments. This is the motivation for studying fully dynamically controllable strategies for CCTPUs.

3. Problem Statement

In this section, we introduce definitions leading up to the dynamic controllability of Controllable Conditional Temporal Problems with Uncertainty (CCTPU). The CCTPU extends the STPU with controllable discrete choices (Yu et al., 2014). We adopt Yu et al.'s definition of the CCTPU, but omit the reward and cost functions since, in this paper, we consider its controllability only. The remainder of this section defines the solution of a CCTPU and different levels of controllability of the CCTPU.

3.1 Preliminary Definitions

An STPU is a constraint satisfaction problem over real-valued time point variables, with constraints that are (upper and lower) bounds on the differences between pairs of variables. However, some time points are *uncontrollable*, meaning that in any execution of the schedule or plan, their values will be chosen non-deterministically (by the environment) within the given bounds, while the values of remaining time point variables are selected by the executing agent, subject to the constraints.

Definition 1. An **STNU or STPU** (Vidal & Fargier, 1999) is a tuple, $< V, E >$.

- V is a set of nodes $V = V_E \cup V_U$, representing executable (V_E) and uncontrollable (V_U) time points,
- $E = ER \cup EC$ is a set of links, called *requirement* and *contingent* links. The contingent links are links ending in an uncontrollable time point, and there is only one contingent link ending in each such time point.
- Each link e_{ij} has a lower bound L_{ij} and upper bound U_{ij}, representing the constraints $L_{ij} \leq t_j - t_i \leq U_{ij}$. For contingent links, $0 < L_{ij} < U_{ij}$.

In order to provide solutions that can be executed more robustly, the CCTP with uncertainty was introduced to solve over-constrained temporal problems with uncertainty and choices (Yu et al., 2014).

Definition 2. A **Controllable Conditional Temporal Problem with Uncertainty (CCTPU)** is a 5-tuple $< V, E, C, D, \ell_E >$, where

- $< V, E >$ is an STPU,

- C is a set of controllable discrete variables,
- $D(c)$ is the domain of variable $c \in C$,
- ℓ_E is a mapping that attaches to each link in E a (possibly empty) conjunction of assignments to variables in C.

Note that when $C = \emptyset$ the CCTPU reduces to an STPU.

Before defining the dynamic controllability of a CCTPU, we introduce the definitions of schedules, execution strategies and strong controllability of a CCTPU. The following definitions extend concepts from the STPU (Vidal & Fargier, 1999; Morris, Muscettola, & Vidal, 2001; Hunsberger, 2009) to the CCTPU.

Definition 3. 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 $A \models \ell_E(e)$; A link $e \in E$ is **deactivated** if $A \not\models \ell_E(e)$.
- T is a mapping $T : V \rightarrow \Re^+ \cup \{0\}$, where $T(v)$ is the scheduled time of time point $v \in V_E$.

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

Although a schedule is defined as a pair of functions, we will, with slight abuse of notation, treat it as just one function assigning values to both discrete variables and time points. Thus, if $S = \langle A, T \rangle$ is a schedule, we may write $S(c)$, meaning $A(c)$, for a discrete variable c and $S(x)$, meaning $T(x)$, for a time point x.

Definition 4. 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 CCTP.

Definition 5. An **execution strategy** for a CCTPU is a tuple $\langle DT, ES \rangle$, where

- $DT : C \rightarrow V \cup \{0\}$ maps each discrete variable c to the time point $DT(c)$ at which the choice for c will be made, the range of $DT(c)$ is the union of all nodes and the beginning time of the network 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 6. 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 $ES(p_1) = ES(p_2)$ for any two projections p_1 and p_2.

In other words, strong controllability means there is a universal schedule which satisfies all constraints in every projection of the problem, and the strong execution strategy maps every projection to this schedule. This means the schedule can be made before execution.

Yu et al. (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. Relaxing an STPU means tightening contingent links (reducing uncertainty) and/or loosening requirement links. The cost of a relaxation is a function of the change to each link, and some links may be excluded from change. Specifically, Yu et al.'s notion of a dynamic controllability of a CCTPU, expressed in our terms, means that 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, $t = ES(p_1)(x)$ and
  - $ES(p_1)(c) = ES(p_2)(c)$ for each discrete variable $c \in C$.

That is, the decisions on discrete variables are strongly controllable.

Definition 12 extends the concept of dynamic controllability to both the discrete and temporal variables of the CCTPU. Before we can state it, however, we present some preliminary definitions and state our assumptions in the following subsection.

3.2 Dynamic Assignments for Discrete Variables

In this subsection, we define the dynamic assignment and prehistory for discrete variables in a CCTPU. The basic idea of dynamic assignments is to assign values to discrete variables based on past observations as dynamic decisions on time points.

As the first thing, we can define the dynamic assignment based on previous definitions.

Definition 7. A **dynamic assignment** $A(c) = dc_i$ is made at $DT(c)$ and it deactivates links with labels $c = dc_j, \forall dc_j \neq dc_i$.

In order to make the definition valid in a dynamically controllable strategy, we have to define the prehistory of the assignment and restrict $DT(c)$ with respect to the links having labels mentioning c.

However, several obstacles prevent us from giving a direct definition of the prehistory: (1) the assignment is not explicitly associated to time points; and (2) the contingent links included in the past observations may be completely different for different choices that have been made. Due to these difficulties, we have to introduce basic definitions, such as *partial assignments, precedences and conditional precedences*, along with our assumptions.

Making dynamic choices for controllable discrete variables in a temporal problem is not a new topic. Conrad and Williams (2011) define the dynamic execution for an STN with controllable discrete variables. This dynamic execution decides to activate an event or not in real-time and maintains a consistent labelled value set. We adopt their definition of environment for the STN with discrete variables as a partial assignment of discrete variables.

Definition 8. (Partial Assignment) The environment is a **partial assignment** PA of the discrete variables. Before execution, the environment is empty $PA = \emptyset$; after a feasible execution, the environment consists of assignments of all discrete variables $PA = A$.

In execution, the dynamic decisions on discrete variables in a strategy are made at specific time points that obey the chronological orders implied by constraints. The observations of contingent links on which these decisions can depend have to complete before the decision time points. As a necessary condition, we introduce the definition of precedence between time points in order to represent the set of contingent links that each decision of discrete variable can observe. Furthermore, the precedence definition can also help to describe and find the chronological order to assign discrete variables.

Definition 9. (Precede) For any pair of time points $v_i, v_j \in V$, v_i *precedes* v_j, denoted as $v_i \preceq v_j$, iff $S(v_i) \leq S(v_j)$, for every consistent schedule S. For any link $e_i \in E$ and $v_j \in V$, e_i *precedes* v_j, $e_i \preceq v_j$, iff $\mathit{start}(e_i) \preceq v_j$ and $\mathit{end}(e_i) \preceq v_j$; v_j *precedes* e_i, $v_j \preceq e_i$, iff $v_j \preceq \mathit{start}(e_i)$ and $v_j \preceq \mathit{end}(e_i)$.

This definition excludes some labelled precedences. For instance, in the example in Figure 5, precedences $A \preceq B$ and $C \preceq D$ hold in every consistent schedule, but precedence $B \preceq C$ and $D \preceq A$ do not exist because they only hold in the environments of $\{c_3 = 1\}$ and $\{c_3 = 2\}$, respectively. With the given precedences, the order of contingent links AB and CD is not decidable, so neither of them is guaranteed to be an observation of deciding discrete variables c_1, c_2 or c_3. Thus, the dynamically controllable decisions of time points and discrete variables cannot be assumed to happen after the observation of any of the contingent links, and therefore have to work for any value in their range. Therefore, we define the conditional precedence under a certain partial assignment PA:

Definition 10. In the environment of partial assignment PA, any pair of time points $v_i, v_j \in V$, v_i *precedes* v_j under PA, denoted as $v_i \preceq_{PA} v_j$, iff $S(v_i) \leq S(v_j)$, for every consistent schedule S such that its assignment $A \models PA$. For any link $e_i \in E$ and $v_j \in V$, e_i *precedes* v_j under PA, $e_i \preceq_{PA} v_j$, iff $\mathit{start}(e_i) \preceq_{PA} v_j$ and $\mathit{end}(e_i) \preceq_{PA} v_j$; v_j *precedes* e_i under PA, $v_j \preceq_{PA} e_i$, iff $v_j \preceq_{PA} \mathit{start}(e_i)$ and $v_j \preceq_{PA} \mathit{end}(e_i)$.

With Definition 10, the example in Figure 5 has conditional precedences $B \preceq_{c_3=1} C$ and $D \preceq_{c_3=2} A$ that enable different dynamic assignments after choosing c_3. Link AB can be the observation to decide c_2, when $c_3 = 1$; Link CD can be the observation to decide c_1, otherwise.

3.3 Assumptions

We introduce three assumptions before describing the rest of the problem formulation. Each of the assumptions will be discussed in more detail later on, where their role in the definition and algorithm become clear, but we state all three of them here for ease of reference.

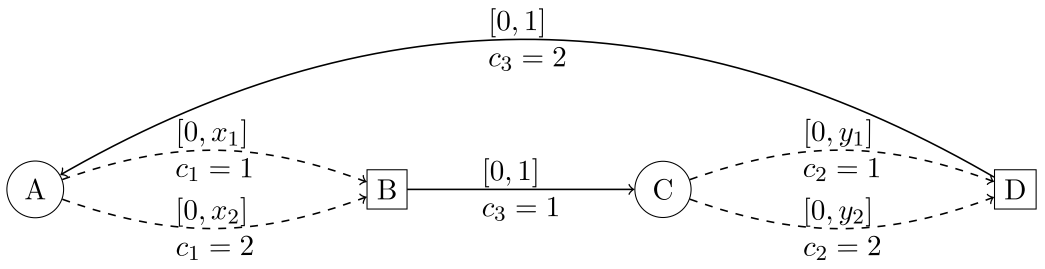


Figure 5: An Example Showing Precedences

Assumption 1. The assignment $A(c)$ is made once, at the time point $DT(c)$, which must precede any link $e \in E$ such that $\ell_E(e)$ mentions c. That is, $DT(c) \preceq e, \forall e$ s.t. $\ell(e) \cap D(c) \neq \emptyset$.

Assumption 2. Given a projection p under a certain partial assignment PA, the **prehistory** of a discrete variable c is the observed durations of contingent links which are activated by PA and must finish before or at $DT(c)$ in every execution, denoted $P_{PA}(p)\{\preceq c\} = \{p_{ij} | e_{ij} \preceq_{PA} DT(c) \text{ and } PA \models \ell(e_{ij}), p_{ij} \in p\}$.

Assumption 3. For each discrete variable c, the $DT(c)$ must be the end point of a contingent link or time point 0.

3.3.1 Limitation of Assumption 1

Assumption 1 associates discrete variables to time points and restricts their decision time to be no later than any link whose label includes an assignment of the variable.

With Assumption 1, Definition 7 means that during execution, before $DT(c)$, the strategy is the same for different options of c, after $DT(c)$, the strategy only needs to respect the sub-network without links deactivated by $A(c)$.

The first part of this assumption is implied by the definition of execution strategy, which is that the decision time point $DT(c)$ is associated with a single time point in V. The other part is that the associated decision time point should be chosen among nodes that precede every link mentioned by c. However, in principle a dynamic execution strategy could make 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.

A CCTPU modelling a situation like this is shown in Figure 6. It has four uncertain events $E_x \to E'_x$ and a discrete variable c. According to Assumption 1, the decision time for the discrete variable c has to be no later than $E4$; thus, it cannot be delayed until $E1'$ and so cannot be made depending on the observation of event $E1 \to E1'$. To some extent, this limitation can be overcome by remodelling the problem. Figure 7 shows an alternative formulation of the CCTPU in Figure 6, where the variable c with three options has been replaced by two binary choices, c_1 and c_2. This allows the choice between $c_1 = K$ and $c_1 = S$ to be postponed. (Sometimes a dummy node for the decision time point of the introduced discrete variables may be needed.) The remodelled example in Figure 7 has a

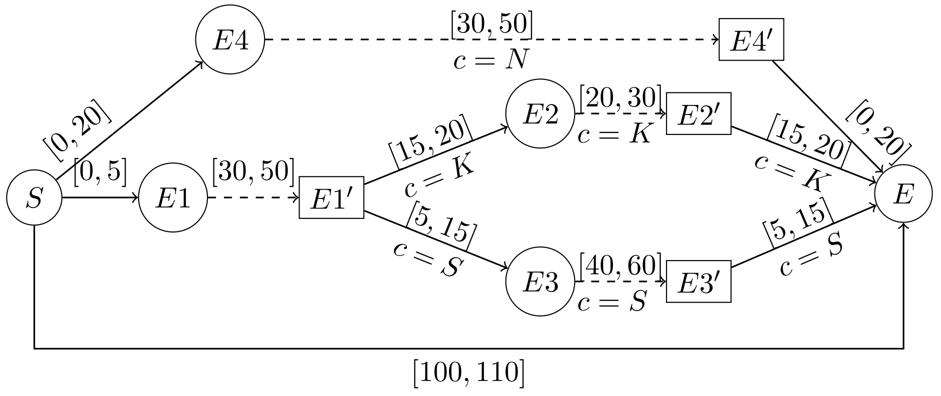


Figure 6: Discrete variable c has to be assigned no later than $E4$ according to Assumption 1.

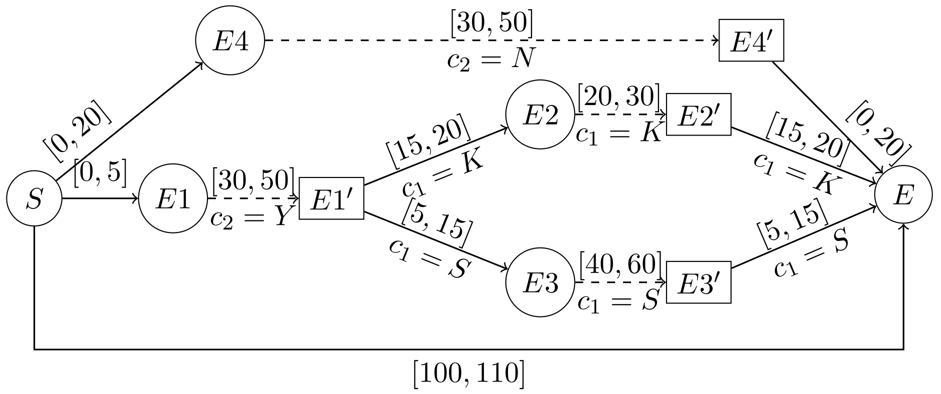


Figure 7: Remodelling the CCTPU in Figure 6, so that the assignment of c_1 can be made at $E1'$.

dynamic strategy with $DT(c_2) = S$ and $DT(c_1) = E1'$. At the beginning, $c_2 = Y$. At $E1'$, we can choose $c_1 = K$ when the duration of $E1 \rightarrow E1'$ is 40 or more, and $c_1 = S$ otherwise.

3.3.2 Limitation of Assumption 2

Assumption 2 restricts the observation on which the assignment of a discrete variable depends. Comparing to the dynamic controllability of an STPU, the *observed situation* (Vidal & Fargier, 1999), or *prehistory* (Morris et al., 2001; Hunsberger, 2009), at any time consists of the observed durations of contingent links that have finished before that time. Given schedule S of an 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 variables to only those contingent links that must finish before the variable's decision time point, so that the set of contingent links in the prehistory is stable under a given partial assignment.

Assumption 2 restricts the observation when making the choice of a discrete variable. The contingent links that are not guaranteed to complete before or at the decision time point are ruled out from the observation set.

Under a certain partial assignment, the **prehistory** of a set of unassigned discrete variables is the union of the prehistories of the variables in the set,

$$P_{PA}\{\prec C_s\} = \bigcup_{c \in C_s} P_{PA}\{\prec c\}.$$

Different from the observation of time point scheduling, contingent links that may end, but must not yet end, before the decision time point of a discrete variable are not observations for making the decision. In the DC reduction for an STPU, in a triangle ABC with contingent link AC, if the contingent link A–C may end but must not end before a time point B, it indicates that B is in the unordered case in the triangle reduction. Thus, B has a wait constraint with A–C. This means that whether A–C has finished or not can be used as an observation in the execution of the STPU.

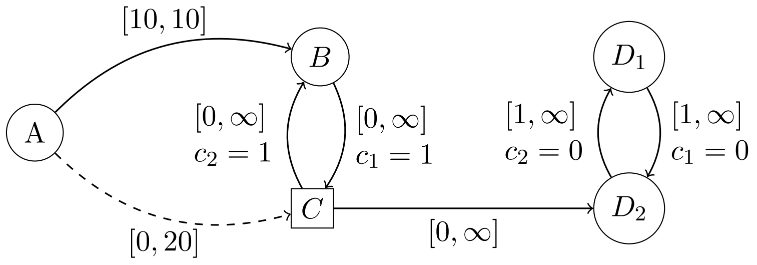


Figure 8: Example illustrating the limitation of Assumption 2.

An example is shown in Figure 8. The precedence constraints between D_1 and D_2 force at least one of the choices $c_1 = 1$ or $c_2 = 1$. At C, we can decide that $c_2 = 1$ iff the duration of A–C is less than 10, and $c_2 = 0$ otherwise. The choice for c_1 must be made no later than at time point B: at this point, we have to choose $c_1 = 1$ if and only if event C has not yet occurred. Under our Assumption 2, however, the A–C link is not part of the prehistory of B, and thus cannot be used to make the decision. Extending our methods to relax this assumption and consider on-going contingent links as observations is one of our avenues for future work (cf. Section 3.6).

Based on the prehistory which is the projections of contingent links that must finish before the decision time point, the dynamic assignment of a discrete variable is the choice made at its decision time point which must precede every link with labels where the assignment is occurring, and the decision activates and deactivates subnetworks accordingly.

3.3.3 Limitation of Assumption 3

Assumption 3 restricts the choice of decision time points to be either the start of the execution or the end points of contingent links. The remaining nodes cannot be decision time points, because different decisions are made according to different observations. However, the observable prehistory that can be used for making assignments, that is, the durations of contingent links that necessarily precede the time point, at those nodes is the same as at some node that is the end point of a contingent link. Figure 9 shows a schematic illustration: The dashed lines are contingent links, and the diamond node is the latest possible decision time for choice c. If the decision is made at r_2, the observation set only contains the prehistory up to and including t_1. Thus, potential decision time points are S, t_1, t_2 and the diamond node.

This assumption guarantees that the observable prehistory differs when making choice c, because it includes the duration of the link just finished. We call the observation of this contingent link the *key observation* for assigning c at this time point. Before deciding on c, the dynamically controllable execution strategy makes consistent decisions regardless of the following options, which means decisions before the key observation are the same for different options of c. However, after observing the key contingent link, different following decisions are made depending on the options of c.

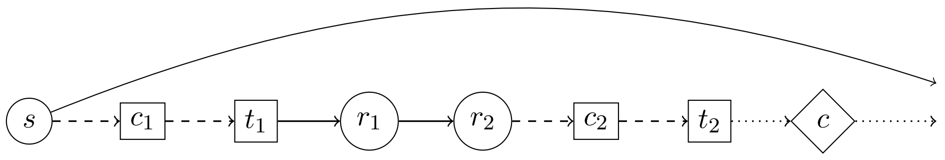


Figure 9: Alternatives for $DT(c)$. Squares are uncontrollable time points, circles controllable time points and the diamond is the latest decision time of c.

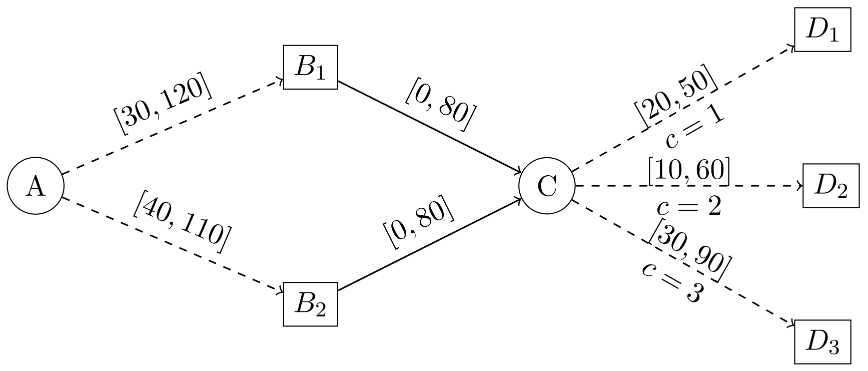


Figure 10: Example showing the limitation of Assumption 3: Time point C cannot be the decision time for c, although making the decision at this point would have both A–B_1 and A–B_2 in the prehistory.

However, Assumption 3 does rule out strategies that wait for parallel contingent links to finish before making a decision. An example is shown in Figure 10.

3.4 Dynamic Controllability of CCTPU

Dynamic controllability means that there is a viable execution strategy in which each decision only depends on observations before the decision. Based on the definition of dynamic assignment of discrete variables and the assumptions made in the preceding section, we now present a definition of dynamic controllability of CCTPU. This definition is sound but only complete with respect to our assumptions. It is possible to broaden the definition to more fully capture the intent of dynamic controllability, and we will discuss this briefly in Section 3.6 below. However, the dynamic controllability algorithm in Section 5 is complete only with respect to the definition we present here.

Definition 11. A CCTPU execution strategy $\langle DT, ES\rangle$ is **viable** iff

- $DT(c)$ precedes the start of every link $e \in E$ such that c appears in $\ell_E(e)$
- and $ES(p)$ is consistent for every projection p.

We now define dynamic controllability of a CCTPU as follows.

Definition 12. 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,
- $P_{PA}(p_1)\{\prec c\} = P_{PA}(p_2)\{\prec c\} \Rightarrow ES(p_1)(c) = ES(p_2)(c)$, for each discrete variable c and partial assignment PA.

In this definition, the decisions on time points are the same as dynamic controllability of STPU, the decisions on discrete variables only depend on the prehistories before the scheduling time of their decision time points, which also obey the origin of dynamic controllability.

3.5 Dynamically Controllable Envelopes

In this section, we define the **dynamically controllable envelope** of an STPU and a CCTPU under a partial assignment. The DC envelope is an important concept to explain our dynamic controllability checking algorithm later in this paper.

Dynamically controllable envelopes intend to define the conditions under which the following CCTPU has a viable dynamic execution strategy. In other words, the subset of uncertain prehistory with which the following CCTPU is dynamically controllable is the dynamically controllable envelope, which can be regarded as a relaxation of contingent links in the prehistory. For example, in the illustrative example shown in Figure 2, the subset $[50, 65]$ of the contingent link "Evacuate A" $\rightarrow$ "A reaches G" is the dynamically controllable envelope of making decision $c = G$ and $[55, 70]$ is the envelope of making decision $c = G'$.

The dynamically controllable envelope of a CCTPU can be aggregated from fully assigned branches which are STPUs. In the example, if we combine the two envelopes $[50, 65]$ and $[55, 70]$, their union covers the whole interval that uncertain event "Evacuate A" $\rightarrow$ "A reaches G" may take. Thus, making a dynamic choice between $c = G$ and $c = G'$ based on the observation of the uncertain event makes the whole strategy viable and dynamic.

The dynamically controllable envelope of a fully assigned CCTPU is the set of relaxations that can make the STPU dynamically controllable. A relaxation reduces the uncertainty by increasing lower bounds and/or decreasing upper bounds of contingent links. In the dynamically controllable envelope of a fully assigned CCTPU, the relaxable links are all contingent links not deactivated by its assignment. One relaxation EU' of a given set of contingent links EU can be formulated as for all $eu \in EU$, there exists $eu' \in EU'$, such that $eu' \subseteq eu$ and eu' and eu start from and end at a same pair of nodes.

$$EU' = relaxed(EU) = \{eu'_{ij} = [l'_{ij}, u'_{ij}] | eu \in EU, eu = [l_{ij}, u_{ij}], l_{ij} \leq l'_{ij} \leq u'_{ij} \leq u_{ij}\}. \quad (1)$$

Given a relaxation EU' and the CCTPU N, an updated network

$$N' = updated(N, EU') = N \setminus EU \cup EU'$$

replaces the set of contingent links with their relaxed counterparts.

Definition 13. The **dynamically controllable envelope** of a CCTPU with a full assignment A is $Env(A) = \{EU' | EU' = relaxed(EU), N' = updated(N, EU'), N'$ is a dynamically controllable STPU under assignment $A.\}$, where EU' is a relaxation of EU.

Envelopes with a common prehistory can be aggregated. For instance, two envelopes with full assignments $Env(A_1)$ and $Env(A_2)$, where $A_1 = PA \cup \{c = d_1\}$ and $A_2 = PA \cup \{c = d_2\}$ have the common partial assignment PA, and aggregate to $Env(PA, c)$ on prehistory $P_{PA}\{\prec c\}$. After aggregating all branches from assigning c to $Env(PA, c)$, the combined envelope $Env(PA, c)$ describes the previous condition, in which one of the options of c can make a dynamically controllable strategy.

Definition 14. The **dynamically controllable envelope** of assigning a discrete variable c in a CCTPU after a certain partial assignment PA is $Env(PA, c) = \{EU' | EU' = relaxed(EU, PA, c), N' = updated(N, EU')$ is dynamically controllable.$\}$ where $EU' = relaxed(EU, PA, c)$ is the set of relaxed bounds of contingent links belonging to $P_{PA}\{\prec PA \cup c\}$ and make the updated CCTPU dynamically controllable.

If given $c' \preceq c$, after aggregating envelopes from all branches $Env(PA \ \cup \{c = d_i\})$ to $Env(PA, c)$, we want to aggregate $Env(PA \setminus \{c' = d'\})$, where $\{c' = d'\} \subset PA$. The combined envelope $Env(PA, c)$ cannot be used directly in the next level aggregation that combining branches for discrete variable $c' \in PA$ because it may contain links belong to $P_{PA}\{\prec c\}$ not $P_{PA \setminus c'}\{\prec c'\}$. Thus, we present Definition 15.

Definition 15. The **dynamically controllable envelope** of a CCTPU under partial assignment PA is $Env(PA) = \{EU' | EU' = relaxed(EU, PA) \wedge N' = updated(N, EU')$ is dynamically controllable$\}$, where $relaxed(EU, PA)$ is the set of relaxed bounds of contingent links where relaxable links belong to $P_{PA}\{\prec PA\}$ and make the updated CCTPU dynamically controllable.

Therefore, the aggregating process to compute the envelope of the CCTPU with partial assignment PA before assigning the next discrete variable c is the set of unions of selected subsets of envelopes of branches $PA'_d = PA \cup \{c = d\}$, which is

$$Env(PA, c) = \{ \bigcup_{d \in D(c)} EU'_d | EU'_d \in Env(PA'_d)\}. \quad (2)$$

Next, envelope $Env(PA)$ selects a subset of $Env(PA, c)$ containing only the relaxations such $eu'_x = eu_x$ for all $eu_x \notin P_{PA}\{\prec PA\}$, i.e., only the relaxations whose relaxed links are in the prehistory of PA, since only those can be observed when making the choice for c.

$$Env(PA) = \{EU' \in Env(PA, c) \mid eu_x \notin P_{PA}\{\prec PA\} \rightarrow eu'_x = eu_x\} \tag{3}$$

The envelope of a CCTPU can be collected by implementing the aggregation in Equation 2 repeatedly until $PA = \emptyset$,

A CCTPU is dynamically controllable if and only if its dynamically controllable envelope with an empty partial assignment is the set of the original bounds,

$$Env(\emptyset) = EU.$$

Additionally, the dynamically controllable envelopes of a CCTPU satisfy the following relation:

$$Env(A) = EU \Rightarrow Env(PA) = EU \Rightarrow Env(\emptyset) = EU. \tag{4}$$

This means a CCTPU is dynamically controllable if it has a dynamically controllable envelope with a partial assignment, which covers the uncertainty in the prehistory of the partial assignment, and if a CCTPU has a dynamically controllable STPU branch, it is dynamically controllable.

3.6 Relaxing the Assumptions

Assumptions 2 and 3 are essentially limitations on the dynamic execution strategies that can be found by the algorithm we present in the remainder of the paper. Removing them from the *definition* of dynamic controllability is straightforward. Assumption 3 does not influence Definition 12 or the definitions it depends on at all. Relaxing Assumption 2 would only change the definition of the prehistory of a discrete choice, making it $S\{\prec DT(c)\}$, analogously with the prehistory of any time point. Where these assumptions play a significant part is in the definition of the DC envelope of a partially assigned CCTPU and the envelope aggregation process (Section 3.5). As will be shown in Section 5, this aggregation process is central to our DC checking algorithm for the CCTPU.

Relaxing Assumption 1 is not so easy. If we view the assignment of a discrete variable as a choice, e.g., a choice between taking different actions, then a dynamic execution strategy must ensure that 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. The assumption that a discrete choice between more than two alternatives is made at one time is somewhat restrictive, as shown by the example in Figure 6, and could be relaxed as long as this requirement is met. If, on the other hand, we view the assignment of a discrete variable simply as a disjunction between different sets of constraints, i.e., that at least one option must eventually be satisfied, then the choice of which one is could be postponed. We will return to this distinction in Section 8.1 when discussing related temporal reasoning models.

4. Extracting Dynamically Controllable Envelopes of STPU

Given an STPU, the algorithm in this section returns its dynamically controllable envelope. The dynamically controllable envelope of an STPU is the observable condition under which the STPU is dynamically controllable. Finding the dynamically controllable envelope of an STPU is to find the subset of situations in which no conflict of dynamic controllability occurs. Our approach starts from a current dynamic controllability checking algorithm, but modifies it to return all conflicts. The envelope is then formulated as a set of constraints that must be satisfies to resolve all the found conflicts.

4.1 Checking Dynamic Controllability for the STPU

Different algorithms that can verify if a given STPU is dynamically controllable have been well studied. The approaches in this paper are mainly based on Morris's algorithms (2006, 2014) which will be discussed in this subsection. Besides Morris's cubic algorithm, Nilsson, Kvarnström, and Doherty (2014) introduced an $O(N^3)$ incremental algorithm based on Shah, Stedl, Williams, and Robertson (2007)'s incremental algorithm, and Hunsberger (2015) introduced *Inky* which is also an incremental algorithm with time complexity $O(N^3)$ which are the most efficient approaches.

The first dynamic controllability checking algorithm was introduced by Vidal and Fargier (1999) with the original definition of dynamic controllability of STPU. The approach is a two-player game – one decision agent plays against the other nature agent. An STPU is dynamically controllable if and only if the decision agent can always make successive decisions based the past no matter what the nature agent has done.

The first polynomial algorithm for checking dynamic controllability of the STPU was introduced by Morris et al. (2001), based on *boundary projection* and *safe networks* (Morris & Muscettola, 2000). This algorithm repeatedly tightens requirement links and introduces wait constraints according to the triangular reduction and wait reduction (which are discussed later) and checks if the tightened network is pseudo-controllable until no link can be tightened or the network is not pseudo-controllable. Nilsson, Kvarnström, and Doherty (2015) have proved that this algorithm is in essence $O(N^4)$.

Morris and Muscettola (2005) provided an $O(N^5)$ algorithm by representing the STPU by a *labelled distance graph* and obtaining a cutoff bounds analogous to Bellman-Ford structure instead of exhausting the tightenings. In the *labelled distance graph*, the requirement links are formulated in the same way as in the distance graph introduced by Dechter et al. (1991). Each link $A \xrightarrow{[l,u]} B$ is represented by two directed links $A \xrightarrow{u} B$ and $B \xrightarrow{-l} A$. The contingent links are represented with labelled edges. Each link $A \xrightarrow{[l,u]} B$ is presented by $A \xrightarrow{b:l} B$ and $B \xrightarrow{B:-u} A$, which are called the lower-case and upper-case edges respectively. Thus, the reduction rules of dynamic controllability can be more uniform and those rules are still applicable in the following and more efficient algorithms.

In 2006, Morris introduced an $O(N^4)$ algorithm. The author first rephrased the reduction rules by transforming the labelled edges for contingent link $A \xrightarrow{[l,u]} B$ to $A \underset{-l}{\overset{l}{\rightleftharpoons}} A' \underset{B:l-u}{\overset{b:0}{\rightleftharpoons}} B$. Thus, the label removal condition $x \geq l^c$ can be changed to $x \geq 0$. The reduction rules are shown in Equation 5. The conditions in the reduction rules are about checking if the weights are negative or not, which helps to introduce a faster propagation algorithm.

$$
\begin{aligned}
&\text{(Upper-case Reduction)} && A \xleftarrow{B:x} C \xleftarrow{y} D \text{ adds } A \xleftarrow{B:(x+y)} D. \\
&\text{(Lower-case Reduction)} && \text{If } x < 0,\ A \xleftarrow{x} C \xleftarrow{c:y} D \text{ adds } A \xleftarrow{x+y} D. \\
&\text{(Cross-case Reduction)} && \text{If } x < 0,\ B \neq C,\ A \xleftarrow{B:x} C \xleftarrow{c:y} D \text{ adds } A \xleftarrow{B:(x+y)} D. \qquad (5) \\
&\text{(No-case Reduction)} && A \xleftarrow{x} C \xleftarrow{y} D \text{ adds } A \xleftarrow{x+y} D. \\
&\text{(Label Removal)} && \text{If } x \geq 0,\ A \xleftarrow{B:x} C \text{ adds } A \xleftarrow{x} C.
\end{aligned}
$$

Using these reduction rules, a sequence of labelled edges can be transformed into a new edge. Furthermore, if a path of edges after the reductions have been applied does not have any lower-case edges, it is called *semi-reducible*.

Theorem 4.1. *(Morris, 2006) An STPU is Dynamically Controllable if and only if it does not have a semi-reducible negative cycle.*

In this paper, we call these semi-reducible negative cycles *dynamic controllability conflicts* or just *conflicts*. With Theorem 4.1, a faster dynamic controllability checking algorithm was introduced, which propagates lower-case edges with Dijkstra's algorithm and checks for semi-reducible negative cycles with the Bellman-Ford algorithm iteratively. The number of iterations is no more than the number of contingent links, so the algorithm is $O(N^4)$.

The current fastest algorithms are $O(N^3)$ (Morris, 2014; Hunsberger, 2015; Nilsson et al., 2014). Morris's method propagates negative links backwards in order to find potential *moat edge*, which is the first edge e' following a lower-case edge e in path P and its reduced distance on path P $D_P(end(e), end(e'))$ is negative. The key process in this algorithm is called *DCbackprop* that propagates the end of a negative link backwards in Dijkstra's algorithm while applying the reduction rules of dynamic controllability. When meeting the end of another negative link, it calls *DCbackprop* again. Otherwise, it sums up the weights of paths until they are not negative. The non-negative paths are reduced as new edges without a lower-case label.

Theorem 4.2. *(Morris, 2014) The DCbackprop procedure encounters a recursive repetition if and only if the STPU is not Dynamically Controllable.*

Theorem 4.2 was proved by Morris (2014). He showed that the DCbackprop procedure encounters a recursive repetition if and only if the STPU has a semi-reducible negative cycle. Therefore, a dynamically controllable STPU can pass the checking algorithm without early termination caused by a conflict.

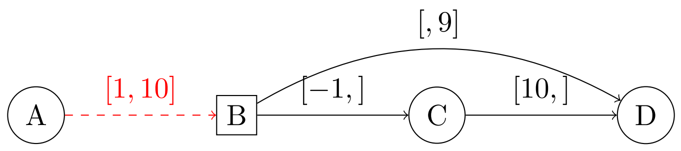


Figure 11: An STPU contains a conflict. Because the temporal constraints on BD and CD infer that $u_{BC} \leq -1$, which means C has to be scheduled before the observation of B. Thus, triangle ACB is in the precede case, and the upper bound of AC $u_{AC} \leq l_{AB} - u_{CB} = 0$. However, if C is scheduled no later than A and the uncertain duration of AB is greater than 1, the requirement link on BC cannot be satisfied.

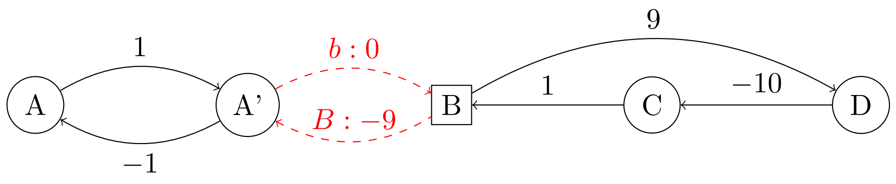


Figure 12: The labelled distance graph of Figure 11.

4.2 Conflict Resolutions of STPU

Finding the dynamically controllable envelope of an STPU is to find the subset of prehistories in which all conflicts do not exist. This problem is similar to relaxing an over-constrained (non-DC) problem. Yu et al. (2014) formulated the relaxation problem as a linear programming model with a set of constraints derived from conflicts, and proposed an algorithm, called CDRU, to iteratively generate conflicts. The conflicts represent the reasons why the STPU is not dynamically controllable. A conflict is a semi-reducible negative cycle in the network after applying dynamic controllability reduction rules (see Section 4.1). A conflict can be represented as follows:

$$\sum_{i \in conf_j} x_i < 0, \tag{6}$$

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

$$\sum_{i' \in conf_j \cap ER} x'_{i'} + \sum_{i \in conf_j \setminus ER} x_i \geq 0 \tag{7}$$

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 (Equation 5) has a label condition:

$$\text{(Cross-case Reduction)} \quad \text{If } x < 0,\ B \neq C,\ A \xleftarrow{B:x} C \xleftarrow{c:y} D \text{ adds } A \xleftarrow{B:(x+y)} D.$$

For example, the STPU in Figure 11 has a conflict that can be found by any dynamic controllability checking algorithms. The reduction process of Morris's $O(N^3)$ algorithm represented by the labelled distance graph is shown in Figure 12. Equation 8 illustrates the reduction process. The pair of parentheses shows an added link $C \overset{0}{\leftarrow} A$ reduced from the labelled distance graph.

$$A \xleftarrow{-1} A' \xleftarrow{B:-9} B \overset{1}{\leftarrow} (C \xleftarrow{-10} D \overset{9}{\leftarrow} B \xleftarrow{b:0} A' \overset{1}{\leftarrow} A) \tag{8}$$

A resolution of the found conflict is

$$-u'_{AB} - l'_{BC} - l'_{CD} + u'_{BD} + l'_{AB} \geq 0.$$

In order to resolve the conflict, the total amount of relaxation is 9.

However, this conflict can also be resolved by only relaxing $C \leftarrow D$ by 1. It will make the back propagation of $C \xleftarrow{-9} D$ stop at B and add a new link $C \overset{0}{\leftarrow} B$. Then the back propagation of $A \xleftarrow{-10} B$ is

$$A \xleftarrow{-1} A'(\xleftarrow{B:-9} B \overset{1}{\leftarrow} C \overset{0}{\leftarrow} B \xleftarrow{b:0} A' \overset{1}{\leftarrow} A)$$

which has to stop before $B \xleftarrow{b:1} A$ because the propagated path $A \xleftarrow{B:-9+1} B$ has a label of B which does not satisfy the condition of cross-case reduction. The complete conflict resolution of the conflict in Equation (8) is

$$-u'_{AB} - l'_{BC} - l'_{CD} + u'_{BD} + l'_{AB} \geq 0 \vee l'_{CD} + u'_{BD} \geq 0.$$

Therefore, the resolution of a single conflict is a disjunction with one linear constraint of the form (7) and other linear constraints over the links whose relaxations can break the reductions:

$$\bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} + \sum_{i \in res_{jk} \setminus ER} x_i \geq 0 \tag{9}$$

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

Yu et al. (2014) solved an LP over constraints of form (7) to find a single relaxation. Bhargava, Vaquero, and Williams (2017) enhanced the conflict exploration process by using an incremental method. However, provided we find *all* conflicts in the STPU, the solution of conflict resolution constraints of Equation 9 represents the space of all relaxations, which is the same as the dynamically controllable envelope.

4.3 Extracting Conflicts from a STPU

In this subsection, we describe the method to extract a complete conflict set of a given STPU. Since the conflicts are negative cycles, the number of conflicts is infinite when there is one negative cycle, and the edges can reduce along the negative cycle infinitely. However, our algorithm only finds conflicts that cannot be divided into other found conflicts, which makes the found conflicts a complete conflict set.

Our algorithm is a variation of the dynamic controllability checking algorithm by Morris (2014). Morris's cubic algorithm identifies a non-dynamically controllable STPU by finding a semi-reducible negative cycle during propagation according to dynamic controllability reduction rules. It propagates every negative link backwards along non-negative links only, adds a new link when the weight of the propagated path is non-negative and raises another propagation when meeting another negative link. A conflict is found when the raised propagation is a negative link that has an unfinished ancestor propagation, and the algorithm terminates. It guarantees that every negative link is propagated once at most.

Our approach expands Morris's $O(N^3)$ algorithm by propagating through non-shortest paths and recording a complete set of conflicts. The conflict resulting in the termination can be learned by memorising the path when tracing back.

However, iteratively extracting and resolving one conflict learned from the dynamic controllability checking algorithm does not lead to such a complete conflict set, since resolving one conflict may prevent us from finding other conflicts. For example in Figure 13, where solid lines are edges and dotted lines are paths consisting positive links to be propagated. Both paths p_1 and p_2 formulate negative cycles $A \leftarrow B \leftarrow A$, but only the shorter one will be found when calling backpropagation from $A \xleftarrow{L_1:-x} B$. Furthermore, if the conflict resolution relaxes $A \leftarrow B$, it may resolve the other conflict in the instance, in which case the the other conflict may never be found.

Another example in Figure 14 shows a case with cyclic calls of backpropagations. If it first calls backpropagation from $A \xleftarrow{L_1:-x} B$, the algorithm will terminate when finding negative cycle I. Furthermore, if the conflict resolution relaxes the overlapped part of the two negative cycles, it may prevent from finding negative cycle II.

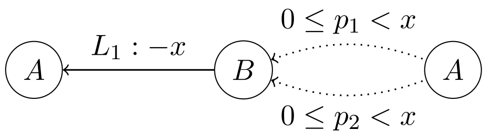


Figure 13: Alternative conflicts I

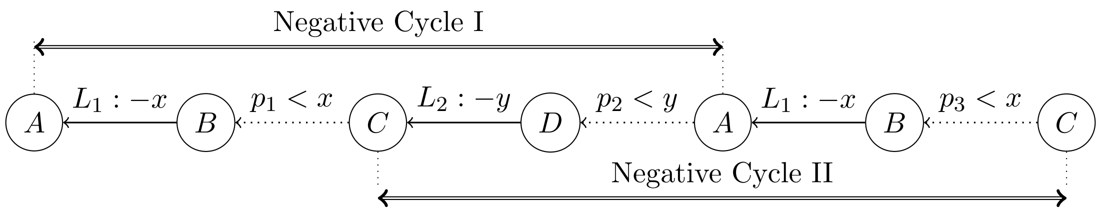


Figure 14: Alternative Conflicts II

Based on these observations, to find all necessary conflict resolutions, we have to modify Morris's algorithm in the following ways: (1) replacing the shortest path process by a method that can propagate through all possible paths and (2) propagating some negative links more than once. If the backpropagation of a negative link is involved in a conflict, but the conflict is learned at cyclic propagations ending at another negative link, this propagation may be *incomplete*. For instance in Figure 14, if negative cycle I is found by the back propagation starting from $A \xleftarrow{L_1:-x} B$, the raised propagation of $C \xleftarrow{L_2:-y} D$ is not complete which needs to be propagated again. If we start a new round of propagation of $C \xleftarrow{L_2:-y} D$, negative cycle II will be found and the propagation is complete. The propagations without ancestor propagations and involving no conflict are complete.

The approach to extract a complete set of conflicts is summarised in Algorithms 1, 2 and 3. Algorithm 3 and Lines 7 – 11 of Algorithm 2 describe the Depth First Search (DFS) process, while the rest is the same as Morris's method.

Algorithm 1 calls $BackPropagation$ once on every negative edge of the labelled distance graph of a given STPU. $NegPathEnds$ records negative paths when tracing back to formulate $NegCycles$.

$BackPropagation$ (Algorithm 2) terminates if the current node recursively reached itself through a negative path (lines 1 – 3) or if propagation from the current node has already been successfully completed (lines 4 – 6). $disEdge[i]$ is the reduced distance edge from i to the end of $srcLink$ in the current round of propagation.

In DFS (Algorithm 3), we enumerate edges ending at the current node $crtNode$ (Line 2 – 29). Lines 3 – 5 prevent cross-case reductions that do not satisfy the label condition. As proven by Morris's algorithm, the algorithm only propagates through non-negative links ending in $crtNode$ (Line 6 – 15). The function $Reduction(X, Y)$ (Line 7) adds two links X and Y based on the dynamic controllability reduction rules (Equation 5) and keeps a record of the reduction history. If the propagated path is positive, a new edge is added (Line 9), else, propagation continues to call DFS (Line 13). When encountering a negative link, the algorithm will call $BackPropagation$ again (Line 17). $bReturn$ marks the existence of a recursively called negative link. If there exists a recursive call, the current path from $crtNode$ to $srcNode$ can be part of the found conflict, so we add the path $disEdge[crtNode]$ to $NegPathEnds[srcNode]$ in line 19. Furthermore, $NegPathEnds[crtNode]$ keeps a record of negative paths (as $NodeX \xrightarrow{e1} crtNode$) ending at the current node, which can propagate through $crtNode \xrightarrow{disEdge[crtNode]} srcNode$ and formulate a new negative path as $NodeX \xrightarrow{NewPath} srcNode$ ending at the source node, where $NewPath$ propagates $e1$ through path $disEdge[crtNode]$. Line 23 adds the negative cycle if it can be found in the propagation of $NewPath$; Line 25 adds the negative path, otherwise.

```
Algorithm: DCEnvelopeSTPU($N$)
Input: An STPU $N$.
Output: A set of conflicts $NegCycles$
1 global $LabelN = labelledDistanceGraph(N)$
2 $NegCycles = \emptyset$
3 for $e$ in negative edges of $LabelN$ do
4     $NegPathEnds = \{\emptyset\}$
5     BackPropagation($e$, $NegPathEnds$, $NegCycles$)
6 end
7 return $NegCycles$
```

Algorithm 1: Extracting the DC envelope of an STPU.

The complexity of our conflict extraction method is $O(E^3)$ in the worst case, when all links are negative and the network is fully connected, where E is the number of links. It is slower than the state-of-art DC checking methods and the CDRU relaxation algorithm.

```
Algorithm: BackPropagation($srcLink, NegPathEnds, NegCycles$)
Input: A negative link $srcLink$, the vector of sets of negative paths end at same nodes
       $NegPathEnds$, 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  endif
4  if prior successfully terminated call with srcLink then
5  |   return True
6  endif
7  $disEdge = \{\}$
8  $disEdge[srcLink.start] = srcLink$
9  if $DFS(srcLink.start, srcLink.end, NegPathEnds, NegCycles)$ then
10 |   return True
11 endif
12 return False
```

Algorithm 2: Modified backpropagation process that can extract all conflicts.

However, it returns the complete set of conflict resolution constraints, not only whether conflicts exist, or one relaxation.

4.4 Dynamically Controllable Envelopes of STPU

After finding the complete set of conflicts, the dynamically controllable envelope of an STPU is the solution space of a constraint model, which is a conjunction of constraints that represent conflict resolutions.

Given the set of relaxable edges ER that represents the uncertain situations which can be partially considered in the STPU, the dynamically controllable envelope of the STPU is $x'_{i'}$ satisfying the following constraints

$$\bigwedge_{j \in Conf} \bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} + \sum_{i \in res_{jk} \setminus ER} x_i \geq 0. \tag{10}$$

The dynamically controllable envelope of an STPU can be used to aggregate the dynamically controllable envelopes of the CCTPU.

5. Dynamic Controllability Checking of CCTPU

Central to our algorithm for finding a dynamic execution strategy is the notion of the envelope of a partial assignment of the discrete variables. In Section 3.5, we defined the envelope of a partially assigned CCTPU in terms of the envelopes of the same CCTPU with one further assignment made. The key property of the DC envelope can be stated as follows: 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

```
Algorithm: DFS($crtNode, srcNode, NegPathEnds, NegCycles$)
Input: Current node $crtNode$, the source node of back-propagation $srcNode$, the vector of sets
       of negative paths end at same nodes $NegPathEnds$ and the set of conflicts $NegCycles$.
Output: Return DFS result and the updated $NegCycles$ and $NegPathEnds$
$bReturn = True$
for $e$ ends with $crtNode$ do
    if $e$ is unusable then
        continue
    endif
    if $e.weight \geq 0$ then
        $NewE = Reduction(disEdge[crtNode], e)$;                    // Equation (5)
        if $NewE.weight \geq 0$ then
            $LabelN$.addEdge($NewE$)
        else
            $TmpDis = disEdge[e.start]$
            $disEdge[e.start] = NewE$
            $bReturn\& = DFS(e.start, srcNode, NegPathEnds, NegCycles)$
            $disEdge[e.start] = TmpDis$
        endif
    else
        if !$BackPropagation(e, NegPathEnds, NegCycles)$ then
            $bReturn = False$
            $NegPathEnds[srcNode].add(disEdge[crtNode])$
            for $e1$ in $NegPathEnds[crtNode]$ do
                $NewPath = Reduction(disEdge[crtNode], e1)$
                if $NewPath$ contains conflicts then
                    $NegCycle.add(NewPath)$
                else
                    $NegPathEnds[srcNode].add(NewPath)$
                endif
            end
        endif
    endif
end
return $bReturn$
```

Algorithm 3: The DFS process that do reductions through positive links.

to the notion of a relaxation of an over-constrained CCTPU, which also allows tightening contingent links, but captures all ways of making the subproblem dynamically controllable. A detailed example of a DC envelope, for the problem in Figure 18, is given by Equation (15) at the end of Section 5.4.

The DC envelope of a set of discrete variables is a combination of the variables' envelopes. For a 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. It 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 in the tree.

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

5.1 Algorithm Structure

Our dynamic controllability checking algorithm for a CCTPU (Algorithm 4) 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 – 8); 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 (line 3). The detailed description of *DCEnvelopeSTPU* is in section 4. A non-leaf node (lines 9 –18) is expanded by assigning values to the next variable and exploring those nodes recursively. After the child branches of a node have been investigated, their DC envelopes are combined and recorded as the DC envelope of the current node (line 14). If the envelope of a node is dynamically controllable, then so is the CCTPU and the algorithm returns successfully. In the worst case, the algorithm is exponential since it may have to explore the whole tree to verify that a CCTPU has no fully dynamic strategy. However, if a dynamic execution strategy exists, it may be found without enumerating all assignments of the discrete variables.

The following subsections explain the three subroutines besides *DCEnvelopeSTPU*: *NextUnassignedVariable*, which determines the order to assign discrete variables, *Union* (line 14), which combines envelopes from branches of the same node, and *isDC* (line 15), which checks if the current solution includes a dynamic strategy for the original problem. We explain these procedures in the following subsections.

5.2 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 this order.

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 18 the choice between $c_2 = 1$ and $c_2 = 2$ depends on the choice for c_1. Dependencies among discrete variables form a directed graph.

To find the chronological order, we build dependencies among variables. Recall our definition of precedences (Definitions 9 and 10), we give the following definition.

```
Algorithm: TreeSearch($Node$, $A$, $S$)
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
$c \leftarrow$ NextUnassignedVariable($Node$)
if $isNull(c)$ then
    $Node.S \leftarrow$DCEnvelopeSTPU($Node$)
    if $isDC(Node.S)$ then
        return TRUE
    endif
    return FALSE
endif
for $dc_i$ in $D(c)$ do
    $NewNode \leftarrow$ Assign($Node, c, dc_i$)
    if $TreeSearch(NewNode)$ then
        return TRUE
    endif
    $Node.S \leftarrow Union(Node.S, NewNode.S)$
    if $isDC(Node.S)$ then
        return TRUE
    endif
end
return FALSE
```

Algorithm 4: Checking dynamic controllability of a CCTPU.

Definition 16. If a link $e \preceq DT(c_i)$ and e can be activated or deactivated by c_j, then c_i **depends on** c_j, denoted as $c_i \prec c_j$. If a link $e \preceq_{PA} DT(c_i)$ and e can be deactivated (the same as activated) by $A(c_j)$, then c_i **depends on** c_j under partial assignment PA, denoted as $c_i \prec_{PA} c_j$.

With these definitions, we can formulate a dependency graph from which we can extract a chronological order in which to assign variables. If the dependency graph does not have a cycle, we can use topological sort to find any possible chronological order. Otherwise, the dependency cycles can be treated as conflicts. If the network is dynamically controllable, at least one of the links within a cycle need to be deactivated by assignments. Therefore, we can just try different possible orders, if a feasible dynamic strategy is available under one such order, the network is still dynamically controllable.

Figures 5 and 15 (the diamond nodes are decision time points) show examples of cyclic dependencies, which will cause a failure to select the next unassigned variable. The cyclic dependencies containing links with different assignments from the same variable, like in Figure 5, are ruled out by our Assumption 1 because the decision time $DT(c_3)$ has to be placed before node A, B, C and D which breaks the dependencies $c_1 \prec c_3$ and $c_2 \prec c_3$. Therefore, assigning c_3 before entering the cycle will break the cyclic dependency. However, cyclic dependencies that contain links with labels of different variables, as in Figure 15, cannot be solved in the same way. To deal with this case, we can remodel the problem by adding a discrete variable c_0, attaching assignments to each link included in the cyclic dependency in Figure 16, so that the cyclic dependency will be broken by any assignment of c_0 which is made before assigning c_1, c_2 and c_3.

Because all cyclic dependencies either contain links with labels of repeated variables or different variables, which can be broken by either assumption 1 or remodelling, we can always find the next unassigned variable that does not depend on any other unassigned variables. The order to assign variables is the chronological order we use to branch the search tree.

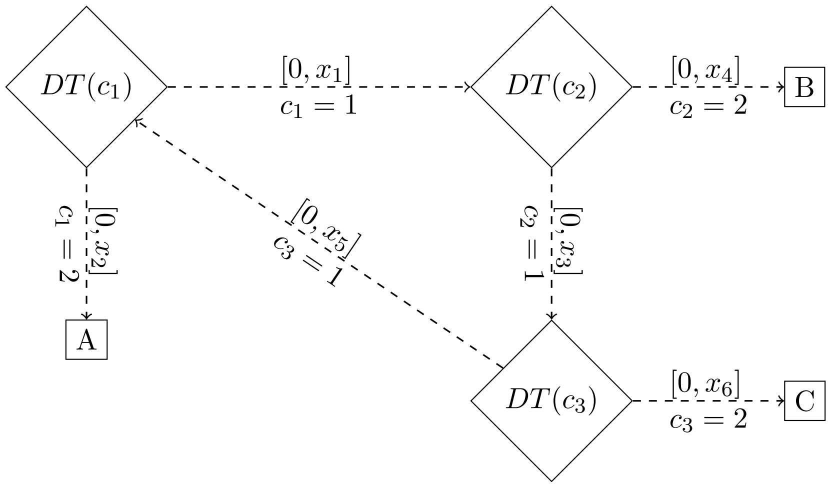


Figure 15: Cyclic Dependencies among Variables II

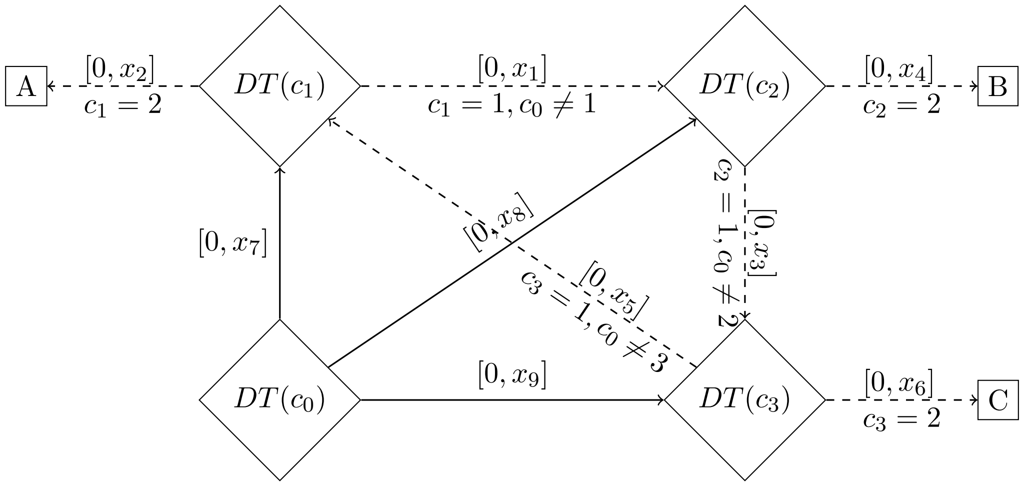


Figure 16: Remodelling Cyclic Dependencies among Variables II

5.3 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.

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 a sequence of contingent inks) as the latest observation in each combining process.

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, and each combining process aggregates child envelopes according to Equation (2). The dynamically controllable envelope as defined in Formula 2 helps us understand the combining process, but the elements of the dynamically controllable envelope are uncountable since the domain of bounds is continuous. Therefore, we use the constraints of conflict resolutions to represent dynamically controllable envelopes, whose solution space is the set of relaxations according to Definition 15.

The envelope of the node to assign discrete variable c under partial assignment PA is a disjunction of child nodes' envelopes:

$$\Phi = \bigvee_{dc_l \in D(c)} \bigwedge_{j \in Conf} \bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl}, \tag{11}$$

where $ER = \{e_i | e_i \in EU \text{ such that } e_i \prec_{PA} DT(c)\}$ and $a_{jkl} = -\sum_{i \in res_{jk} \setminus ER} x_i$ is the sum of bounds of links after $DT(c)$. As the selection in Equation (3) may fail because no suitable element can be found, the child envelopes may be infeasible when reducing relaxable links. Those envelopes will be removed before being combined. Furthermore, if no child envelope is available to be combined, the current envelope does not contain dynamically controllable execution strategy, which does not need to be combined in upper levels.

The envelope can be expanded into a conjunction of disjunctions, as shown in the following equation.

$$\Phi = \bigwedge \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl} \tag{12}$$

Each conjunct takes one conflict resolution (which is a disjunction of linear constraints) from every child node branch in equation (11), 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 (10), makes the combining process uniform at all levels of the tree. Φ can cover the whole uncertainty if and only its negation is infeasible within the original bounds of relaxable contingent links.

$$\begin{aligned}\neg\Phi &= \neg \bigwedge \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl} \\ &= \bigvee (\neg \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl}) \\ &= \bigvee \bigwedge \sum_{i' \in res_{jk} \cap ER} x'_{i'} < a_{jkl} \end{aligned} \tag{13}$$

As each disjunct in equation (13) contains a conjunction of linear constraints, we can use an LP solver to perform its test. A disjunct can be removed, when it is infeasible, which means its negation covers the whole uncertainty. We only keep the remaining disjunctions for the following combining processes. Additionally, if every disjunct is infeasible, Φ covers every uncertain situation in the prehistory, which means the current node is dynamically controllable and an execution strategy can be found within the combined envelope.

5.3.1 Decision Consistency in Prehistory

The decision consistency aims to explain why and how we consider one contingent link in each combining process.

During execution, the strategy splits at a decision time point $DT(c)$ by assigning variable c. For example in the illustrative Figure 2, after deciding which way B will choose, the constraints representing the other option do not matter any more. However, even though we keep all contingent links preceding the decision time point as relaxable links in the dynamically controllable envelopes, only one of them can be chosen to have its uncertainty split into branches each time. In Figure 9, there may exist more than one contingent link before the latest decision time point c. If one child branch tackles both $s \rightarrow t_1$ and $r_2 \rightarrow t_2$ partially, the temporal scheduling of r_1 and r_2 separates the two observations. The strategy is related to dynamic controllability reduction rules. The temporal scheduling of r_1 and r_2 only deals with the relaxed $s \rightarrow t_1$.

When combining envelopes from branching decisions of a discrete variable, the envelopes must contain relaxable variables on no more than one contingent link. If the envelope contains linear constraints on variables of both contingent links in Figure 9, deciding requirement links between two partial observations means the decision has to be made before the observation of the second contingent link, which is not dynamically controllable. Otherwise, the envelope contains separate constraints on both contingent links. Although several such envelopes can combine and cover the whole prehistory, the decision on the discrete variable has been partially made in the decision of the requirement link in this case, which breaks our assumption 1.

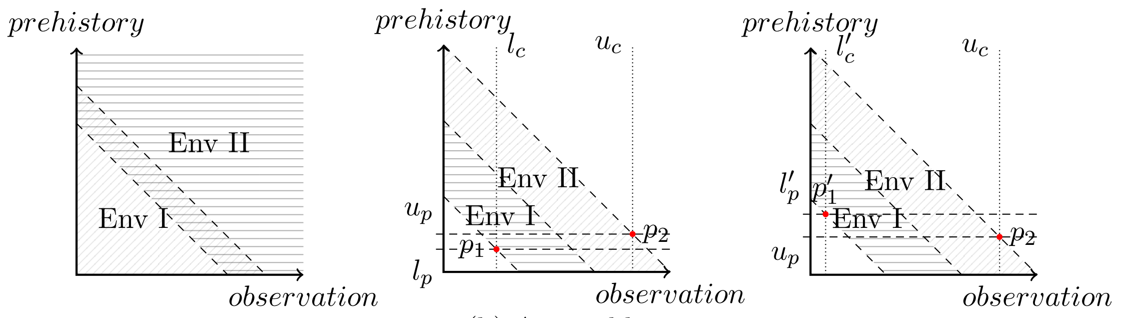
In each combining process, we consider one contingent link as the critical observation that can be partially tackled in child branches by making a choice for one discrete variable. We try the end node of every contingent link that precedes the latest decision time of c as $DT(c)$, replacing the variables 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 $c = dc_i$.

After selecting the key observation, we check the feasibility of each disjunct in equation (13) with the following linear programming problem.

$$s.t. \bigwedge \sum_{i' \in res_{jk} \cap ER \setminus \{x_{ij}\}} p_{i'} + x_{ij} < a_{jkl} \tag{14}$$
$$l_{ij}^c \leq x_{ij} \leq u_{ij}^c$$

Besides the change of x_{ij}, we also temporally use $p_{i'}$ to represent the other edges in the conflict resolutions. It is because both lower and upper bounds of contingent links in the envelope represent the uncertain projection in prehistory, which is observed at the time the choice will be made.

Figure 17 illustrates situations of the reduction of a disjunction of constraints with a common contingent link. Figure 17(a) shows an easy situation, where $E_c^I = \{[-\infty, a]\}$, $E_c^{II} = \{[b, \infty]\}$ and $a \geq b$, so the union of envelopes I and II covers the whole space. It 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 17(b) illustrates a general and feasible situation. Envelopes I and II cover the area from the lower bound of E_c^I to the upper bound of E_c^{II}. The intersections of the observation bounds $[l_c, u_c]$ with the edges of Envelopes I and II 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 17(c) illustrates an infeasible situation. The uncertainty is so significant that the combined envelope cannot provide a viable range for the prehistory before the observation.



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

Figure 17: Combined Envelope

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.

5.4 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, and a separate check is done for each disjunct.

If the problem is not solved during the combining process at any interior nodes 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.

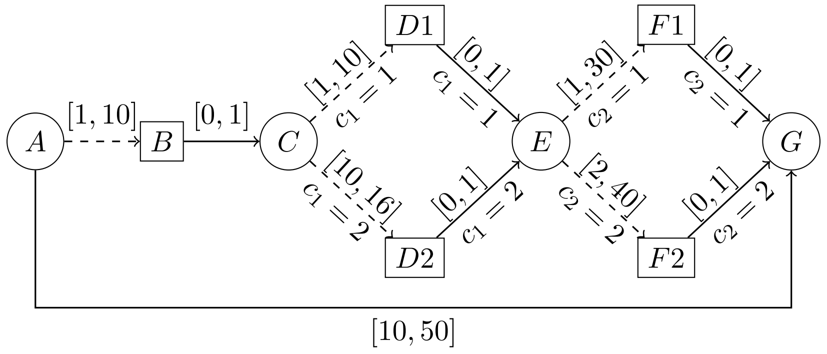


Figure 18: An example of CCTPU with two discrete variables

We explain the combining process and node checking with an example with two discrete variables, shown in Figure 18. The circles are controllable nodes, the squares are uncontrollable nodes and the diamond nodes are latest decision time points. The dashed lines are contingent links and the solid lines are requirement links. There two decision variables c_1 and c_2 and each of them has two options.

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 (due to Assumption 3). 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 $\{c_1 = 1\}$ when making decision at D_1 is the solution space of the following constraints:

$$\begin{aligned} & l'_{AB} + u'_{BC} + l'_{CD_1} + u_{D_1E} + l_{EF_1} + u_{F_1G} - l_{AG} \geq 0 \\ \vee & \begin{cases} -u'_{AB} - l'_{BC} - u'_{CD_1} - l_{D_1E} - u_{EF_2} - l_{F_2G} + u_{AG} \geq 0 \\ l'_{AB} + u'_{BC} + l'_{CD_1} + u_{D_1E} + l_{EF_2} + u_{F_2G} - l_{AG} \geq 0 \end{cases} \end{aligned} \tag{15}$$

where the variables x' are links before the decision time point, variables x are links not before the decision time point and the two disjuncts of constraints represent the conflict resolutions of the two options.

Because of the decision consistency in the prehistory, variables of links that are not the critical observation have to represent the consistent decision (See Equation 14). Equation (15) can be rewritten as

$$\begin{array}{ll} & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 \quad (c_2 = 1) \\ \vee & \left\{ \begin{array}{ll} p'_{AB} + p'_{BC} + x'_{CD_1} \leq 10 & (c_2 = 2) \\ p'_{AB} + p'_{BC} + x'_{CD_1} \geq 6 & (c_2 = 2) \end{array} \right. \end{array} \tag{16}$$

Here, the bounds of each relaxable link in the prehistory have been replaced with a single variable p_{ij} representing the projection, and the bounds of the key observation likewise with a variable x_{CD_1}. The bounds of the non-relaxable links, which are just constants, have been added up on the right-hand side of each inequality.

Equation 16 is then expanded to a conjunction of disjunctions:

$$\begin{array}{ll} & \left\{ \begin{array}{lll} & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 & (c_2 = 1) \\ \vee & p'_{AB} + p'_{BC} + x'_{CD_1} \leq 10 & (c_2 = 2) \end{array} \right. \\ \wedge & \left\{ \begin{array}{lll} & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 & (c_2 = 1) \\ \vee & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 6 & (c_2 = 2) \end{array} \right. \end{array} \tag{17}$$

The first branch in Equation (17) 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

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

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 at least 6. Since we have from the combined envelope (Equation 16) that $c_2 = 1$ if viable when $p_{AD_1} \geq 7$ and $c_2 = 2$ when $p_{AD_1} \in [6, 10]$, at D_1 if $p_{AD_1} \in [6, 7]$, then choose $c_2 = 2$, if $p_{AD_1} \in [7, 10]$, we can choose $c_2 = 1$ or $c_2 = 1$, and if $p_{AD_1} \geq 10$ the only choice is $c_2 = 1$. (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.

6. Correctness

If the approach in Section 5 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 of 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.

In this section, we prove that (1) given an STPU, the conflicts extracted by Algorithm 1 is a complete set of conflicts; (2) given a CCTPU, the solution of our method has a dynamically controllable execution strategy with the dynamic assignment on discrete variables satisfying assumptions we have made. By proving (1), we show the correctness and completeness of extracting DC envelopes of STPU.

6.1 Validation of Dynamically Controllable Envelopes of STPU

In section 4, the modified dynamic controllability checking algorithm extracts a complete set of conflicts of a given STPU. Using disjunctive linear constraints as conflict resolutions, we represent the dynamically controllable envelopes of STPU by the solution space of the constraint model. To validate the correctness and completeness of our method, we prove that (1) all solutions satisfying constraints of their dynamically controllable envelope are dynamically controllable; (2) a dynamically controllable STPU satisfies its dynamically controllable envelope constraints.

In order to prove correctness, we introduce the following theorem.

Theorem 6.1. *If in a conflict, one negative link has more than one occurrences and one occurrence can propagate to another occurrence via a negative path, this conflict resolution can be the sum of other conflict resolutions.*

Proof. We use $\{e_1^-, e_2^-, ..., e_n^-,\}$ to represent the existence of negative link e^- in conflict C. e_i^- can propagate to e_j^- via a negative path in C. The distance of the negative path on C is $D_C(start(e_i^-), start(e_j^-))$ which is negative. Because $start(e_i^-)$ and $start(e_j^-)$ are the same node, the negative path is a negative cycle which can be resolved by a conflict resolution. If the rest part of C is negative, it is another conflict $D_C(start(e_j^-), start(e_i^-)) < 0$. Otherwise, C can be resolved by resolving $D_C(start(e_i^-), start(e_j^-))$. □

Proof. Let C be a conflict, and e^- a negative link that occurs more than once in C. Let $e_1^-, e_2^-, ..., e_n^-$ represent the ordered occurrences of e^- in C. Suppose e_i^- can propagate to e_j^- via a negative path in C. The distance of the negative path on C is $D_C(start(e_i^-), start(e_j^-))$ which is negative. Because $start(e_i^-)$ and $start(e_j^-)$ are the same node, this path is actually a negative cycle which can be resolved by a conflict resolution. If the remaining part of C is negative, it is another conflict $D_C(start(e_j^-), start(e_i^-)) < 0$. Otherwise, C can be resolved by resolving $D_C(start(e_i^-), start(e_j^-))$. □

Thus, multiple occurrences of a negative link in a conflict must be part of added links, not the start of any negative sequences. In other words, those multiple occurrences are not in the piled-up *backpropagation* processes (of Morris's cubic algorithm and our algorithm). Their *backpropagation* has been completed before finding the negative cycle. With Theorem 6.1, our algorithm only has to propagate one negative link once in a recursive process.

To prove the correctness, we only need to prove we found all conflicts that do not have multiple repetitions of a single negative link or where the distances among repetitions are non-negative. We also use Theorems 4.1 and 4.2 proved by Morris (2006, 2014) to prove the following:

Theorem 6.2. ***Correctness:*** *The solutions satisfying constraints of conflict resolutions extracted by the modified dynamic controllability checking method (Algorithm 1) are dynamically controllable.*

Proof. To prove the correctness, we suppose conversely there is a solution N satisfying constraints of conflict resolutions extracted by the modified DC checking method but not dynamically controllable. According to Theorem 4.1 N has a semi-reducible negative cycle C. By Theorem 4.2, C can be found by DCbackprop procedure in the Morris's DC checking algorithm.

Based on the process of DCbackprop procedure, C results from a series of calling DCbackprop on negative edges $\{e_1^-, e_2^-, ..., e_k^-\}$ and each propagating backwards through positive edges. These positive edges are either original or added edges (the added edges are positive paths in DCbackprop procedure, see line 9 in Algorithm 3).

First, we prove the case that all edges in C are original edges. The structure of C is shown in figure 19, where $\{p_1^+, p_2^+, ..., p_k^+\}$ are the paths of positive links propagated through, $\sum_i (e_i^- + p_i^+) < 0$ and $e_i^- + p_i^+ < 0$ for all i. Calling backpropagation from one negative edge e_i^- in C, all negative paths starting from e_i^- including p_i^+ in C will be enumerated by the DFS process until meeting other negative edges (line 17 in Algorithm 3) including the next negative edge e_{i+1}^-. Another backpropagation process will be called when meeting e_{i+1}^-. The recursive calls result in a termination of calling an ancestor negative link e_i^-. Therefore, C will be found. It contradicts that N satisfying the constraint of all conflict resolutions.

Then we consider the situation where C consists of original and added edges. In this situation shown in Figure 20, where e'^+_{add} is the added edge, which split p_i^+ into two parts, from the propagation of e_x^- through p_x^+ and $e_x^- + p_x^+ > 0$, the rest edges are kept as before. In this case, we have to prove that e'^+_{add} is added while propagating e_i^- (line 2 in Algorithm 3)to the end point of e_x^-. There are four situations when calling backpropagation on e_x^- in the backpropagation of e_i^-: (1) it is an ancestor call, (2) the backpropagation of e_x^- is successfully terminated before, (3) the backpropagation has not been called or (4) it has been unsuccessfully called. In situation (1), C is a conflict that contains a negative path between two occurrences of e_x^-, so it can be resolved by other found conflict resolutions as shown by Theorem 6.1. In situation (2), e'^+_{add} has been added. In situation (3) and (4), e'^+_{add} is going to be added unless, in p_x^+, there is a negative link that has been called in an open ancestor call, which makes C contain a negative path between two existences of that negative link.

Therefore, C will be found. It contradicts that N satisfying the constraint of conflict resolution of C. □

Theorem 6.3. ***Completeness:*** *Every dynamically controllable STPU that is a relaxed instance (tightening contingent links) of the given STPU satisfies the constraints of conflict resolutions extracted by the modified dynamic controllability checking method (Algorithm 1).*

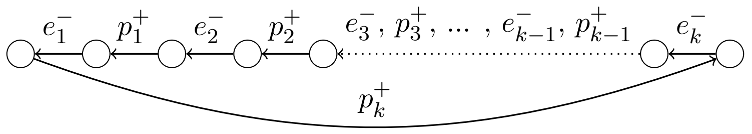
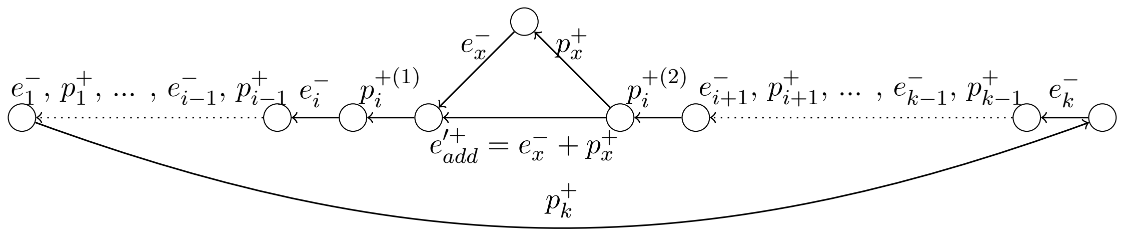


Figure 19: A semi-reducible negative cycle without added edges



e_x^-
p_x^+
$e_1^-, p_1^+, \dots, e_{i-1}^-, p_{i-1}^+$
e_i^-
$p_i^{+(1)}$
$p_i^{+(2)}$
$e_{i+1}^-, p_{i+1}^+, \dots, e_{k-1}^-, p_{k-1}^+$
e_k^-
$e_{add}'^+ = e_x^- + p_x^+$
p_k^+

Figure 20: A semi-reducible negative cycle with added edges

Proof. We assume there is a dynamically controllable STPU N that is a relaxed instance of the given STPU and N does not satisfy the constraints of conflict resolutions extracted by the modified DC checking method.

Thus, N violates a constraint CR_0 in the form of Equation 7, which means it violates every branch of the disjunctive linear constraint. However, the variables in CR_0 represent the bounds of links in conflict C_0. The dissatisfaction of N means the bounds of links of N formulate a conflict the same as C_0, which contradicts that N is dynamically controllable. □

6.2 Validation of Dynamic Controllability of CCTPU

In this subsection, we show that if a CCTPU is found to be dynamically controllable by our approach, there is a viable strategy that makes both temporal decisions and discrete variable assignments dynamically.

To validate our approach, we add the dynamic decisions of discrete variables to the dynamic execution algorithm for STPUs by Morris and Muscettola (2000). The dynamic decisions of discrete variables contain its decision time point $DT(c)$, which is the end of a contingent link or the start time point, and $A(c)$ based on the situation of the prehistory. If a CCTPU is dynamically controllable, it has a dynamically controllable envelope $Env(PA)$ that covers all uncertain situations in its prehistory. $Env(PA)$ is either in a leaf node when PA is A, or combined from child nodes that have one more assignment of a discrete variable c. If the algorithm finds a dynamically controllable STPU with a full assignment, its execution assigns all discrete variables at the beginning and executes the CCTPU as an STPU. Otherwise, a sequence of $DT(c)$ and $A(c)$ can be extracted from tracing back the combining process. Therefore, the execution is represented in Figure 21. The executions of discrete variables are added in the lines with star marks, the rest lines are the same as executing an STPU.

```
Algorithm: ExecuteCCTPU
Input: A dynamically controllable CCTPU $N$.

0  Perform initial propagation from the start time point.

1* Assign discrete variables whose decision time has been reached, according to their
   prehistory and the envelope calculated. If there is more than one such variable, assign
   them following the same chronological order used to prove dynamic controllability of
   $N$. Then immediately execute any controllable time points that have reached their
   upper bounds.

2  Arbitrarily pick a controllable time point TP that is live and enabled and not yet
   executed, and whose waits, if any, have all been satisfied.

3  Execute TP. Halt if network execution is complete. Otherwise, propagate the effect
   of the execution.

4* Advance current time, propagating the effect of any contingent time points that occur,
   until either
   • the contingent time point is the decision time of a discrete variable $c$: go to 1*; or
   • a controllable time point becomes eligible for execution under 1* or 2.

5  Go to 1*.
```

Figure 21: Execution of the dynamic strategy of CCTPU

Figure 21 shows the general dynamic execution strategy for a CCTPU that has been shown to be dynamically controllable. It follows the execution algorithm for STPUs provided by Morris et al. (2001), with instructions for assignment of discrete variables added to steps 1 and 4 (marked with asterisks); otherwise the algorithm is the same as the original.

The critical step is (1*), where the decision to assign a discrete variable is taken. Let PA be the partial assignment made so far, and suppose c is the next variable to be assigned: the algorithm selects a value d for c such that the observed values of contingent links in the prehistory of c belong to $Env(PA \cup \{c = d\})$. At the end of execution, the complete assignment A defines an STPU that consists of only the activated links. The scheduling of controllable time points made by the algorithm satisfies all requirement links of that STPU Morris et al. (2001). It remains for us to show that at each decision time point $DT(c)$ all contingent links in $P_{PA}\{\prec c\}$ have been observed and that there exists a value $d \in D(c)$ such that the observed prehistory belongs to $Env(PA \cup \{c = d\})$. The first part is immediate from the definition of the precedence relation and Assumption 2.

Because the checking algorithm has found the CCTPU to be dynamically controllable, there exists a (possibly empty) partial assignment PA such that $Env(PA)$ covers every outcome of the contingent links in its prehistory. This assignment can be made at the start of execution. Consider the next variable, c, to be assigned. $Env(PA)$ is a subset of

$Env(PA, c)$, which relaxes only contingent links in $P_{PA}\{\prec PA\}$; thus it covers any outcome of any contingent link that is in the prehistory of c but not the prehistory of the PA, and thus the observed outcome before $DT(c)$ is also in $Env(PA, c)$. But $Env(PA, c)$ is simply the union of $Env(PA \cup \{c = d\})$ for all $d \in D(c)$. Thus, there is some value for c whose envelope contains the observed outcome.

Hunsberger (2009) introduced the Real-Time Execution Decision (RTED) formalism to describe the actions that an agent may take while executing an STPU that is dynamically controllable. It could also be used to represent the dynamic execution strategy for a CCTPU.

7. Experimental Results

We illustrate the benefit of fully dynamic strategies for CCTPUs 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 each leaf node (full set of assignments) in turn and considers the CCTPU dynamically controllable if it finds one leaf that induces a dynamically controllable STPU.

7.1 Experimental Setup

We use the benchmark generator by Yu (2016), based on Zipcar problems (Yu & Williams, 2013; Yu et al., 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 between options 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 missions, which randomly deviates by $\pm 20\%$ from the estimated bounds of the sequence of activities.

We generated 16000 test cases, with 1–8 discrete variables and 1–10 options for each variable. Regarding the size of the networks, the numbers of nodes, links and contingent links are in the ranges 11–170, 11–330 and 4–162, respectively.

7.2 Results

The result is shown in Figure 22. The tests are grouped by DC checking results in the chart: "infeasible" means the problem is not dynamically controllable; "feasible with fixed option" means the algorithm found a viable strategy with a static assignment of the discrete variables; and "feasible with dynamic options" means it found a strategy that both assigns discrete variables and schedules time points dynamically.

The white bars represent the result of our implementation with a fixed assignment. 77.1% of the test cases are infeasible when using a static assignment, compared to just 22.4% which are still infeasible when assigning also discrete variables dynamically. The number of instances in which the implementation of our fully dynamic algorithm found a strategy with a static assignment is slightly fewer than the number that can actually be solved by such a strategy; this is because the algorithm sometimes finds a viable strategy with dynamic assignment earlier in the search, and then stops. Those instances are counted in the last group, rather than in the middle.

Figure 23 shows the runtime comparison between the implementations of dynamic controllability with fixed and dynamic assignments. The y-axis describes the number of problems solved, where *solved* means the implementation terminates with a checking result of dynamically controllable or not. 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 at runtime limits over 1 second, which may be because in test cases that have dynamic strategy with dynamic assignment, it terminates before exploring the whole search tree.

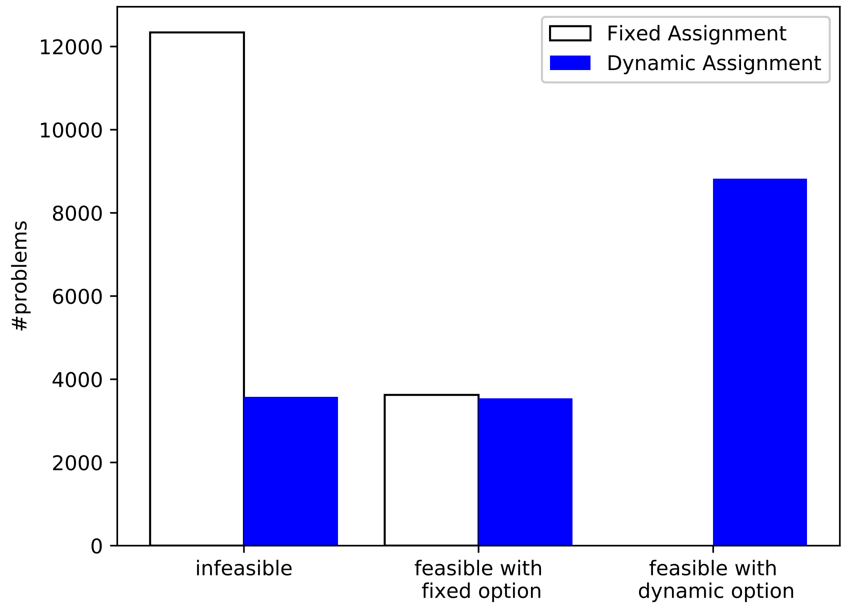


Figure 22: Distribution of the number of instances for which a strategy with fixed and dynamic assignments of the discrete choices, respectively, was found.

7.3 A Simple Optimisation Experiment

Answering the question how controllable, flexible, robust or generally well a temporal problem with uncertainty and choices could be provides more information than just checking the feasibility or controllability of the problem. Furthermore, it is more challenging to answer those questions because we need to formulate and solve an optimisation problem instead of checking feasibility.

It is still an open problem that how to formulate an optimisation model which can represent the fully dynamic strategy for a CCTPU. The difficulties of formulating the optimisation model are associating decision time points to the temporal networks, representing the "splitting" of uncertainty of key contingent links dealt by each branch, and so on. How-

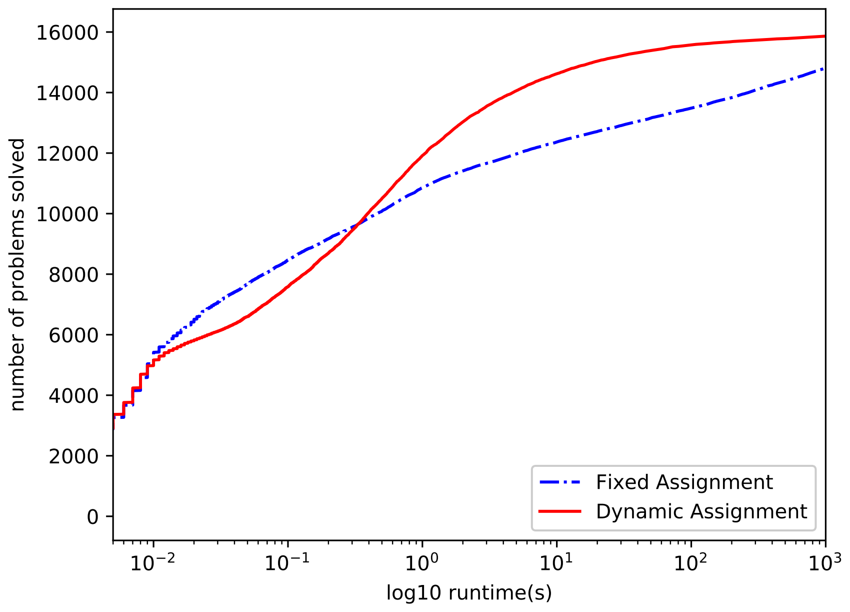


Figure 23: Number of problems solved as a function of runtime.

ever, besides a complete optimisation model, we can use alternative methods to perform optimisation, for example the framework of the binary search with DC checking at the core.

In this section, we present a simple optimisation experiment that is based on the dynamic controllability checking algorithm introduced in Section 5 to further demonstrate the advantage of making assignments dynamically. The experiment is based on optimising a robustness measure called maximum deviation, introduced by Cui, Yu, Fang, Haslum, and Williams (2015). It maximises the deviation (delay) of uncertain durations under which the problem is still dynamically controllable. It is a worst-case measure, so after setting the maximum deviation, the worst case is a CCTPU. Thus, we can use the algorithm introduced in this paper to test how dynamically controllable the worst case is.

In this experiment, we set the lower bounds of the contingent links to their original lower bounds and upper bounds are the sum of their lower bounds and the maximum deviation. The maximum deviation is found by a binary search on top of the checking algorithm.

Our test set comprises 2000 networks with one discrete variable. The result of the experiment is shown in Figure 24. In less than half of the test cases, the worst case deviation that can be handled by dynamic control remains the same if we use dynamic assignments instead of fixed assignments. In 171, 302, 336 and 285 test cases, the maximum deviation improves by up to 10%, 20%, 30%, and more than 30%, respectively.

8. Related Work

In addition to the techniques and methods that we have used as a basis in this paper, we discuss other related work in this section.

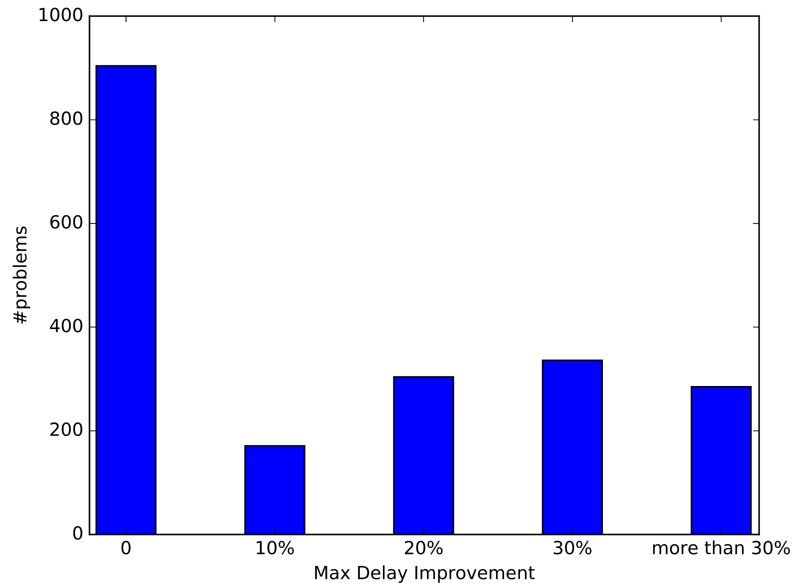


Figure 24: Improvement of Max Delay from Fixed Assignment to Dynamic Assignment

8.1 Related Temporal Reasoning Models

In the recent trend of introducing various variants of temporal networks, CCTPU is not the only one that considers uncertainty, conditions and discrete choices. In this subsection, we discuss three temporal reasoning models – CSTNU, DTNU and CSTNUD – that also adds these factors to temporal networks.

8.1.1 Conditional Simple Temporal Network with Uncertainty

Among the variety of temporal reasoning models, the Conditional Simple Temporal Network with Uncertainty (CSTNU) is closely related to the CCTPU. Both CSTNU and CCTPU consider temporal problems with uncertainty and discrete conditions, but the CSTNU models uncontrollable and observable conditions.

The Conditional Simple Temporal Network with Uncertainty (CSTNU) (Hunsberger et al., 2012) combines the Conditional Simple Temporal Network (CSTN) and Simple Temporal Network with Uncertainty (STNU). A CSTN attaches logical conditions, formulated over a set of binary proposition symbols (P), to time points and links, with the interpretation that only time points whose conditions are true will be executed, and only requirement links whose conditions are true need to be satisfied. An assignment of truth values to the propositions is called an *execution scenario*, and represents a particular outcome or mode of execution of the temporal network.

The CSTNU adds to the CSTN contingent links, which are links ending in uncontrollable time points just like in an STNU. The label of an uncontrollable node must be the same as the label of the contingent link that ends in it, and of the controllable node at the start of that link, so that a contingent link is executed in full if and only if it is started. In a CSTNU, the execution scenario is initially unknown. Each proposition p is associated with an observation time point, $O(p)$, which is the time point when the agent will find out the value of p. Hence, the uncertain situation in a CSTNU is a combination of the uncertain execution scenario, sc, and the projection of the contingent links, ω, which is the same as the projection of an STNU. The pair (sc, ω) is called a *drama*. The projection of a CSTNU onto a drama, denoted by $drPrj(sc, \omega)$, is an STN. An *execution strategy* for a CSTNU is a mapping from dramas to schedules of the controllable time points. The controllability of a CSTNU considers how well the execution strategy can deal with the drama. If an execution strategy is able to solve every drama of the CSTNU, it is dynamically controllable.

An algorithm for checking the dynamic controllability of a CSTNU was introduced by Combi, Hunsberger, and Posenato (2014). It is based on the algorithm for checking dynamic controllability of STNU (Morris et al., 2001) which has a complexity of $O(N^5)$. The authors revised the reduction rules by considering the consistency of labels and adding label modification rules to the process to tackle the conditions of the CSTNU.

8.1.2 Conditional Disjunctive Temporal Networks with Uncertainty

Another temporal reasoning model that considers uncertainty and options is the Disjunctive Temporal Networks with Uncertainty (DTNU) (Venable & Yorke-Smith, 2005). The DTNU extends the STNU by considering disjunctive temporal constraints. Each requirement constraint is a disjunction of temporal constraints between any pairs of nodes. Each contingent link has a set of disjoint intervals, instead of a single interval, for the uncertain duration.

The uncertain situations and schedules of the DTNU are not very different from those of the STNU. A strongly controllable DTNU has a time assignment to every controllable time point that ensures all constraints will be satisfied (Peintner et al., 2007). The strong controllability of disjunctive temporal problems can be solved by SMT (Cimatti, Micheli, & Roveri, 2015). A dynamically controllable DTNU has a dynamic strategy that, like the dynamic strategy of an STNU, depends only on past observations, and ensures all constraints are satisfied for any outcome of the contingent durations (Venable, Volpato, Peintner, & Yorke-Smith, 2010; Cimatti, Micheli, & Roveri, 2016).

A DTNU that does not have disjunctions on contingent links can be approximately modelled as a CCTPU by introducing a discrete variable for each requirement constraint that has disjunctions and adding labels accordingly. This, however, disregards the difference in *when* decisions are made: a dynamic execution strategy for a DTNU is only required to eventually satisfy one disjunct in each disjunctive constraint, whereas a dynamic strategy for a CCTPU, under our assumptions, is required to choose which option will be satisfied before scheduling any time point that is conditioned on the choice.

The difference is illustrated by the example in Figure 25: the disjunctive constraint $B - A \in [30, 60] \vee B - A \in [60, 90] \vee B - A \in [90, 120]$ is of course satisfied by any outcome of the contingent link A–B, but there is no way to make a choice of which disjunct will be true before observing it.

There is also no way to express disjunctive contingent links in a CCTPU, since these correspond to uncontrollable discrete choices, made by nature rather than the executing agent.

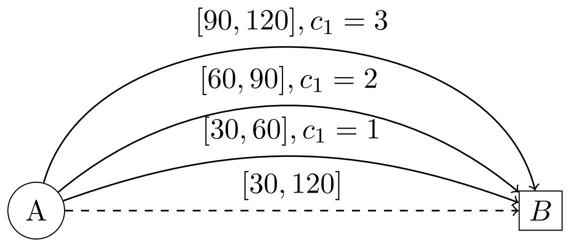


Figure 25: Illustration of the difference between a choice and a disjunctive constraint.

Conversely, not every CCTPU can be modelled as a DTNU since the controllable discrete choices in the CCTPU can model disjunctions of conjunctions of temporal constraints, but the DTNU is not able to represent these conjunctions, i.e., to enforce a correlation between the choices of disjuncts to satisfy different constraints. Also, the CCTPU allows to model a controllable choice of which contingent links to activate or deactivate, which is also not expressible in the DTNU. Neither type of problem subsumes the other.

The Conditional Disjunctive Temporal Network with Uncertainty (CDTNU) is a combination of CSTNU and DTNU, which models uncontrollable conditions and disjunctive constraints at the same time (Cimatti, Hunsberger, Micheli, Posenato, & Roveri, 2016). However, this also does not subsume the CCTPU, as it also does not allow to model a controllable choice between uncertainties.

8.1.3 Conditional Simple Temporal Networks with Uncertainty and Decisions

Zavatteri (2017) introduced a temporal reasoning model called the Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUD) which considers uncontrollable and controllable conditions at the same time. The controllable conditions are named *decisions* in the CSTNUD. A **CSTNUD** is a tuple, $< V, E, L, OV, DV, O, P, \ell, O >$, where:

- $< V, E, L, OV, O, P, \ell >$ is a CSTNU,
- DV is a set of decision time points such that $DV \cap OV = \emptyset$,
- $O : P \to OV \cup DV$ is a bijection associating a unique observation or decision time point to each proposition. If $O(p) \in OV$, p is observable (and uncontrollable), whereas if $O(p) \in DV$, p is a controllable decision.

The proposition set is $P = OP \cup DP$, where OP are the uncontrollable conditions that are observable, and DP are the controllable conditions.

The CCTPU and CSTNUD temporal reasoning models have some differences. The CCTPU model does not include uncertainty over discrete choices, as modelled by the uncontrollable conditions in the CSTNUD. On the other hand, in the CCTPU we have defined the mapping from discrete variables to their decision time points as part of the solution strategy, whereas in the CSTNUD they are given a fixed decision time point as part of the problem statement. Although smart choices of fixed decision time points could be as good as dynamic decision time points, it depends on the wellness of the model formulation.

For example in Figure 26, the choice of c_1 can be made at B or D. If we model the problem by CSTNUD (without any uncontrollable conditions) and assign the fixed decision time point at D, we cannot find a feasible strategy since the envelopes $[16, 20]$ when $c_1 = 1$ and $[10, 15]$ when $c_1 = 2$ cannot cover the uncertain situations from A to D (there is a gap $[15, 16]$). In contrast, if we formulate the problem by CCTPU, the algorithm introduced in this paper will try both B and D to be the decision time point of c_1 and the decision made at B is part of a dynamically controllable strategy. Because the envelopes $[2, 10]$ when $c_1 = 1$ and $[1, 5]$ when $c_1 = 2$ can cover the uncertain situations from A to B. Even though the CSTNUD model may have the same feasible solution by modelling the fixed decision time point at B, it is not guaranteed.

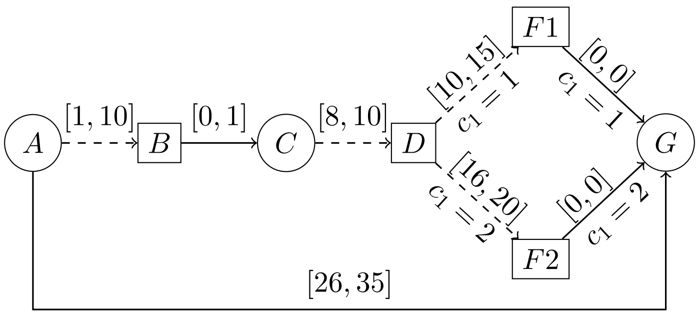


Figure 26: An example to show the benefit of dynamic decision time point

The dynamic controllability of a CSTNUD can be decided by a reduction to a Timed Game Automaton (Zavatteri, 2017). We discuss this method in the next subsection.

8.2 Verification Approaches for Temporal Problems with Uncertainty

The DC checking algorithm in this paper is based on a constraint programming technique – bound propagation. However, the controllability of different temporal problems can also be represented using formalisms such as Timed Game Automata (TGA), and solved with TGA verification tools.

8.2.1 Representing Dynamic Controllability by Timed Game Automata

Cimatti et al. (2016) used Timed Game Automata (TGA) to represent the dynamic controllability of STNU, CSTNU, DTNU and CDTNU. Using TGA to represent the dynamic controllability is the first sound and complete approach for DTNU and CDTNU. Zavatteri (2017) also used TGA to represent the dynamic controllability of the CSTNUD, and implemented the model in UPPAAL-TIGA (Behrmann, Cougnard, David, Fleury, Larsen, & Lime, 2007), which is a verification tool for TGA. In the following, we will briefly outline the reductions of some temporal reasoning models to TGA, and discuss the possibility and practicality of modelling dynamic controllability of the CCTPU as a TGA.

A Timed Automaton (TA) (Alur & Dill, 1990) is a tuple $< \Sigma, S, s_0, C, E >$, which adds a finite set of real-valued clocks C to a finite automaton $< \Sigma, S, s_0, E >$, where Σ is an alphabet, S is the finite set of states (also known as "locations"), s_0 is the initial state and E are the transitions. A transition represents the change from one state to another state on input. Each transition of a TA has a *guard* condition over clocks, which describes the requirement to make the transition, and a set of clocks that will be reset to zero when the transition is taken.

A Timed Game Automaton (TGA) (Maler, Pnueli, & Sifakis, 1995) divides the set of transitions into controllable and uncontrollable. Formulating the dynamic controllability of STNU as a TGA, Cimatti et al. (2016) used two states in which the agent or environment, respectively, execute their transitions, and a goal state to model the agent achieves the goal. Thus, the agent's aim is to find a counter-strategy to reach the goal location against the prevention from the environment. The TGA model of an STNU has one clock that measures global time, which is never reset, one clock that measures time since the last contingent time point, and one clock for each time point which is reset once, when that node is executed.

To model the uncontrollable conditions of the CSTNU, the TGA is augmented with a clock for each proposition and a controllable transition for the location representing the environment that resets this clock if the proposition is observed to be true. That the proposition was observed to be false can be inferred if the observation time point has been executed and this clock has not been reset. In the same way, a clock and transition pair is added to model the decisions in a CSTNUD (Zavatteri, 2017).

It seems plausible that dynamic controllability of the CCTPU, with dynamic discrete choices, can also be modelled as a TGA, and therefore decided using TGA verification tools. As mentioned above, the only additional complication required to model a CCTPU that is not already present in the CSTNUD is allowing the agent to chose the decision time point for controllable variables. This could be potentially be handled using two different clocks to represent when the decision is taken and what the decision is, essentially introducing a new time point representing the decision time, which is constrained to be executed no later than any link whose label depends on the decision. However, we have not developed such an encoding in detail. Moreover, even if the TGA approach is feasible, verifying dynamic controllability by using bound propagations may be more efficient. Zavatteri (2017) solved an example with two decisions, 4 contingent links and 11 nodes by implementing the TGA in UPPAAL-TIGA, which took about one minute (running on a virtual machine with Intel i7 2.8GHz CPU and 5G RAM). Solving a problem of the same size with the DC checking algorithm in this paper needs less than a second (running on a desktop with Intel i5 3.2GHz CPU and 4G RAM). Furthermore, solving the optimisation, rather than decision, version of the problem (relaxing an over-constrained problem) in the same setting takes about five seconds. While this comparison is limited and not exactly rigorous, it does suggest that the greater generality of the TGA as a method of implementing dynamic controllability checking comes at a computational cost

8.2.2 Other Approaches for Temporal Problem Verification

In the context of uncertainty, the robust execution of plans or schedules is a main concern of automated systems. A reactive execution strategy can respond when uncertainty is resolved through observation; a proactive-reactive approach (Herroelen & Leus, 2005) involves some combination of planning, i.e., formulating a strategy ahead of execution, and leaving some flexibility in the strategy to react at execution time. Verifying dynamic controllability is an example of a proactive-reactive approach, but the concept exists more widely in the scheduling and AI literature.

Musliner, Durfee, and Shin (1993) used a two-level architecture to build a real-time control system, called CIRCA, focusing on meeting hard deadlines. CIRCA has an AI subsystem focused on task level goals and a real-time subsystem that can achieve controllability. It sacrifices completeness of the task level calculation to guarantee a certain notion of safety in the real-time system. Muscettola, Nayak, Pell, and Williams (1998) developed a real-time system that focuses on tight deadlines, resource constraints and concurrent activities for the spacecraft domain that works over long periods of time. A dynamic execution strategy for a temporal problem achieves task level completeness under the assumption that the uncertain durations are within certain bounds. However, using intervals to describe the durations of uncertain events sacrifices some accuracy. Verifying dynamic controllability can be viewed as a proactive process with assumptions on the reactive process, which guarantees that the process will satisfy constraints at execution time as long as the environment satisfies the assumption. Furthermore, using flexibility metrics related to dynamic controllability to design the reactive process can be an intelligent way to enhance its robustness. For example, in a scheduling problem, adding temporal slacks to achieve dynamic controllability can be more efficient than adding them uniformly.

9. Conclusion and Future Work

In this paper, we extend the definition and verification of dynamic controllability to temporal problems with uncertainty and controllable options, the CCTPU, which can solve the problem with an entirely dynamo control strategy in which both temporal scheduling and controllable options are decided dynamically. The dynamic decisions on the discrete options enable a fully flexible strategy in which every decision is based on past observations. Compared to previous work on dynamic control of the CCTPU, which considered making only scheduling decisions dynamically and discrete choice assignments statically, we showed that fully dynamic strategies can be found in more test cases.

Since our dynamic controllability checking algorithm relies on a set of assumptions, relaxing those assumptions is one of our directions for future work. Although some of our limiting assumptions can be circumvented by remodelling the problem, finding a more intelligent approach that can describe the dynamic decision time points would be a useful extension.

The addition of controllable discrete variables makes the CCTPU a better model for many problems than the STPU, but also more difficult to solve. Many real-world problems that can be represented as a CCTPU are not controllable even with dynamic variable assignment, because the temporal constraints are too tight or the uncertainty is too high. Instead of only checking dynamic controllability, it is in this situation more useful to answer how far from controllable the problem is. On the other hand, schedules made without considering uncertainty may fail when executing in uncertain circumstances. Similar to the optimisation problems of STPU (Cui et al., 2015; Casanova, Pralet, Lesire, & Vidal, 2016), an optimisation model of the CCTPU with fully dynamic strategy can answer how robust or flexible these schedules are. Thus, developing approaches to CCTPU relaxation and optimisation is also an important direction for future work.

Extending the verification of dynamic controllability to more general decision-making problems such as scheduling and planning is also a potential are of future work. Controllable discrete variables can be used to model different ways of resolving resource allocation conflicts that arises when some activities cannot be executed at the same time because the sum of their resource demands exceeds the resource capacity. Different ways to add precedence constraints to resolve the conflict can then be modelled as the values of a controllable discrete variable. Using a CCTPU to represent a set of different partial-order schedules for a scheduling problem enhances the flexibility contained in the solution set beyond the flexibility that is afforded by one partial-order schedule. More robust or flexible schedules may be achieved by modeling resource conflicts as a CCTPU and using the method in this paper to verify if this problem is dynamically controllable or not. However, encoding all resource conflicts and their resolutions is likely to result in a too large CCTPU. Furthermore, planning problems are more complicated than scheduling problems. (Cimatti, Do, Micheli, Roveri, & Smith, 2018) tackled the problem of temporal planning with uncontrollable action durations and applied strong controllability for temporal networks to the planning problem. A possible compromise is to use controllable discrete variables to describe the choice of alternative plans obtained from planners, then analysing robustness, controllability and other performances of the plan set.

10. Acknowledgement

We would like to thank Peng Yu and Cheng Fang for providing the data sets of the Zipcar problem and discussing the work. We would like to thank Caroline Even for providing the test cases of the evacuation planning problem. We also would like to thank the JAIR reviewers and editor for their insightful and helpful suggestions, which helped us consider the work in different views and make the paper more solid and understandable.

References

Alur, R., & Dill, D. (1990). Automata for modeling real-time systems. In Paterson, M. S. (Ed.), *Automata, Languages and Programming: 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings*, pp. 322–335, Berlin, Heidelberg. Springer Berlin Heidelberg.

Barták, R., & Čepek, O. (2007). Temporal networks with alternatives: Complexity and model. In *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference*, pp. 641–646.

Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., & Lime, D. (2007). Uppaal-tiga: Time for playing games!. In *Computer Aided Verification: 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007. Proceedings*, pp. 121–125, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bhargava, N., Vaquero, T., & Williams, B. (2017). Faster conflict generation for dynamic controllability. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4280–4286. AAAI Press.

Casanova, G., Pralet, C., Lesire, C., & Vidal, T. (2016). Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming. In *Proc. 22nd European Conference on Artificial Intelligence (ECAI)*, pp. 930–938.

Cimatti, A., Do, M., Micheli, A., Roveri, M., & Smith, D. E. (2018). Strong temporal planning with uncontrollable durations. *Artificial Intelligence*, *256*, 1 – 34.

Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., & Roveri, M. (2016). Dynamic controllability via timed game automata. *Acta Informatica*, *53*(6), 681–722.

Cimatti, A., Micheli, A., & Roveri, M. (2015). Solving strong controllability of temporal problems with uncertainty using smt. *Constraints*, *20*(1), 1–29.

Cimatti, A., Micheli, A., & Roveri, M. (2016). Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 3116–3122. AAAI Press.

Combi, C., Hunsberger, L., & 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)*, pp. 314–331.

Conrad, P. R., & Williams, B. C. (2011). Drake: An efficient executive for temporal plans with choice. *J. Artif. Int. Res.*, *42*(1), 607–659.

Cui, J., & Haslum, P. (2017). Dynamic controllability of controllable conditional temporal problems with uncertainty. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 61–69.

Cui, J., Yu, P., Fang, C., Haslum, P., & Williams, B. (2015). Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 52–60.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artif. Intell.*, *49*(1-3), 61–95.

Even, C., Pillac, V., & Van Hentenryck, P. (2014). Nicta evacuation planner: Actionable evacuation plans with contraflows. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI)*, ECAI'14, pp. 1143–1148, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289 – 306. Project Management and Scheduling.

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)*, pp. 155–162.

Hunsberger, L. (2015). New techniques for checking dynamic controllability of simple temporal networks with uncertainty. In *Agents and Artificial Intelligence*, Vol. 8946 of *Lecture Notes in Computer Science*, pp. 170–193. Springer International Publishing.

Hunsberger, L., Posenato, R., & 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*, pp. 2–4.

Maler, O., Pnueli, A., & Sifakis, J. (1995). On the synthesis of discrete controllers for timed systems. In Mayr, E. W., & Puech, C. (Eds.), *STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science Munich, Germany, March 2–4, 1995 Proceedings*, pp. 229–242, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

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

Morris, P., & Muscettola, N. (2000). Execution of temporal plans with uncertainty. In *In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pp. 491–496.

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

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

Muise, C. (2014). *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution*. Ph.D. thesis, University of Toronto.

Muise, C., Beck, J. C., & McIlraith, S. A. (2016). Optimal partial-order plan relaxation via maxsat. *Journal of Artificial Intelligence Research (JAIR)*, *57*(1), 113–149.

Muscettola, N., Nayak, P. P., Pell, B., & Williams, B. C. (1998). Remote agent: To boldly go where no ai system has gone before. *Artif. Intell.*, *103*(1-2), 5–47.

Musliner, D. J., Durfee, E. H., & Shin, K. G. (1993). Circa: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(6), 1561–1574.

Nilsson, M., Kvarnström, J., & Doherty, P. (2015). Revisiting Classical Dynamic Controllability: A Tighter Complexity Analysis. In Duval, B., van den Herik, J., Loiseau, S., & Filipe, J. (Eds.), *Proceedings of the Fifth International Conference on Agents and Artificial Intelligence (ICAART)*, pp. 243–261. Springer International Publishing.

Nilsson, M., Kvarnström, J., & Doherty, P. (2014). Incremental dynamic controllability in cubic worst-case time. In *Proc. 21st International Symposium on Temporal Representation and Reasoning (TIME)*, pp. 17–26.

Peintner, B., Venable, K. B., & Yorke-Smith, N. (2007). Strong controllability of disjunctive temporal problems with uncertainty. In Bessière, C. (Ed.), *Principles and Practice of*

Constraint Programming (CP) 2007, pp. 856–863, Berlin, Heidelberg. Springer Berlin Heidelberg.

Policella, N., Cesta, A., Oddi, A., & Smith, S. F. (2007). From precedence constraint posting to partial order schedules: A csp approach to robust scheduling. In *AI Communications, Special Issue on Constraint Programming for Planning and Scheduling*, pp. 163–180.

Shah, J. A., Stedl, J., Williams, B. C., & 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)*, pp. 296–303.

Stergiou, K., & Koubarakis, M. (2000). Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, *120*(1), 81 – 117.

Timmons, E., & Williams, B. (2015). Enumerating preferred solutions to conditional simple temporal networks quickly using bounding conflicts. In *AAAI Workshops*.

Tsamardinos, I. (2002). A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence*, pp. 97–108, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

Venable, K. B., Volpato, M., Peintner, B., & Yorke-Smith, N. (2010). Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *Proc. of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-20910) in ICAPS-2010*, pp. 50–59.

Venable, K. B., & Yorke-Smith, N. (2005). Disjunctive temporal problems with uncertainty. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pp. 1721–1722, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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

Weld, D. (1994). Introduction to least commitment planning. *AI Magazine*, *15*(4), 27–61.

Yu, P. (2016). BCDR Test Generator. https://github.com/yu-peng/BCDRTestGenerator.

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

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

Yu, P., Williams, B., Fang, C., Cui, J., & Haslum, P. (2017). Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *Journal of Artificial Intelligence Research*, *60*, 425–490.

Zavatteri, M. (2017). Conditional Simple Temporal Networks with Uncertainty and Decisions. In Schewe, S., Schneider, T., & Wijsen, J. (Eds.), *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, Vol. 90 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 23:1–23:17, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.