Weak and Dynamic Controllability of Temporal Problems with Disjunctions and Uncertainty

K. Brent Venable and Michele Volpato
University of Padova, Italy
kvenable@math.unipd.it
mvolpato@studenti.math.unipd.it

Bart Peintner
Artificial Intelligence Center
SRI International, USA
peintner@ai.sri.com

Neil Yorke-Smith
American University of Beirut, and
SRI International, USA
nysmith@aub.edu.lb

Abstract

The Temporal Constraint Satisfaction Problem with Uncertainty (TCSPU) and its disjunctive generalization, the Disjunctive Temporal Problem with Uncertainty (DTPU), are quantitative models for temporal reasoning that account simultaneously for disjunctive constraints and for events not under the control of the executing agent. Such a problem is Weakly Controllable if in each possible scenario the agent can find a decision for that scenario that satisfies all the constraints; further, a problem is Dynamically Controllable if the agent can build a consistent decision online, as the scenario is incrementally revealed. We first consider Weak Controllability. We present two sound and complete algorithms for checking Weak Controllability of DTPUs. The first algorithm needs to examine only a limited number of scenarios, but operates only on a restricted class of problems. The second algorithm is fully general, but is more expensive in terms of space. We then consider Dynamic Controllability. We present a complete algorithm for testing this property for TCSPUs. Complexity results are presented for all three algorithms.

Introduction

The Simple Temporal Problem (STP) (Dechter, Meiri, and Pearl 1991) is a celebrated success of constraint-based temporal reasoning. Since its introduction, various efforts have extended the expressiveness of the STP; in order to model more realistic scheduling and planning situations. Among these extensions is the Disjunctive Temporal Problem with Uncertainty (DTPU) (Venable and Yorke-Smith 2005; Peintner, Venable, and Yorke-Smith 2007). The DTPU permits non-convex and non-binary constraints, and also accounts for time-points not under the control of the executing agent; these are called uncontrollable time-points. Prior work defined three concepts of controllability of DTPUs—Strong, Weak, and Dynamic—and studied Strong Controllability. A problem is Strongly Controllable if the executing agent can find one decision that satisfies all the constraints under all scenarios of the uncontrollable time-points. A problem is Weakly Controllable if in each possible scenario the agent can find a decision for that scenario that satisfies all of the constraints. Finally, and perhaps the most useful notion in practice, a problem is Dynamically Controllable if the agent can build a consistent decision online, as the scenario is incrementally revealed.

This paper examines the two notions of controllability that have received less attention in the literature to date. We first consider Weak Controllability. We present two sound and complete algorithms for checking Weak Controllability of DTPUs. The first algorithm needs to examine only a limited number of scenarios, but operates only on a restricted class of problems. The second algorithm is fully general, but is more expensive in terms of space. We then consider Dynamic Controllability. We present a complete algorithm for testing such property for the Temporal Constraint Satisfaction Problem with Uncertainty (TCSPU), a restriction of the DTPU that permits some amount of disjunctivity. Complexity results are presented for all three algorithms. Proofs have been omitted due to space limitations.

Background

Simple Temporal Problems with Uncertainty

A Simple Temporal Problem with Uncertainty (STPU) (Vidal and Fargier 1998) is an STP in which uncertainty is allowed. Exogenous forces referred to as ‘Nature’ choose the value of uncontrollable time-point variables. Nature will ensure that the value of an uncontrollable variable (i.e., event) respects a single contingent constraint \( \lambda - X \in [a, b] \) where \( X \) is a controllable variable, \( \lambda \) is the uncontrollable variable, and \( a \geq 0 \). An STPU is defined as a tuple \( < V_c, V_u, C, C_u > \), where \( V_c \) and \( V_u \) are the sets of controllable and uncontrollable variables, respectively, \( C \) is a finite set of binary temporal constraints over \( V_c \cup V_u \), and \( C_u \subseteq C \) is the set of contingent constraints, one for each element of \( V_u \).

Given an STPU \( P \), a decision (or control sequence) \( d \) is an assignment to the controllable variables of \( P \), a situation (or realisation) \( w \) is a set of durations on contingent constraints (set of elements of contingent intervals). A schedule is a complete assignment to the variables of \( P \), i.e., it is a decision combined with a situation. We say that a schedule is viable if it is consistent with all the constraints. Sol\( (P) \) is the set of all viable schedules of \( P \). A projection \( P_w \) corresponding to situation \( w \) is the STP obtained replacing each contingent constraint with its duration in \( w \). Proj\( (P) \) is the
set of all projections of \( P \). A strategy \( S \) maps every projection \( P_w \) into a schedule including \( w \). A viable strategy \( S \): \( \text{Proj}(P) \rightarrow \text{Sol}(P) \) maps every projection \( P_w \) into a schedule \( \in \text{Sol}(P) \) including \( w \).

When uncontrollable variables are present the issue of consistency is replaced by that of controllability. There are three levels of controllability. In problems that exhibit Strong Controllability, there exists a time assignment to all executable variables that ensures all constraints will be satisfied whatever Nature’s realisation of the uncontrollable events. Weak Controllability, instead, ensures the existence of a solution for each complete situation likely to arise in the external world. Dynamic Controllability, finally, best suits dynamic applications, where the effective task durations are only observed as far as execution progresses.

An algorithm for checking Weak Controllability of STPUs (Vidal and Fargier 1998) uses the polynomial test for Strong Controllability, in addition to the concepts of Pseudo-controllability and Weak Controllability on bounds. An STPU is pseudo-controllable if in the minimal network of the associated STP (i.e., considering the STPU as an STP forgetting about the distinction between contingent and executable events) no interval on a contingent constraint is tightened. Instead, an STPU is weakly controllable on bounds if \( \forall w \in \{l_1, u_1\} \times \{l_2, u_2\} \times \cdots \times \{l_k, u_k\} \), where \( k \) is the number of contingent constraints and \( l_i, u_i \) are the lower and upper bounds respectively of the contingent constraint \( C_i \), there exists a strategy \( S \) such that \( S(P_w) \) is a solution of \( P_w \). In determining controllability, pseudo-controllability is used as a pre-processing step, since if an STPU is weakly controllable then it is also pseudo-controllable. The converse is not always true. For Weak Controllability on bounds, it is proved that an STPU is weakly controllable if and only if it is weakly controllable on bounds (Vidal and Fargier 1998). Hence, to test Weak Controllability it is sufficient to test Weak Controllability on bounds.

A dynamic execution strategy assigns a time to each executable variable that may depend on the outcomes of contingent constraints in the past, but not on those in the future. It is shown in Morris and Muscettola (2001) that determining Dynamic Controllability for STPUs is tractable, and an algorithm is presented that runs in polynomial time under the assumption that the size of constraints were bounded. The algorithm involves repeated checking of pseudo-controllability. In this paper we will describe an extension of the Morris et al. approach to TCSPUs.

Morris and Muscettola (2005) proposed a more uniform, but less intuitive, formulation of the reductions used in the Dynamic Controllability algorithm. An alternative representation is introduced for STPUs called labelled distance graph. With this representation is possible to provide a strongly polynomial algorithm that tests Dynamic Controllability for STPUs in \( \mathcal{O}(n^5) \) where \( n \) is the number of variables. A more efficient algorithm (\( \mathcal{O}(n^4) \)) is given in Morris (2006). It reasons in terms of the absence of a particular type of negative cycle. This is analogous to the consistency of ordinary STP in terms of the absence of negative cycles in the distance graph. In another work, Shah and Williams (2008) study incremental dynamic execution of STPUs.

### Disjunctive Temporal Problems with Uncertainty

The Disjunctive Temporal Problem (DTP) generalizes the STP by permitting disjunctions of time-point variables in the constraints (Tsamardinos and Pollack 2003). A DTP with Uncertainty (DTPU) (Venable and Yorke-Smith 2005) allows for both disjunctive constraints and contingent events. It permits constraints with two or more STPU constraints. We will sometimes call a disjunction the DTPU constraint and a disjunct the STPU constraint.

As defined in Peintner, Venable, and Yorke-Smith (2007), a DTPU is a tuple \( < V_c, V_u, C, C_u > \), where \( V_c \) and \( V_u \) are the sets of controllable and uncontrollable variables, respectively, \( C \) is a finite set of disjunctive temporal constraints over \( V_c \cup V_u \), and \( C_u \subseteq C \) is the set of binary contingent constraints, one for each element of \( V_u \).

An exact solution of a DTPU, \( s = s_c \cup s_u \) is a complete assignment to all the variables \( V = V_c \cup V_u \) that satisfies all constraints in \( C \). The controllable part of the solution \( s_c \) is the decision; the uncontrollable part \( s_u \) is the realisation.

In addition to constraints of type \( S \) (simple STP constraints), \( S_c \) (contingent STPU constraints) and \( S_e \) (executable STPU constraints), a DTPU features the types (Peintner, Venable, and Yorke-Smith 2007):

1. **DTP** \( (D) \): A disjunction of two or more STP constraints (e.g., “The image action must be ended 2 minutes before drilling begins (controllable) or the image action must be started after drilling begins (controllable)”).

2. **Executable DTPU** \( (D_c) \): A disjunction of two or more executable STPU disjuncts (e.g., “The image action must end 2 minutes before drilling ends (uncontrollable) or the image action must start at least 1 minute after drilling ends (uncontrollable)”).

3. **Mixed executable DTPU** \( (D_{mc}) \): A disjunction of STP and executable STPU constraints (e.g., “The image action must end before drilling starts (controllable) or the image action must start at least 1 minute after drilling ends (uncontrollable)”).

4. **Contingent DTPU** \( (D_c) \): A disjunction of two or more contingent STPU constraints. Nature chooses which disjunct will be satisfied (e.g., “Drilling can take 5–10 minutes or 15–20, depending on the equipment installed”).

If the set \( D_c \) of disjunctive contingent constraints of a DTPU \( P \) is empty, we call \( P \) a Simple-Natured DTPU.

Given a DTPU \( P \), a situation \( w \) is a set of durations on contingent constraints. A schedule is a complete assignment to the variables of \( P \). A schedule is viable if it is consistent with all the constraints. \( \text{Sol}(P) \) is the set of all viable schedules of \( P \). Thus \( \text{Sol}(P) \) is also the set of all exact solutions of the DTP obtained by treating all contingent constraints of \( P \) as executable constraints. A projection \( P_w \) corresponding to situation \( w \) is the DTP obtained replacing each contingent constraint with its duration in \( w \). \( \text{Proj}(P) \) is the set of all projections of \( P \). A strategy \( S \) maps every projection \( P_w \) into a schedule including \( w \). A viable strategy \( S \): \( \text{Proj}(P) \rightarrow \text{Sol}(P) \) maps every projection \( P_w \) into a schedule \( \in \text{Sol}(P) \) including \( w \).
In Peintner, Venable, and Yorke-Smith (2007), a sound and complete algorithm to determine whether Strong Controllability holds for a DTPU is given. In this paper we will address Weak Controllability and Dynamic Controllability on a restricted class: namely Temporal Constraint Satisfaction Problems with uncertainty.

A Temporal Constraint Satisfaction with Uncertainty (TCSPU) permits constraints with two or more STPU constraints, but limits the disjunctivity by stipulating the multiple constraints must have the same two variables in their scopes. Formally, a TCSPU is a tuple $<V_c, V_u, C, C_u>$, where $V_c$ and $V_u$ are respectively the sets of controllable and uncontrollable variables, $C$ is a finite set of binary disjunctive temporal constraints over $V_c \cup V_u$, and $C_u \subseteq C$ is the set of binary contingent constraints, one for each element of $V_u$. Since DTPUs are a generalization of TCSPUs, the definitions given above for DTPUs apply to TCSPUs as well.

As for STPUs, a dynamic execution strategy of TCSPUs assigns a time to each controllable variable that may depend on the outcomes of contingent constraints in the past, but not on those in the future. Let us recall the following standard notation: $[S(P_w)]_x$ is the time assigned to executable variable $x$ by schedule $S(P_w)$ and $[S(P_w)]_x$ (called the history of $x$ in $S(P_w)$) is the set of durations corresponding to contingent events which have occurred up to $[S(P_w)]_x$.

**Definition 1 (Dynamic Controllability of TCSPUs)** A TCSPU $P$ is dynamically controllable if and only if $\exists$ a strategy $S$ such that $\forall P_w \in \text{Proj}(P)$, $S(P_w)$ is a solution of $P_w$ (i.e., $S$ is viable) and $\forall P_1, P_2 \in \text{Proj}(P)$:

$$[S(P_1)]_x = [S(P_2)]_x \Rightarrow [S(P_1)]_x = [S(P_2)]_x$$


### Weak Controllability of a DTPU

In this section we propose two algorithms that solve the problem of testing Weak Controllability of DTPUs. The first algorithm exploits and extends the concept of Weak Controllability on bounds. The second algorithm employs two steps: the first step finds all possible schedules that satisfy the DTP obtained by treating all contingent constraints as executable constraint. The second step searches through the space of realisations (i.e., all possible instantiations of contingent constraints) to ascertain whether each realisation is contained in at least one of the satisfying schedules.

### Weak Controllability on Bounds for DTPUs

For the first algorithm we are not going to consider disjunctions of two or more contingent constraints. Indeed, uncertain non-convex durations are relatively rare.

We can define Weak Controllability of DTPUs similarly to the one for STPUs:

**Definition 2 (Weak Controllability of DTPUs)** A DTPU $P$ is weakly controllable if and only if $\exists$ a strategy $S$ such that $\forall P_w \in \text{Proj}(P)$, $S(P_w)$ is a solution of $P_w$.

If a DTPU $P$ is Simple-Natured then we can prove that $P$ is weakly controllable if it contains a weakly controllable component STPU.

**Theorem 1** Let $P$ be a Simple-Natured DTPU and $P_i$ a component STPU of $P$. If $P_i$ is weakly controllable then $P$ is weakly controllable as well.

The contrary does not hold, for example the DTPU of Figure 1 is weakly controllable, in fact considering $D = 0$, if $D' = 4$ then $I = 5$ satisfies all constraints, while if $D' = 5$ then $I = 7$ does and, finally if $D' = 6$ then $I = 7$ satisfies all constraints. But both the STPU components are not weakly controllable.

Vidal and Fargier (1998) prove that a STPU is weakly controllable if and only if it is weakly controllable on bounds. This does not hold for DTPUs.

For example, if we reconsider a DTPU as the one of Figure 1, except for the interval on constraint $I - D'$ that is now $[1,1]$, it is easy to see that it is weakly controllable on bounds but it is not weakly controllable.

Intuitively, some situations may be supported only by elements not included in the disjunctions. We thus need a to adapt the definition of Weak Controllability on bounds to the context of DTPUs.

**Definition 3 (WC on bounds w.r.t. a constraint)** A DTPU $P$ is weakly controllable on bounds with respect to one of its contingent constraint $C_i$ if $\forall w \in [l_1 \ldots u_1] \times \cdots \times [l_k \ldots u_k]$, where $k$ is the number of contingent constraints and $l_i, u_i$ are the lower and upper bound of the contingent constraint $C_i$, there exist a strategy $S$ such that $S(P_w)$ is a solution of $P_w$.

The first approach we propose consists of checking if $P$ is weakly controllable on bounds w.r.t. each constraints $C_i$ and, if it is so, then to verify that the DTPUs, obtained by replacing $C_i$ with its bounds, have a common weakly controllable component STPU. If there are weakly controllable component STPUs in common then we will build a weakly controllable component STPU of $P$. If there are not such weakly controllable component STPUs then we split the domain of $C_i$ into two smaller domains treating them like two different contingent constraints and act recursively.

![Figure 1: A weakly controllable DTPU.](attachment:figure1.png)

$D'$ is the only un-controllable event, white arrows represent executable constraints, the black arrow represents a contingent constraint.
Theorem 2 Let P be a Simple-Natured DTPU and C_i be a contingent constraint of P. Let P_l and P_u be the DTPU obtained tightening the constraint C_i to its lower bound l_i and upper bound u_i. If both P_l and P_u are weakly controllable and if they have a common weakly controllable component STPU then P is weakly controllable as well.

Exploiting Weak Controllability on bounds: DTPU-WC-Bounds

The result in the previous section suggests a way to test the Weak Controllability of a DTPU. We can check Weak Controllability of P_l and P_u recursively by tightening respectively to their lower and upper bound one by one each remaining contingent constraint. When there are no more constraints to tighten we have DTPs and we can check consistency for those. By verifying the intersection of their sets of solutions, without considering the last contingent constraint that cannot be the same in the solutions of P_l and P_u, we can decide Weak Controllability for the last DTPU.

If the intersection is empty, then we must split the domain of C_i into two smaller domains obtaining two different DTPUs. If both of them are weakly controllable then we can say that the initial DTPU is weakly controllable.

In what follows we will use the following notation: we will denote with S_C the set of contingent single disjunct constraints, with C_D the set of multi-disjunct constraints (D ∪ D_l ∪ D_u), and with C_S all others (S ∪ S_K). DTPU-WC-Bounds receives as input the three sets of constraints and returns the set of solution STPUs that are weakly controllable and whether the DTPU is weakly controllable or not; if there is no weakly controllable component STPU then it returns an empty set. The algorithm is composed by two mutually recursive functions, DTPU-WC and CheckIntersection.

DTPU-WC checks if there are contingent constraints to be processed. If there are none then it returns all the solutions of the current associated DTP. Otherwise it checks Weak Controllability on bounds for current DTPU by choosing a contingent constraint and replacing it with its lower (resp. upper) bound obtaining two different DTPUs, L and U. If both of them are weakly controllable then the initial DTPU is weakly controllable on bounds and the algorithm must verify with CheckIntersection if they have a common component STPUs.

Pseudocode is given in Algorithm 1. Line 1 is the exit condition for the recursion that is triggered when a full realisation is built, in this case all the contingent constraints have been instantiated. The DTPU becomes a classical DTP and Weak Controllability is equivalent to classical consistency. If the DTP is consistent then it is weakly controllable and (line 4) we return the set of solutions in form of STPs, otherwise (line 3) we return an empty set followed by the fact that it is not weakly controllable.

If there are contingent constraints then a contingent constraint C_i is removed from S_C and replaced with its lower bound, DTPU-WC checks Weak Controllability and does the same for the upper bound. If the presence of a non-weakly controllable DTPU is detected then the execution is stopped and DTPU-WC returns (∅, false) (lines 9 and 12).

Algorithm 1 DTPU-WC-Bounds

DTPU-WC(S_C, C_D, C_S)
1: if S_C = ∅ then
2: sol ← DTP-Solutions(C_D, C_S) (it’s a DTP: solve and return its solutions)
3: if sol = ∅ then return (∅, false) (not WC)
4: else return (sol, true) (this DTP is consistent, return the set of solutions)
5: else
6: choose and remove C_i from S_C (C_i is a simple contingent constraint)
7: C_S ← C_S \ {X_{i1} − X_{i2} ∈ [l_i, u_i]} \{C_i is in the form \}
8: if L = (∅, false) then return (∅, false) (found inconsistent DTP)
9: C_S ← C_S \ {X_{i1} − X_{i2} ∈ [u_i, u_i]}
10: U ← DTPU-WC(S_C, C_D, C_S)
11: if U = (∅, false) then return (∅, false) (found inconsistent DTP)
12: return CheckIntersection(S_C, C_D, C_S, l_i, u_i, L, U)

CheckIntersection(S_C, C_D, S_K, l_i, u_i, L, U)
1: if S_C = ∅ then return (L \ U)\{C_i is a simple contingent constraint\}
2: if u_i − l_i = 1 then return (L, true) (there is nothing to split)
3: choose and remove C_i such that l_i < k_i < u_i (there is no WC STPU component)
4: C_S ← C_S \ {X_{i1} − X_{i2} ∈ [k_i, k_i]}
5: return DTPU-WC(S_C, C_D, C_S)
6: if K = (∅, false) then return (∅, false) (a part is not weakly controllable)
7: inf ← CheckIntersection(S_C, C_D, C_S, l_i, u_i, L, K)
8: sup ← CheckIntersection(S_C, C_D, C_S, k_i, u_i, K, U)
9: if inf ≠ (∅, false) AND sup ≠ (∅, false) then
10: return (∅, true) (the two parts are WC and so the whole is WC)

Once DTPU-WC has checked Weak Controllability on bounds it has to call CheckIntersection to be sure that Weak Controllability is ensured (line 13).

Procedure CheckIntersection verifies that the intersections of weakly controllable component STPUs of the DTPUs obtained tightening the constraint C_i to its lower and upper bounds are not empty (line 1). In this case we have at least a component STPU for the initial DTPU that is weakly controllable and so we return the set of all weakly controllable components adding C_i to each of them.

Otherwise, if the intersection is empty, we have to split the interval of C_i into two different intervals (line 3) and check the two obtained DTPUs recursively (lines 7 and 8).

If C_i cannot be further split and the two DTPUs corresponding to the l_i branch and u_i branch are weakly controllable then if we join l_i and u_i into a single contingent constraint we obtain a weakly controllable DTPU. However, in addition to returning ‘true’, we also return an empty set of supporting STPUs (since there is not a common weakly controllable component STPU, line 2).

Example 1 Consider the DTPU of Figure 2, which has two uncontrollable variables: D' and A'. For the sake of compactness we call C_1 constraint A' = A ∈ [4, 6] and C_2 constraint D' − D ∈ [0, 5]. Moreover we call d_1 disjunct D − A ∈ [5, 5], d_2 disjunct D − A ∈ [7, 7], d_3 disjunct
\(B' - A' \in [1, 7]\) and \(d_4\) disjunct \(A' - B \in [2, 8]\).

Let us simulate the execution of DTPU-WC-Bounds on this problem. First DTPU-WC chooses a contingent constraint, for example \(C_1\), and replaces it with \(A' - A \in [4, 4]\) and then it replaces the other contingent constraint with \(D' - D \in [0, 0]\). The DTP \(P_1\) obtained in this way is consistent. Its STP solutions are two: \(s_1\) is the STP formed by all the initial variables, all the non-disjunctive executable constraint between them, the constraint \(A' - A \in [4, 4]\), the constraint \(D' - D \in [0, 0]\) and the disjuncts \(d_1\) and \(d_3\), while \(s_2\) differs from \(s_1\) only because \(d_3\) is replaced with \(d_4\). Now DTPU-WC replaces \(D' - D \in [0, 0]\) with \(D' - D \in [5, 5]\) obtaining another consistent DTP, \(P_2\), whose solutions are again \(s_1\) and \(s_2\). The execution between the two sets of solutions is non-empty, which means that the STPU obtained by replacing the contingent constraint \(D' - D\) with \(D' - D \in [0, 5]\) in \(s_1\) and the one obtained by replacing that same constraint in \(s_2\) are weakly controllable. Thus the DTPU \(P_{L_1}\) obtained by replacing the contingent constraint \(D' - D\) with \(D' - D \in [0, 5]\) in \(P_1\) (or in \(P_2\); we call it \(L_3\) because it is the one obtained by replacing the first contingent constraint with its lower bound) has two weakly controllable component STPUs and thus is weakly controllable.

The execution proceeds to consider \(A' - A \in [6, 6]\) and \(D' - D \in [0, 0]\). The DTP obtained, \(P_3\), has the following solutions: \(s_3\) that differs from \(s_4\) only because of the disjunctive constraints (\(s_3\) considers \(d_2\) and \(d_3\), and \(s_4\), that differs from \(s_3\) only because \(d_2\) is replaced with \(d_4\). Next replacing \(D' - D \in [5, 5]\), DTPU-WC obtains \(P_4\), which has solutions \(s_3\) and \(s_4\) again. Thus, as it was for the DTPU \(L_1\) of the first branch, the DTPU \(U_1\) (i.e., the one obtained by tightening the first contingent constraint to its upper bound) is weakly controllable.

The intersection between \(\{s_1, s_2\}\) and \(\{s_3, s_4\}\) is empty, so CheckIntersection needs to split \(C_1\) replacing it with \(A' - A \in [5, 5]\). Considering the bounds of the other contingent constraint we obtain these solutions: first \(\{s_3, s_4\}\) and then again \(\{s_3, s_4\}\).

Now the intersection between the sets of weakly control-
The algorithm employs two steps. The first step finds a value for each contingent constraint at each step of the recursion.

The first three lines of Phase II initialize the variables for the search. \( V \) holds the set of contingent constraints that do not yet have a value assigned to them (initially it contains all contingent constraints). \( A \) holds the assignments to the contingent constraints (initially empty). \( P \) contains the set of all solutions that support the assignments in \( A \). Given that \( A \) starts empty, \( P \) is initialized to contain all solutions to the underlying DTP.

At each step of the search, algorithm VerifyCoverage is called with the current values of \( V \), \( A \), and \( P \), and the supports sets \( h \). It returns a boolean value to indicate whether or not all realisations are supported.

Line 3 is the exit condition for the recursion that triggers when all contingent constraints have been assigned. This represents the leaf of the recursive tree. The function isValidAssignment checks whether the realisation is an actual solution to at least one of the solutions in \( P \). If this call returns true, then the realisation is covered. This check could be avoided if each assignment is propagated through the minimal networks at each step. However, delaying this step until the end reduces theoretical complexity.

Line 4 removes a contingent constraint \( r \) from \( V \) (possibly using a heuristic). The for loop iterates through the entire scope of \( v \). In all cases, \( v \) is either an STPU constraint or a binary DTPU contingent constraint \( (D_C) \). Thus, the scope of the constraint is equivalent to all possible durations of an uncontrollable process.

The first step in the iteration (line 6) adds the assignment of \( r \) to \( v \) into the set \( A \). The second step of the iteration (line 7) is central to the algorithm: it intersects the set \( P \), which contains all solutions that support all previous assignments, with \( h^r \), which is all solutions that support \( v = r \) alone. If the result, \( P' \), is not empty, then at least one solution supports the assignments in \( A \), and the search continues (line 9). Otherwise, false is returned which causes the algorithm to eventually exit, indicating the DTPU is not WC.

If the exit condition is reached, there is the option to store the realisation and associated solutions (line 2). This supports an executor that desires or needs to simply look up a schedule based on a known realisation. A compact encoding for the look up, such as a binary decision diagram, will reduce the unavoidable storage space required.

As for the previous algorithm, we now proceed to state completeness and soundness results.

**Theorem 5 (Completeness)** Let \( P \) be a DTPU. If \( P \) is weakly controllable then DTPU-WC-Search on \( P \) returns 'true'.

**Theorem 6 (Soundness)** Let \( P \) be a DTPU. If DTPU-WC-Search on \( P \) returns 'true' then \( P \) is weakly controllable.
The worst case time complexity of Phase I is the combination of solving the underlying DTP and storing each solution in the support sets. Solving a DTP requires $O(d^e * n^2)$. The worst case space complexity of this step is the space required to store the solutions: $O(d^e * e)$. This is disk space, not memory; thus space rather than time is the dominates.

The second aspect of Phase I is the storage of the support sets. This requires one operation for each minimal network value of each contingent constraint: $O(w * q)$ for each stored solution. This is the space complexity as well. Thus, the total time complexity of Phase I is $O(d^e * n^2 + d^e * w * q)$. For Phase II, the VerifyCoverage algorithm is called up to $w^q$ times, once for each element of the combined realisation space. The only complex element within the loop is line 7, which performs an intersection of two possibly large sets. Both sets can possibly hold the entire set $S$, so the time complexity is $O(d^e)$. The time complexity of isValidAssignment is $O(d^e * n^3)$, and so the total time complexity of Phase II is $O(w^q * (d^e * n^3))$.

For the whole algorithm, the total time complexity is therefore $O(d^e * n^2 + d^e * w * q + w^q * (d^e * n^3))$, and the total space complexity is $O(d^e * e * w * q)$.

**Comparing the Two Algorithms**

The two algorithms act on different types of problems. DTPU-WC-Bounds accepts only Simple-Natured DTPUs, while DTPU-WC-Search accepts all kinds of DTPUs and is thus more general. However, we argue that, Simple-Natured is a reasonable restriction since uncertain non-convex durations are relatively rare.

Regarding time complexity, the two algorithms have both exponential time complexity. However if a DTPU $P$ is weakly controllable then VerifyCoverage is called $w^q$ times, reaching the worst case, regardless of $P$. DTPU-WC-Bounds, on the contrary, may spend less time if $P$ has a weakly controllable component STPU becoming independent from the maximum domain size of contingent constraint. Thus we conjecture that DTPU-WC-Bounds behaves better in practice.

Regarding space complexity, DTPU-WC-Bounds has a better worst case complexity than DTPU-WC-Search and, moreover, since in DTPU-WC-Bounds is not necessary to keep each STP solution separated from the others, while it is for DTPU-WC-Search, a further reduction may be easily achieved.

Our future work is to implement the two algorithms and compare them empirically on a set of benchmark DTPUs.

**Dynamic Controllability of a TCSPU**

In this section we propose a complete algorithm for testing Dynamic Controllability of TCSPUs. We extend the method described in Morris and Muscettola (2001) for Dynamic Controllability of STPUs by adapting it to disjunctive constraints. The solving strategy we will pursue is that of making explicit the constraints that are implicit in the definition of Dynamic Controllability.

**Reductions**

We start by introducing the concept of *absolute bounds*:

- $w$, the maximum domain size of contingent constraint

The worst case time complexity of Phase I is the combination of solving the underlying DTP and storing each solution in the support sets. Solving a DTP requires $O(d^e * n^2)$. The worst case space complexity of this step is the space required to store the solutions: $O(d^e * e)$. This is disk space, not memory; thus space rather than time is the dominates.

The second aspect of Phase I is the storage of the support sets. This requires one operation for each minimal network value of each contingent constraint: $O(w * q)$ for each stored solution. This is the space complexity as well. Thus, the total time complexity of Phase I is $O(d^e * n^2 + d^e * w * q)$. For Phase II, the VerifyCoverage algorithm is called up to $w^q$ times, once for each element of the combined realisation space. The only complex element within the loop is line 7, which performs an intersection of two possibly large sets. Both sets can possibly hold the entire set $S$, so the time complexity is $O(d^e)$. The time complexity of isValidAssignment is $O(d^e * n^3)$, and so the total time complexity of Phase II is $O(w^q * (d^e * n^3))$.

For the whole algorithm, the total time complexity is therefore $O(d^e * n^2 + d^e * w * q + w^q * (d^e * n^3))$, and the total space complexity is $O(d^e * e * w * q)$.
due to the fact that if all possible occurrences of $C$ support the absolute bounds of the contingent constraint then they must support each value between them, even if not all of these values belong to the contingent constraint. Moreover, the absolute bounds must be supported in any case.

- If $z > 1$ (i.e., there is more than one disjunct on $B − C$). In this case we add $n$ new constraints with $z$ disjuncts each, in this way:

$$C − A \in [y_1 − v_1, x_1 − u_1] \lor \cdots \lor [y_1 − v_z, x_1 − u_z] \quad C − A \in [y_2 − v_1, x_2 − u_1] \lor \cdots \lor [y_2 − v_z, x_2 − u_z]$$

These $n$ constraints become a single constraint using the intersection operation. We will call this the induced constraint from $B$ on $C − A$.

In this way we ensure that each possible value for $B$ is consistent with all possible occurrences of $C$ because the first constraint we add ensures that each possible value for $C$ supports the entire first disjunct of $B − A$, the second constraint ensures that each possible value for $C$ supports the entire second disjunct of $B − A$ and so on.

It is possible to prove the following result.

**Lemma 2** If the precede reduction causes an inconsistency then the problem is not dynamically controllable.

**Unordered reduction.** If $u_1 < 0$ and $v_z ≥ 0$ then $C$ may or may not follow $B$. For a dynamic strategy, $C$ has to wait for $B$ to be executed or for an amount of time that ensure that any time at which $B$ is executed must be consistent with all possible occurrences of $C$.

As for the case of STPUs, for a viable dynamic strategy, if $B$ has not already occurred, $C$ cannot be executed at any time before a certain amount of time expires after $A$.

If $\exists i$ such that $u_i < 0 < v_i$, then we can consider $C$ as two disjuncts: $[u_i, 0] \lor [0, v_i]$. This allows us to divide the set of disjuncts of $B − C$ in negative disjuncts (i.e., the disjuncts that admit only values less than 0) and non-negative disjuncts (i.e., the disjuncts that admit only values grater than or equal to 0).

Given a subset $[x, y]$ of $B − A$, let us denote with $I_{\{x,y\}}$ the intersection between the induced constraint on $[x, y]$ and the current $C − A$ constraint. We say that a subset $[x, y]$ of $B − A$ is supported by the non-negative disjuncts of $B − C$ if $I_{\{x,y\}}$ is not empty.

To calculate the values that allow us to execute $C$ before $B$, assuring us that, whenever $B$ occurs, its value will be consistent with the one we choose for $C$, we need to find the subsets $[x_i, y_m]$ of $B − A$ (where $y_m$ is the absolute upper bound of $B − A$) that are supported by the non-negative disjuncts of $B − C$. To find them we use Algorithm 3.

We will now describe how Algorithm 3 works. An element $p$ of a disjunct of $I_{\{x,y\}}$ is good if $p > \text{prec}(x)$ where $\text{prec}(x)$ is the highest $x_j \in [x, y]$ belonging to a disjunct of $B − A$.

In line 6 we remove from the current $I_{\{x,y\}}$ the values that are not good. In fact, if $C$ is executed $p$ time units after $A$, $B$ has not yet occurred and $p$ is not a good value, then, if $B$ occurs $\text{prec}(x)$ time units after $A$ we are not sure that the problem will be consistent. Hence, if $C$ is executed $p$ time units after $A$, and $B$ has not yet occurred, if $p$ is a good value, then the problem is consistent, whenever $B$ occurs.

In Algorithm 3 we consider only subsets $[x, y]$ where $x$ is a lower bound of a disjuncts of $B − A$, in fact $[x_1, y_m]$ is supported and a disjunct of $I_{\{x,y\}}$ contains a good value, where $x_i$ is the lower bound of a disjunct of $B − A$, if and only if for each $x_i$ such that $x_i < x_i \leq y_i$, $[x_i, y_m]$ is supported and contains a good value. This is due to the fact that if $x_i < x_i \leq y_i$, then $\text{prec}(x) = x - 1$ and the upper bounds of the disjuncts of $I_{\{x,y\}}$ are decreased by one unit in $I_{\{x_i-1, y_m\}}$. Thus we need a jump from a lower bound $x_i$ to the upper bound $y_i-1$ if we want that $\text{prec}(x)$ and the upper bounds of $I_{\{x,y\}}$ decrease of different values.

In the end we have a set of disjuncts $S$ that contain the amounts of time that we can wait after $A$ before executing $C$ ensuring that, whenever $B$ occurs, the problem will be consistent (note that the absolute upper bound of $S$ is less than or equal to $y_m$).

Now that we have found these values, we have to check the subsets of $B − A$ that are supported by the negative disjuncts of $B − C$. We are interested in the larger $[x_1, y]$, where $y$ is an element of a disjuncts of $B − A$, that is supported by the negative disjuncts. This subset allow us to say that we can wait the execution of $B$ until $y$, i.e., if $B$ occurs before $y + 1$ time units after $A$ then we can find a value of $C$ such that the problem is consistent.

To find this subset we consider the initial triangle of constraints without the non-negative disjuncts of $B − C$, we compute the minimal network $P'$ of this new problem. The subset $[x_1, y]$ we are searching for is the largest subset containing $x_1$ of $P'$ that is also a subset of the initial problem.

Let $\tau$ be the set of disjuncts defined as follows:

$$\tau = S \cap \{y \geq 0\}$$

where $y$ is the absolute upper bound of the subset $[x_1, y]$ found while searching the largest subset supported by negative disjuncts of $B − A$ as described above.

Let $t$ be the absolute lower bound of $\tau$. We can distinguish two different cases.

1. If $y < y_m$ and the absolute upper bound of $\tau$ is less than or equal to $x_1$ (or if $\tau$ is empty), then $C$ must always occurs before $B$. Thus we add the constraint $C − A \in \tau$ (this
2. If \( y = y_m \) or the absolute upper bound of \( \tau \) is greater than \( x_1 \), then \( C \) has to wait either for \( B \) to occur, or it can be safely executed \( t \) time units after \( A \). Thus we add the wait \( ⟨ B, t, \tau \rangle \) to the constraint \( C - A \).

**Example 2 (Unordered reduction)** Let us consider this problem: \( B - A \in [1, 1] \lor [4, 6] \lor [9, 11] \), and \( B - C \in [-13, -5] \lor [2, 8] \lor [10, 14] \), and \( C - A \in [-13, -4] \lor [0, 24] \), where \( B \) is the uncontrollable event. The DTPU is pseudo-controllable. Since \(-13 < 0 \) and \( 14 \geq 0 \), this is an unordered case. To calculate \( S \), we first consider the subset \([9, 11]\) of \( B - A \). The induced constraint we first consider is \( C - A \in [-3, -1] \lor [3, 7] \), and \( I_{[9,11]} = \{3, 7\} \). It is supported; \( S = \emptyset \lor \{3, 7\} \). We continue testing \([4, 11]\). In this case the induced constraint is \( C - A \in [-2, -1] \lor I_{[4,11]} = \emptyset \). This is not supported, so we can stop here with \( S = \{3, 7\} \). \([1, 11]\) is not supported by the non-negative disjuncts of \( B - C \), in fact the induced constraints are: \( C - A \in [-7, -1] \lor [-13, -9] \), \( C - A \in [-2, 2] \lor [-8, -6] \), and \( C - A \in [3, 7] \lor [-3, -1] \). They become \( C - A \in [-2, -1] \lor I_{[1,11]} = \emptyset \).

The minimal network involving only the negative constraint of \( B - C \) is: \( B - A \in [1, 1] \lor [4, 6] \lor [9, 11] \), and \( B - C \in [-13, -5] \lor [2, 8] \lor [10, 14] \), and \( C - A \in [6, 24] \). Thus the negative disjunct entirely supports the contingent constraint, so \( y = y_m \) and we can add the wait \( ⟨ B, 7, [7, 7] ⟩ \). The associated graph of this problem is the one of Figure 4. Thus \( C \) can either wait for \( B \) (and then wait for the propagation of its value), or wait \( 7 \) time units after \( A \) and then execute.

The following property can be shown to hold.

**Lemma 3** If an unordered reduction is applicable then a strategy is viable and dynamic if and only if it satisfies the added wait \( ⟨ B, t, \tau ⟩ \).

**Unconditional reduction.** An unconditional reduction is applied after a new wait \( ⟨ B, t, \tau ⟩ \) is added to \( C - A \). It is applied only if \( t \leq x_1 \). In this case we add the constraint \( C - A \in \tau \lor [x_1, +\infty) \) to the problem, in this way we remove from \( C - A \) the values less than \( x_1 \) that \( C \) cannot take after \( A \). Differently from the case of STPUs, we cannot remove the wait, because in TCSPUs waits there is a disjunction that has to be satisfied.

**General reduction.** A general reduction is applied after a new wait \( ⟨ B, t, \tau ⟩ \) is added to \( C - A \). It is applied only if \( t > x_1 \). Controllable variable \( C \) must either wait for \( B \) or for \( t \) time units after \( A \). \( B \) can occur at \( x_1 \) the earliest, thus \( C \) can occur at \( x_1 \) the earliest. This induces a new constraint on \( C - A \): \( [x_1, +\infty) \).

**Regressions**

A regression is a propagation of a wait from one constraint to another. The wait is regressed from an executable constraint to another, but can be caused by another executable constraint or by a contingent constraint.

**Simple regression.** Simple regressions involve no contingent constraints. Consider the triangular TCSP of Figure 5.

\[ C \text{ has to wait either for } B \text{ to occur or for a value belonging to disjuncts of } \tau \text{ of time units to pass after } A. \text{ Moreover } C \text{ has to occur } [u_1, v_1] \lor \cdots \lor [u_2, v_2] \text{ time units after } D. \text{ It follows that } D \text{ must wait either for } B \text{ to occur or for an amount of time that is feasible with } C - A \text{ and } C - D. \text{ Hence, if } B \text{ has not occurred yet, then if we want to execute } D \text{ then } D - A \text{ must be within the bounds of the constraint on } D - A \text{ obtained by computing the minimal network on the triangle } C - A \in \tau \text{ and } C - D \in [u_1, v_1] \lor \cdots \lor [u_2, v_2], \gamma \text{ is the intersection between this constraint and the initial } D - A \text{ constraint, } g \text{ is the absolute lower bound of } \gamma. \]

**Lemma 4** If the simple regression causes an inconsistency then the problem is not dynamically controllable.

**Contingent regression.** Consider again the triangular TCSP of Figure 5, but now assume that \( C \) is a uncontrollable event and that \( C - D \) is a contingent event. Contingent regression is applied only if there is a contingent constraint involved, different from \( B \), and if \( t \geq 0 \).

\( C \text{ has to wait either for } B \text{ to occur or for a value belonging to disjuncts of } \tau \text{ of time units to pass after } A. \text{ Moreover } C \text{ has to occur } [u_1, v_1] \lor \cdots \lor [u_2, v_2] \text{ time units after } D. \)

Then, it follows that \( D \text{ must wait either for } B \text{ to occur or for an amount of time that is feasible with } C - A \text{ and}
with all the possible values of \( C - D \). Hence, if \( B \) has not occurred yet, to execute \( D \) then \( D - A \) must be within the bounds of the induced constraint on \( C - D \) of \( \tau \). \( \gamma \) is the intersection between this induced constraint and the initial \( D - A \) constraint; \( g \) is the absolute lower bound of \( \gamma \).

**Lemma 5** If the contingent regression causes an inconsistency then the problem is not dynamically controllable.

**Wait reduction.** A wait reduction is applied after a new wait \( (B_i, t_i, \tau_i) \) is added to a constraint \( C - A \) only if there are other waits on that constraint.

Suppose there are other waits \( (B_1, t_1, \tau_1) \cdots (B_k, t_k, \tau_k) \) on \( C - A \). Then \( C \) has to wait either for \( B_i \) to occur or for a value belonging to disjuncts of \( \tau_i \) for each \( i \).

Suppose \( \tau_1 \cap \cdots \cap \tau_k \cap \tau_0 \) is empty, then the projection where for each \( i \), \( B_i \) occurs after the absolute upper bound of \( \tau_i \), cannot be mapped in a feasible solution. In fact, there is no value for \( C \) that supports all the contingent constraints involved in the waits. Thus we have to replace each \( \tau_i \) with the intersection \( \tau_1 \cap \cdots \cap \tau_k \cap \tau_0 \).

**Dynamic Checking of Controllability**

Based on the reductions and regressions, we define a DC checking algorithm, TCSPU-DC, as an adaptation of the one in Morris and Muscettola (2001). We omit the pseudocode.

TCSPU-DC first computes the minimal network of the TCSPU \( P \) seen as a TCSP. If a contingent constraint is tightened then there is at least a value of its intervals that is not supported. Thus, it cannot be dynamically controllable. If none of \( P \)’s contingent constraints is tightened then we can proceed with the reductions. We first apply any possible pre-cede reduction and unordered reduction. These may lead to tighten some constraints and add some waits. Thus TCSPU-DC applies all possible regressions, and, after those, it applies the wait, unconditional and general reductions. Since constraints may be tightened, we compute again the minimal network and check again pseudo-controllability. TCSPU-DC continues this loop until no more constraints are tightened or until it finds an inconsistency. This process is complete, because all the reductions and regressions are justified by the definition of Dynamic Controllability (see Lemmata 2, 3, 4, and 5) or by the meaning of wait (unconditional, general and wait reductions). In fact, if it returns ‘false’ then the TCSPU in input is not dynamically controllable. We have not proved that it is sound.

The complexity of TCSPU-DC depends on: the number of variables \( n \), the maximum number of disjuncts per constraint \( d \), the number of contingent constraints \( q \), and the maximum domain size of contingent constraint \( w \). To find the minimal network of a TCSP has exponential time complexity.

For each of the \( q \times n \) triangles the reduction that takes more time is the unordered. Algorithm 3 takes \( O(w) \), computing the minimal network for the negative disjuncts of \( B - C \) takes \( O(d^3) \) (there are \( d^3 \) STPU component of which TCSPU-DC has to find the minimal network). TCSPU-DC repeats this process until inconsistency is found, thus a constraint has no more allowable values, or until the quiescence of the problem. Thus polynomial time. Thus the worst case time complexity is exponential due to the complexity of the step consisting of finding the minimal network.

**Conclusion**

Disjunctive Temporal Problems with Uncertainty allow expression of non-convex and non-binary temporal constraints, and also uncontrollable time-point variables that are not under the control of the executing agent.

This paper has presented two sound and complete algorithms for checking Weak Controllability of DTPUs. The first algorithm needs to examine only a limited number of scenarios, but operates only on a restricted class of problems. The second algorithm is fully general, but is more expensive in terms of space. We then presented a complete algorithm for testing Dynamic Controllability of TCSPUs.

The clear direction for future work is to implement the algorithms and compare them empirically on a set of benchmark temporal problems.

**Acknowledgements** The authors thank the COPLAS’10 reviewers for their suggestions.

**References**


