This is a pre-print version of the homonymous paper appearing in AAAI 2016.

Dynamic Controllability of Disjunctive Temporal Networks: Validation and Synthesis of Executable Strategies

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, 38123, Italy
cimatti@fbk.eu

Andrea Micheli
FBK and University of Trento
Trento, 38123, Italy
amicheli@fbk.eu

Marco Roveri
Fondazione Bruno Kessler
Trento, 38123, Italy
roveri@fbk.eu

Abstract

The Temporal Network with Uncertainty (TNU) modeling framework is used to represent temporal knowledge in presence of qualitative temporal uncertainty. Dynamic Controllability (DC) is the problem of deciding the existence of a strategy for scheduling the controllable time points of the network observing past happenings only.

In this paper, we address the DC problem for a very general class of TNU, namely Disjunctive Temporal Network with Uncertainty. We make the following contributions. First, we define strategies in the form of an executable language; second, we propose the first decision procedure to check whether a given strategy is a solution for the DC problem; third we present an efficient algorithm for strategy synthesis based on techniques derived from Timed Games and Satisfiability Modulo Theory. The experimental evaluation shows that the approach is superior to the state-of-the-art.

Introduction

Temporal Networks (TN) are a common formalism to represent and reason about temporal constraints over a set of time points (e.g. start/end of activities in a scheduling problem). Since the introduction of Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991), several extensions have been introduced (Tsamardinos and Pollack 2003). Vidal and Fargier (Vidal and Fargier 1999) introduced TN with uncertainty (TNU), where some of the time points are not under the control of the scheduler, but can only be observed. In this setting, the problem is to find assignments to the controllable time points so that the constraints can be solved for all the choices of the uncontrollable time points. Several variants are defined, e.g. strong controllability (SC) and weak controllability (WC). Here we focus on Dynamic Controllability (DC): a TNU is DC if there exists a strategy to schedule the controllable time points such that all the constraints are satisfied, and such that it can observe the uncontrollable time points once they happened. DC is arguably the most widely applicable problem in TNU, with applications ranging from planning and scheduling of space satellites (Frank et al. 2001) to robotics (Effinger et al. 2009).

DC is a very hard problem. The vast majority of the literature is subject to two limiting assumptions. First, it deals with DC in the restricted setting of Simple Temporal Networks with Uncertainty (STNU-DC). Second, the focus is on the decision problem (Morris 2006), and not on the explicit generation of an executable strategy. This requires the execution, at run-time, of complex scheduling algorithms (Hunsberger 2014). Unfortunately, this is unacceptable in many applications. For example, the deployment of a scheduling algorithm directly on-board of the controlled system may be subject, in a safety-critical setting, to computational or certification constraints.

In this paper we tackle the DC problem for TNUs lifting these restrictions. We consider the expressive class of Disjunctive Temporal Networks with Uncertainty (DTNU), where disjunctions are allowed in both the problem requirements (called free constraints) and in the assumptions on the environment behavior (called contingent links). The key idea is to approach the problem by focusing on the executable strategies from the start. We make the following contributions. First, we define a language to express easily executable, dynamic strategies that are sufficient for every DTNU-DC problem. Second, we present the first algorithm for validating a given strategy: the strategy is checked for being dynamic and for always yielding a valid schedule of the controllable time points. Third, we propose an algorithm to decide DTNU-DC. If the problem admits a solution, the algorithm has the unique property of synthesizing a correct-by-construction, dynamic executable strategy. The proposed techniques are derived from a fast Timed Game Automata (TGA) solving algorithm (Cassez et al. 2005), augmented with a pruning procedure based on Satisfiability Modulo Theory (SMT) (Barrett et al. 2009). We implemented several variants of the validation and synthesis algorithms, and we carried out an extensive experimental evaluation. The synthesis procedure exhibits practical applicability, solving problems of significant size, and demonstrating orders-of-magnitude improvements with respect to (Cimatti et al. 2014a), that is the only other approach to decide DTNU-DC.

Related Work. The DTNU formalism with the relative controllability problems was originally proposed in (Venable and Yorke-Smith 2005), it has been used in several works (Peintner, Venable, and Yorke-Smith 2007; Venable et al. 2010), but no general solution for DTNU-DC is presented. The only approach for solving DTNU-DC is in (Cimatti et al. 2014a). It generalizes the STNU-DC encoding to timed games of (Cimatti et al. 2014b) to DTNU and conditional networks. The approach allows for the generation of strategies in form of stateless controllers, that can be executed without on-line reasoning or constraint propagation. The focus is mainly theoretical, and the approach is completely unpractical, being unable to solve even moderate-size DTNU-DC problems. The DC problem has been widely studied in the STNU subclass (Morris, Muscettola, and Vidal 2001; Morris and Muscettola 2005; Morris 2006; Hunsberger 2009; 2010; 2013; 2014). All these works represent dynamic strategies implicitly, as networks managed by an on-line reasoning algorithm (Morris 2006; Hunsberger 2013; Morris 2014). Finally, we mention (Cimatti, Micheli, and Roveri 2015b; 2015a) that apply SMT techniques to two other poroblems in the context of DTNU, namely strong and weak controllability.

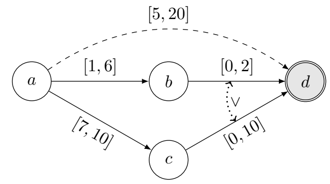
Problem definition

A DTNU models a situation in which a set of time points need to be scheduled by a solver, but some of them can only be observed and not executed directly.

Definition 1. *A DTNU is a tuple $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, where: $\mathcal{T}$ is a set of time points, partitioned into controllable ($\mathcal{T}_c$) and uncontrollable ($\mathcal{T}_u$); $\mathcal{C}$ is a set of* free constraints*: each constraint c_i is of the form, $\bigvee_{j=0}^{D_i} x_{i,j} - y_{i,j} \in [\ell_{i,j}, u_{i,j}]$, for some $x_{i,j}, y_{i,j} \in \mathcal{T}$ and $\ell_{i,j}, u_{i,j} \in \mathbb{R} \cup \{+\infty, -\infty\}$; and $\mathcal{L}$ is a set of* contingent links*: each $l_i \in \mathcal{L}$ is of the form, $\langle b_i, \mathcal{B}_i, e_i \rangle$, where $b_i \in \mathcal{T}_c$, $e_i \in \mathcal{T}_u$, and $\mathcal{B}_i$ is a finite set of pairs $\langle \ell_{i,j}, u_{i,j} \rangle$ such that $0 < \ell_{i,j} < u_{i,j} < \infty$, $j \in [1, E_i]$; and for any distinct pairs, $\langle \ell_{i,j}, u_{i,j} \rangle$ and $\langle \ell_{i,k}, u_{i,k} \rangle$ in $\mathcal{B}_i$, either $\ell_{i,j} > u_{i,k}$ or $u_{i,j} < \ell_{i,k}$.*

Intuitively, time points belonging to $\mathcal{T}_c$ are time decisions that can be controlled by the solver, while time points in $\mathcal{T}_u$ are under the control of the environment. A similar subdivision is imposed on the constraints: free constraints $\mathcal{C}$ are constraints that the solver is required to fulfill, while contingent links $\mathcal{L}$ are the assumptions that the environment is assumed to fulfill. Given a constraint $c \in \mathcal{C}$ we indicate with $TP(c)$ the set of time points occurring in c, and given any subset of the time points $P \subseteq \mathcal{T}$, we indicate with $\mathcal{C}(P)$ the set $\{c \mid TP(c) \subseteq P\}$ of the constraints that are defined on time points in P. Each uncontrollable time point e_i is constrained by exactly one contingent link to a controllable time point b_i called the activation time point of e_i that we indicate with $\alpha(e_i)$. A temporal network without uncertainty (TN) is a TNU with no contingent links and no uncontrollable time points. Consistency is the problem of checking if there exists an assignment to the time points that fulfills all the free constraints. A DTNU without uncertainty is called DTN (Tsamardinos and Pollack 2003).

A TNU is a game between the solver and the environment: the solver must schedule all the controllable time points fulfilling the free constraints, the environment schedules the uncontrollables fulfilling the contingent links. The network is said to be controllable if it is possible to schedule all the controllable time points for each value of the uncontrollable ones without violating the contingent constraints. Depending on the observability that the solver has, different degrees



$$\mathcal{T} \doteq \{a, b, c, d\}$$
$$\mathcal{C} \doteq \{b - a \in [1, 6],\; c - a \in [7, 10],\; d - b \in [0, 2] \vee d - c \in [0, 10]\}$$
$$\mathcal{L} \doteq \{\langle a, \{\langle 5, 20 \rangle\} \rangle, d\}$$

Figure 1: An example DTNU.

of controllability have been defined. In this paper we focus on the DC problem, that consists in deciding the existence of a strategy for scheduling all the controllable time points that respect the free constraints in every situation allowed by the contingent constraints. Differently from weak controllability, a dynamic strategy is not allowed to observe the future happenings of uncontrollable time points. Due to space constraints we do not report the full DC semantics that can be found in (Morris, Muscettola, and Vidal 2001; Hunsberger 2009). For the sake of this paper it suffices to say that a problem is DC if and only if it admits a dynamic strategy expressed as a map from partial schedules to Real-Time Execution Decisions (RTEDs). A partial schedule represents the current state of the game, that is the set of time points that have executed so far and their timing. Each RTED has one of two forms: `Wait` or $\langle t, \chi \rangle$. A `Wait` decision means "wait until some uncontrollable time point happens"; a (t, χ) decision stands for "if nothing happens before time t, schedule all the time points in χ at time t. A strategy is valid if, for every possible execution of the uncontrollable time points, all the controllable time points get scheduled in such a way that all the free constraints are satisfied.

Two different DC semantics exist in the literature, depending on whether the solver is allowed to react to an uncontrollable happening by immediately scheduling a controllable time point (Morris 2006) or a positive amount of time is required to pass (Hunsberger 2009). In this paper we adopt the first one, but the following results should be adaptable to the other case as well. Figure 1 shows an example DTNU. The problem has a single uncontrollable time point d that uncontrollably happens between 5 and 20 time units after a happens. A dynamic strategy σ_{ex} for the problem reads as follows. First, schedule a at time 0, then wait 5 seconds. If d interrupts the waiting, immediately schedule b then wait until 7 and then schedule c, otherwise immediately schedule b (at time 5) and wait until time 10. If d interrupts the waiting, wait until 7 and then schedule c, otherwise immediately schedule c and then wait for d to happen.

Strategy Representation and Validation

Existing approaches for the STNU-DC problem, produce strategies as constraint networks that need to be scheduled at run-time by an executor. These networks encode a possibly infinite number of RTEDs. Following the approach in (Cimatti et al. 2014b), it is possible to produce a strategy that is executable but unstructured: given a state of the execution (set of time points scheduled and a condition on their timing) the strategy associates an action to execute. We propose a language to express readily executable, closed-form strategies that closely resembles the structure of a program.

Clocks and Time Regions. We use clocks and time regions (Maler, Pnueli, and Sifakis 1995) as a basic data structure for representing sets of partial schedules. Given a TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, let $\overline{T} \doteq \{\overline{x} \mid x \in \mathcal{T}\}$ be a set of non-negative real-valued clock variables, one for each time point in $\mathcal{T}$. We define the time regions of $\overline{T}$ as the set $\Phi(\overline{T})$ of all possible formulae expressed by the following grammar, where $\bowtie \in \{<, \leq, >, \geq, =\}$.

$$\phi ::= \top \mid \bot \mid \overline{x} \bowtie k \mid \overline{x} - \overline{y} \bowtie k \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi$$

Intuitively, a time region represents a set of assignments to the clock variables in $\overline{T}$. We assume the usual intersection ($\phi \wedge \psi$), union ($\phi \vee \psi$) and complement ($\neg\phi$) operations are defined; in addition, we define the following operations.

$$\phi\nearrow \doteq \exists \delta \leq 0. \phi[x \to (x + \delta) \mid x \in \overline{T}]$$
$$\phi\swarrow \doteq \exists \delta \geq 0. \phi[x \to (x + \delta) \mid x \in \overline{T}]$$
$$\rho(\phi, \overline{x}) \doteq (\exists \overline{x}. \phi) \wedge \overline{x} = 0$$

where $\phi[x \to y]$ indicates the substitution of term x with term y in ϕ. Given a time region ϕ, $\phi\nearrow$ is the result of letting time pass indefinitely, while $\phi\swarrow$ is the time region from which it is possible to reach ϕ by letting time pass. The $\rho(\phi, \overline{x})$ operation unconditionally assigns the $\overline{x}$ clock to value 0 (clock reset). These are common operations in Timed Automata analysis: each operator yields a time region in $\Phi(\overline{T})$. Efficient and canonical implementations of time regions exists (Bengtsson 2002). Given a time region ϕ, we define the set $FV(\phi)$ of its free variables as the set of the clocks that occur in the definition of the region. We use a clock for each time point and encode the constraints in such a way that the clock measures the time passed since the time point was scheduled. For example, a time region $x - y \leq 2 \wedge x = 3$ indicates that 3 time units ago (at time $t - 3$), x was scheduled and y has been scheduled at least 1 time unit ago.

Strategy Language. We define the following language to compactly express strategies in a readily executable form.

Definition 2. *A strategy σ is either $\bullet$, $w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)$ where ψ is a time region, each $e_i \in \mathcal{T}_u$ and each σ_x is a strategy or $s(b); \sigma'$ where $b \in \mathcal{T}_c$ and σ' is a strategy.*

We have a single wait operator that waits for a condition ψ to become true. The wait can be interrupted by the observation of an uncontrollable time point or when the waited condition becomes true (represented by the $\dashv$ symbol). For each possible outcome, the strategy prescribes a behavior, expressed as a sub-strategy. In addition to wait statements, we also have the $s(b); \sigma'$ construct that prescribes to immediately schedule the b time point (we assume no time elapses), and then proceed with the strategy σ'. The $\bullet$ operator is a terminator signaling that the strategy is completed. For example, the strategy σ_{ex} for the DTNU in Figure 1, expressed in this language is as follows:

$$\begin{aligned}\sigma_{ex} \doteq s(a); w(a = 5, d &: s(b); w(a = 7, \dashv : s(c); \bullet),\\ &\dashv : s(b); w(a = 10, d : w(a \geq 7, \dashv : s(c); \bullet),\\ &\qquad \dashv : s(c); w(\bot, d : \bullet))).\end{aligned}$$

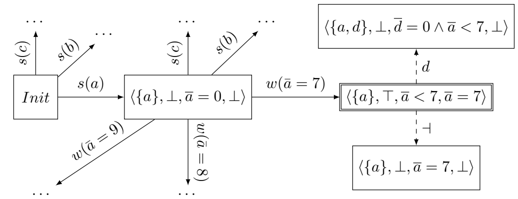


Figure 2: A portion of $\mathbb{S}$ for Figure 1 DTNU.

Theorem 1. *A DTNU is DC if and only if it admits a solution strategy expressible as per Definition 2.*

Proof. (Sketch) Following the approach in (Cimatti et al. 2014a), we know that a memory-less TGA strategy exists for every DC problem. Our strategy can be seen as a representation of the computation tree of that TGA strategy. $\square$

This syntax for strategies is similar to a loop-free program. In practice, one needs to represent the program allowing common strategies in different branches to be shared to avoid combinatorial explosion of the strategy size. For the sake of this paper we keep the simpler tree representation. The structure of the program explicitly represent the different branches that the strategy may take: each computation path from the beginning of the strategy to $\bullet$ is a way of scheduling the time points in a specific order.

A strategy σ needs two characteristics for being a solution to the DTNU-DC problem: dynamicity and validity. σ is dynamic if it never observes future happenings and is valid if it always ends in a state where all the controllable time points are scheduled and all the free constraints are satisfied, regardless of the uncontrollable observations. Using our strategy syntax, we can syntactically check if a strategy is dynamic: it suffices to check, for each branch of the strategy, that each wait condition ψ is defined on time points that have been already started or observed. Formally, this can be done by recursively checking the free variables of each ψ:

$$\begin{aligned}&dyn(P, \bullet) \doteq \top;\\ &dyn(P, s(b); \sigma') \doteq dyn(P \cup \{b\}, \sigma');\\ &dyn(P, w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)) \doteq\\ &\qquad FV(\psi) \subseteq P \wedge dyn(P, \sigma_\dashv) \wedge \textstyle\bigwedge_{i=1}^{n} dyn(P \cup \{e_i\}, \sigma_i).\end{aligned}$$

Proposition 1. *A strategy σ is dynamic if and only if $dyn(\emptyset, \sigma) = \top$.*

Validation. We now focus on the problem of validating a given strategy. We first present a search space that encodes every possible strategy for a given TNU. The search space $\mathbb{S}$ is an and-or search space where the outcome of a wait instruction is an and-node (the result of a wait is not controllable by the solver), while all the other elements of the strategy language are encoded as or-nodes (they are controllable decisions). The search space is a directed graph $\mathbb{S} \doteq \langle V, E \rangle$, where E is a set of labeled edges and each node in V is a tuple $\langle P, w, \phi, \psi \rangle$ where $P \in 2^{\mathcal{T}}$ is a subset of the time points representing the time points that already happened in the past, w is a Boolean flag marking the state as a waiting state, while both ϕ and ψ are time regions. ϕ represents the set of temporal configuration in which the state can be; ψ is set only in and-nodes to record the condition that has been waited for. The graph is rooted in the node $Init \doteq \langle \emptyset, \bot, \top, \bot \rangle$.

Given a time region ϕ we define the time region that specifies the waiting time for a condition ψ as a time region $\omega(\phi, \psi) \doteq \phi\nearrow \wedge \neg(\psi\nearrow)$. This is the set of time assignments that the system can reach from any assignment compatible with ϕ while waiting for condition ψ. Moreover, for each uncontrollable time point e, we define the $uc(e)$ time region as $\bigvee_{\langle \ell, u \rangle \in \mathcal{B}} \overline{\alpha(e)} \geq \ell \wedge \overline{\alpha(e)} \leq u$ where $\langle \alpha(e), B, e \rangle \in \mathcal{L}$. Intuitively, $uc(e)$ is the portion of time in which the uncontrollable time point e might be observed, according to the contingent links. In fact, this is a transliteration of a contingent link into a time region.

The transitions in E are defined as follows (we indicate transitions with arrows between two states):

- $\langle P, \bot, \phi, \bot \rangle \xrightarrow{s(b)} \langle P \cup \{b\}, \bot, \rho(\phi, \overline{b}), \bot \rangle$ with $b \in \mathcal{T}_c \setminus P$;
- $\langle P, \bot, \phi, \bot \rangle \xrightarrow{w(\psi)} \langle P, \top, \omega(\phi, \psi), \psi \rangle$;
- $\langle P, \top, \phi, \psi \rangle \xrightarrow{e} \langle P \cup \{e\}, \bot, \rho(\phi, \overline{e}) \wedge uc(\overline{e}), \bot \rangle$ where $e \in \mathcal{T}_u \setminus P$ and $\alpha(e) \in P$;
- $\langle P, \top, \phi, \psi \rangle \xrightarrow{\dashv} \langle P, \bot, \psi, \bot \rangle$.

Nodes having $w = \bot$ are considered or-nodes, while the others are and-nodes. Intuitively, the first rule allows to immediately start a controllable time point if we are not in a state resulting from a wait. The second rule allows the solver to wait for a specific condition ψ, the resulting state is an and-node because the outcome of the wait can be either a timeout ($\dashv$) or an uncontrollable time point. The last two rules explicitly distinguish these outcomes. We remark that, when a time point x is scheduled or observed, the corresponding clock $\overline{x}$ is reset and the set P keeps record of the time points that have been scheduled or observed.

This search space directly mimics the structure of our strategies, and as such it is infinite due to the infinite number of conditions that can be waited for. Nonetheless, this space is conceptually clean and very useful to approach the validation problem. Figure 2 depicts the first portion of the search space $\mathbb{S}$ for the running example problem in Figure 1.

The procedure for validating a strategy is shown in Algorithm 1: the algorithm navigates the search space $\mathbb{S}$ applying the strategy prescriptions (thus finitizing the search) and checking that each branch invariably yields to a state where all the free constraints are satisfied and all the time points are scheduled. The algorithm starts from $Init$ and recursively validate each branch of the strategy. The region Ψ, used in line 2, is defined as the region where all the free constraints are satisfied, as follows:

$$\Psi \doteq \bigwedge_{c_i \in \mathcal{C}} \bigvee_{j \in [0, D_i]} \ell_{i,j} \leq \overline{y}_{i,j} - \overline{x}_{i,j} \leq u_{i,j}.$$

A time region implies Ψ if it satisfies all the free constraints. Clocks measure the time passed since the corresponding time point has been executed, therefore a constraint $x - y \in [\ell, u]$ is represented by the region $\ell \leq \overline{y} - \overline{x} \leq u$. The procedure executes a strategy starting from $Init$; then it recursively explores the search space enforcing the controllable decisions of the strategy σ in or-nodes (lines 3-8) and branching to explore all possible uncontrollable outcomes in and-nodes (lines 9-15). It is easy to see that the algorithm takes a number of steps that is linear in the size of the strategy because at each step the strategy gets shortened (two steps are needed to shorten a wait).

Algorithm 1 Dynamic Strategy Validation Procedure

1: **procedure** VALIDATE(σ, $\langle P, w, \phi, \psi \rangle$)
2:   **if** $P = \mathcal{T}$ **then return** ($\sigma = \bullet$ **and** ISVALID($\phi \rightarrow \Psi$))
3:   **if** $w = \bot$ **then**
4:     **if** $\sigma = s(b); \sigma'$ **then**
5:       **return** VALIDATE(σ', $\langle P \cup \{b\}, \bot, \rho(\phi, b), \bot \rangle$)
6:     **else if** $\sigma = w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)$ **then**
7:       **return** VALIDATE(σ, $\langle P, \top, \omega(\phi, \psi), \psi \rangle$)
8:     **else return** $\bot$ ▷ $\sigma = \bullet$, but not all time points scheduled
9:   **else**   ▷ $\sigma \doteq w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)$
10:     **for all** $\langle P, w, \phi, \psi \rangle \xrightarrow{\hat{e}} \langle P \cup \{\hat{e}\}, \bot, \phi \wedge uc(\hat{e}), \bot \rangle$ **do**
11:       **if** $\nexists i. \hat{e} = e_i$ **then return** $\bot$ ▷ $\hat{e}$ it is not handled by σ
12:       $c \coloneqq$ VALIDATE(σ_i, $\langle P \cup \{\hat{e}\}, \bot, \rho(\phi, e) \wedge uc(e), \bot \rangle$)
13:       **if** $c = \bot$ **then return** $\bot$
14:     **if** ISUNSATISFIABLE(ψ) **then return** $\top$ ▷ The strategy waited for $\bot$
15:     **return** VALIDATE($\sigma_\dashv$, $\langle P, \bot, \psi, \bot \rangle$)

Proposition 2. *The validation procedure in Algorithm 1 is sound and complete.*

Synthesizing Dynamic Strategies

We now discuss synthesis of dynamic strategies that are valid by construction. In theory, one could do a classical and-or search in the space $\mathbb{S}$: if the search terminates, the trace of the search itself is a valid strategy. However, the infinity of the space makes this choice impractical. The problem with the search space $\mathbb{S}$ is the presence of explicit waits: while the length of each path is finite, the arity of each or-node is infinite. We take inspiration from (Cimatti et al. 2014a) where the authors use TGAs to encode the DC-DTNU problem: the idea of the approach is to exploit the expressiveness of the TGA framework by casting the DC problem to a TGA reachability problem.

TGA Reachability. A Timed Game Automaton (TGA) (Maler, Pnueli, and Sifakis 1995) augments a classical finite automaton to include real-valued clocks: transitions may include temporal constraints, called guards. Each transition may also include clock resets that cause specified clocks to take value 0 after the transition. Each location may include an invariant and the set of transitions is partitioned into controllable and uncontrollable transitions.

Definition 3. *A TGA is a tuple, $A = (L, l_0, Act, \overline{X}, E, I)$, where: L is a finite set of locations; $l_0 \in L$ is the initial location; Act is a set of actions partitioned into controllable Act_c and uncontrollable Act_u; $\overline{X}$ is a finite set of clocks; $E \subseteq L \times \Phi(\overline{X}) \times Act \times 2^{\overline{X}} \times L$ is a finite set of transitions; and $I : L \rightarrow \Phi(\overline{X})$ associates an invariant to each location.*

A TGA can be used to model a two-player game between an agent and the environment, where the agent controls the controllable transitions, and the environment controls the uncontrollable transitions. Invariants are used to define constraints that must be true while the system is in a location. A reachability game is defined by a TGA and a subset of its locations called the goal (or winning) state (Cassez et al. 2005). The state-of-the-art approach for solving TGA Reachability Games is (Cassez et al. 2005). Essentially, the algorithm optimistically searches a path from the initial state to a goal state and then back-propagates under-approximations of the winning states using the timed-controllable-predecessor operator. The approach terminates if the initial state is declared as winning or if the whole search space has been explored.

Synthesis via TGA. We retain the basic idea behind the search space $\mathbb{S}$ of explicitly representing the possible orderings in which the time points may happen in time, but we exploit the TGA semantics to implicitly represent the controllable waits. In TGA time can elapse inside locations until a transition (controllable or uncontrollable) is taken. This is conceptually analogous to wait for a condition allowing to take a controllable transition, provided that the wait may be interrupted by an uncontrollable transition. We model uncontrollable time points as uncontrollable transitions with appropriate guards, and controllable time points as controllable transitions. Unfortunately, this yields a TGA whose size is exponential in the number of time points: we need a state for each subset of $\mathcal{T}$. We represent this exponential TGA implicitly, by constructing transitions and constraints on-demand while exploring the symbolic state space using an algorithm derived from (Cassez et al. 2005). The implicit expansion allows us to construct only the transitions that are needed for the strategy construction process. The implicit TGA is defined as $\langle 2^{\mathcal{T}}, \emptyset, \mathcal{T}, \overline{T}, E, \emptyset \rangle$. Each subset of the time points is a location, the initial state is the empty set where no time points have been scheduled. We have an action label for each time point: a controllable time point yields a controllable transition, an uncontrollable corresponds to an uncontrollable transition. The set of clock variables is $\overline{T}$ as in the space $\mathbb{S}$ and the transition relation E is defined as:

- $P \xrightarrow{b,\, fc(P,b)} P \cup \{b\}$ with $b \in \mathcal{T}_c$, if $b \notin P$;
- $P \overset{e,\, uc(e)}{\dashrightarrow} P \cup \{e\}$ with $e \in \mathcal{T}_u$, if $\alpha(e) \in P$ and $b \notin P$.

where each transition implicitly resets the time point corresponding to its label and the $fc(P, b)$ and $uc(e)$ functions define the guards of the transitions.

$$fc(P,b) \doteq \begin{cases} \top & \text{iff } \mathcal{C}(P) = \emptyset \\ \bigwedge_{c_i \in \mathcal{C}(P)} \bigvee_{j \in [0, D_i]} \overline{y}_{i,j} - \overline{x}_{i,j} \in [\ell_{i,j}, u_{i,j}] \end{cases}$$

Algorithm 2 reports the pseudo-code of the approach derived from (Cassez et al. 2005). We write $s_1 \overset{x}{\Rightarrow} s_2$ for either $s_1 \xrightarrow{x} s_2$ or $s_1 \overset{x}{\dashrightarrow} s_2$, where no distinction is needed. Each time an expansion is required (lines 3, 10 and, implicitly, 12) the relative portion of the search space is created. The algorithm works as follows. win is a map that records the winning portion of visited states. The $wait$ set contains the transitions in the symbolic space that need to be analyzed and is initialized with the outgoing transitions of the initial state (line 3). The PUSH and PPUSH functions insert a set of elements in the set (the difference between the two is explained later), while the POP function picks and removes an element from the set. The dep map is used to record the set of explored edges that lead to a state. W keep track of the visited states and perform two different computations depending on whether a new state is encountered or a re-visit happens (line 6). In the first case, the algorithm explores the state space forward, in the other it back-propagates winning states. In the forward expansion, the distinction between controllable and uncontrollable transitions is disregarded until a goal state is reached. When a goal state is found, the winning states are recorded in the win map and the transition leading to the goal is re-added to the $wait$ set to trigger the back-propagation of the winning states. The back-propagation computes the set of winning states and updates the win table if needed. Then, all the explored edges leading to s_1 are set to be re-explored because their winning states may have changed due to this update of the winnings of s_1. Line 16 decides the controllability of the problem: if the initial state has no winning subset, it means that all the search space has been explored and no strategy exists.

Algorithm 2 Synthesis algorithm

1: **procedure** SYNTHESIZE( )
2: $\quad wait := \emptyset; dep := \emptyset; win := \emptyset$
3: $\quad$ PPUSH($wait$, $\{\langle \emptyset, \top \rangle \overset{x}{\Rightarrow} \langle P, \phi\nearrow \rangle \mid \langle \emptyset, \top \rangle \overset{x}{\Rightarrow} \langle P, \phi \rangle\}$)
4: $\quad$ **while** $wait \neq \emptyset \wedge win[\langle \emptyset, \top \rangle] = \bot$ **do**
5: $\quad\quad (s_1 \overset{x}{\Rightarrow} s_2) :=$ POP($wait$) $\quad \triangleright \langle P_1, \phi_1 \rangle = s_1, \langle P_2, \phi_2 \rangle = s_2$
6: $\quad\quad$ **if not** ALREADYVISITED(s_2) **then**
7: $\quad\quad\quad dep[s_2] := \{s_1 \overset{x}{\Rightarrow} s_2\}$
8: $\quad\quad\quad$ **if** $P_2 = \mathcal{T}$ **and** ISSATISFIABLE($\phi_2 \wedge \Psi$) **then**
9: $\quad\quad\quad\quad win[s_2] := \phi_2 \wedge \Psi;$ PUSH($wait$, $\{s_1 \overset{x}{\Rightarrow} s_2\}$)
10: $\quad\quad\quad$ **else** PPUSH($wait$, $\{s_2 \overset{x}{\Rightarrow} \langle P_3, \phi_3\nearrow \rangle \mid s_2 \overset{x}{\Rightarrow} \langle P_3, \phi_3 \rangle\}$)
11: $\quad\quad$ **else**
12: $\quad\quad\quad win^* :=$ BACKPROPAGATEWINNING(s_1)
13: $\quad\quad\quad$ **if** $win^* \not\subset win[s_1]$ **then**
14: $\quad\quad\quad\quad win[s_1] := win[s_1] \vee win^*;$ PUSH($wait$, $dep[s_1]$)
15: $\quad\quad\quad dep[s_2] := \{s_1 \xrightarrow{x} s_2\}$
16: $\quad$ **if** $win[\langle \emptyset, \top \rangle] = \emptyset$ **then return** $\bot$ **else return** MKSTRATEGY(dep, win)

Theorem 2 (Correctness)**.** *With no pruning, the algorithm terminates and returns $\bot$ if and only if the TNU is not DC.*

Proof. (Sketch) The search space explored by algorithm is a restriction of the one for the TGA in (Cimatti et al. 2014a) that captures the DC problem semantics. We remove states whose winning set is empty as they cannot satisfy the free constraints: this is a sound and complete restriction. □

MKSTRATEGY builds a strategy as per Definition 2 using a forward search guided by the dep and win maps: wait conditions are extracted by the projection of the winning states.

Ordered and Unordered states. So far, we considered the discrete component (the TGA location) of each state as a set P. This is conceptually clean but it might be a drawback. For example, suppose the search decides to schedule the sequence A then B then C and then, by backpropagation of the winning, this is insufficient to find a strategy. The search can now explore the path A then C then B computing the relative winning states. If we disregard the order of states, it is possible (depending on temporal constraints) that a state

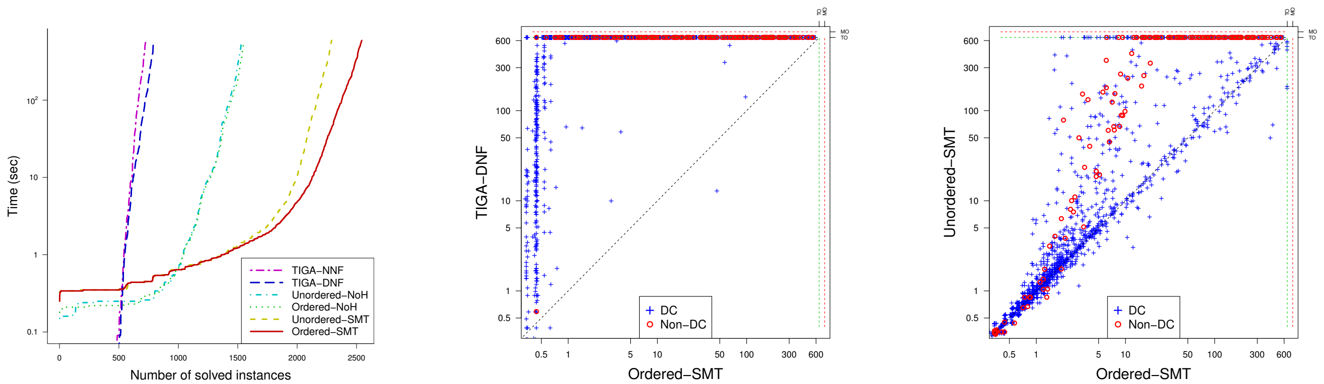


Figure 3: Results for the experimental evaluation. All the plots are in logarithmic scale and measures are in seconds.

$\langle\{A, B, C\}, \phi\rangle$ is explored in both cases, causing its winning region to be updated twice (resulting in the disjunction of the winning of the two visits). This is not a correctness issue, but a lot of disjunctions are introduced, and disjunctions heavily impact the performance of time region manipulations.

A solution that produces a variant of the algorithm is to consider P as a sorted set, hence distinguishing between different orderings of time points. This increases the theoretical search space size (that become more than factorial in the number of time points: $\sum_{i=1}^{|\mathcal{T}|} i!$) but possibly simplifies the time region management. The ordered exploration distinguishes states with the same set of scheduled time points but different scheduling order. In the example, we would have two final states because $\langle\langle A, B, C\rangle, \phi\rangle$ is different from $\langle\langle A, C, B\rangle, \phi\rangle$. Hence, the winnings of $\langle\langle A, B, C\rangle, \phi\rangle$ are not updated. This moves some of the complexity from temporal disjunctive reasoning to the discrete search itself.

Pruning Unfeasible States. The use of an explicit representation (ordered or unordered) of the past time points allows for an important optimization: the pruning of unfeasible paths. When the algorithm adds a new, unexplored transition to the $wait$ set (lines 3 and 10), the ordering of the time points resulting from the transition may be inconsistent with the free constraints. For example in the DTNU of Figure 1, any path starting with time point b is never going to satisfy the free constraints, hence the expansion $\langle\emptyset, \top\rangle \xrightarrow{b} \langle\{b\}, \overline{b} = 0\rangle$ can be discarded. The PPUSH function (short for "pruning push") is demanded to insert a set of elements in the $wait$ set, but it may discard unfeasible transitions, working as a filter. This pruning greatly reduces the search space especially in TNUs with many constraints.

We identified a pruning method for PPUSH, using a consistency check. For each transition $s_1 \overset{x}{\Rightarrow} \langle P_2, \phi_2\rangle$, we check if, disregarding uncertainty, it is possible for the time points in P_2 to be executed before all the time points in $\mathcal{T} \setminus P_2$. For this reason we convert the input DTNU in a DTN (without uncertainty) by considering each uncontrollable time point as controllable, and each contingent link as a free constraint. We then add the following to the DTN constraints:

$$\mathcal{C}_{add} \doteq \{y - x \in [0, \infty] \mid x \in P, y \in \mathcal{T} \setminus P\}.$$

If the resulting problem is consistent, then the transition is added to the $wait$ set, otherwise it is discarded. Note that in lines 14 and 23 we do not use PPUSH because we are not exploring new transitions but re-visiting (already-checked) transitions. This pruning only removes incompatible orderings, hence it maintains soundness and completeness. Since this check is performed many times, an incremental algorithm is needed for good performance. We exploit the encoding of DTNs in Satisfiability Modulo Theory (SMT) in (Cimatti, Micheli, and Roveri 2015b).

If we use ordered states, the added constraint can be strengthened to represent the order encoded in the state.

$$\mathcal{C}_{add} \doteq \{x_{i+1} - x_i \in [0, \infty] \mid \langle x_1, \cdots x_n\rangle = P, i \in [1, n-1]\} \\ \cup \{y - x_n \in [0, \infty] \mid \langle x_1, \cdots x_n\rangle = P, y \in \mathcal{T} \setminus P\}$$

Intuitively, we exploit the total order stored within states to build a stronger constraint: we force the order of time-points using $[0, \infty]$ constraints and assert that all the other time points occur after the last time point in the ordering. The added constraints are linear in the number of time points, while in the unordered case they are quadratic.

Experimental Evaluation

We implemented the validation and synthesis algorithms in a tool called PYDC. The solver, written in Python, uses the PyDBM (Bulychev 2012) library to manipulate time regions and the PYSMT (Gario and Micheli 2015) interface to call the MathSAT5 (Cimatti et al. 2013) SMT solver. We analyzed four versions of our synthesis algorithm: UNORDERED-NOH, that is the synthesis algorithm with no pruning; ORDERED-NOH, that is the synthesis algorithm with no pruning that considers ordered states, UNORDERED-SMT and ORDERED-SMT that use incremental SMT solving for pruning the unfeasible paths. The benchmark set (Cimatti, Micheli, and Roveri 2015a) is composed by 3465 randomly-generated instances (1354 STNU, 2112 DTNU). All are known to be weakly controllable, but only 2354 are (known to be) dynamically controllable. The benchmarks range from 4 to 50 time points. The tool and the benchmarks are available at `https://es.fbk.eu/people/amicheli/resources/aaai16`.

We compare our approach against TIGA-NNF and TIGA-DNF, that are the two encodings proposed in (Cimatti et al. 2014a). These encoding are the only available techniques able to solve the DTNU-DC problem. We use the UPPAAL-TIGA (Behrmann et al. 2007) state-of-the-art TGA solver to solve the encoded game. The results, obtained with time and memory limits set to 600s and 10GB, are shown in Figure 3. For lack of space, we do not discuss the performance of the validation check; we remark it is negligible compared to synthesis.

We first notice that our synthesis techniques are vastly superior to the TIGA-based approaches (cactus plot, left). The approach is able to solve 2543 against the 799 of TIGA-DNF, with difference in run-times of up to three orders of magnitude (scatter plot, center). The cactus plot also shows that the SMT-based pruning yields a significant performance boost, both for the unordered case (from 1531 to 2289 solved instances), and the ordered case (from 1552 to 2543). Finally, the scatter plot on the right shows that the ordered case is vastly superior to the unordered one. Further inspection shows that UNORDERED-SMT explores an average of 2447.9 symbolic states, compared to the 95.8 of ORDERED-SMT. In the latter case, the ordering information allows the SMT solver to detect unfeasible branches much earlier.

Conclusions and Future Work

In this paper we tackled the problem of Dynamic Controllability (DC) for the general class of Disjunctive Temporal Network with Uncertainty (DTNU). By considering strategies in the form of an executable language, we obtain a radically different view on the problem. This paves the way to two key contributions. We propose the first procedure to validate whether a given strategy is a solution to the DC problem: this is a fundamental step for settings where strategies are hand-written, or have been manually modified. Then, we define a decision procedure for DTNU-DC, that is able to synthesize executable strategies. At the core, we combine techniques derived from Timed Games and Satisfiability Modulo Theory. The experimental evaluation demonstrates dramatic improvements wrt the state-of-the-art.

In the future, we will compare the run-time properties of executable strategies (performance, footprint) with respect to reasoning-based run-time execution (Hunsberger 2013; Morris 2014). Moreover, we will explore the existence of efficient subclasses of the DC-DTNU problem, e.g. by restricting the structure of the strategies.

References

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*. IOS Press. 825–885.

Behrmann, G.; Cougnard, A.; David, A.; Fleury, E.; Larsen, K.; and Lime, D. 2007. Uppaal-tiga: Time for playing games! In *CAV*. 121–125.

Bengtsson, J. 2002. *Clocks, DBM, and States in Timed Systems*. Ph.D. Dissertation, Uppsala University.

Bulychev, P. 2012. The uppaal pydbm library — http://people.cs.aau.dk/ adavid/udbm/python.html.

Cassez, F.; David, A.; Fleury, E.; Larsen, K. G.; and Lime, D. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, 66–80.

Cimatti, A.; Griggio, A.; Schaafsma, B. J.; and Sebastiani, R. 2013. The MathSAT5 SMT solver. In *TACAS*, 93–107.

Cimatti, A.; Hunsberger, L.; Micheli, A.; Posenato, R.; and Roveri, M. 2014a. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *TIME*.

Cimatti, A.; Hunsberger, L.; Micheli, A.; and Roveri, M. 2014b. Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In *AAAI*.

Cimatti, A.; Micheli, A.; and Roveri, M. 2015a. An smt-based approach to weak controllability for disjunctive temporal problems with uncertainty. *Artificial Intelligence* 224:1–27.

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

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Effinger, R. T.; Williams, B. C.; Kelly, G.; and Sheehy, M. 2009. Dynamic controllability of temporally-flexible reactive programs. In *ICAPS*.

Frank, J.; Jonsson, A.; Morris, R.; and Smith, D. E. 2001. Planning and scheduling for fleets of earth observing satellites. In *i-SAIRAS*.

Gario, M., and Micheli, A. 2015. pySMT: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop*.

Hunsberger, L. 2009. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *TIME*, 155–162.

Hunsberger, L. 2010. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *TIME*, 121–128.

Hunsberger, L. 2013. A faster execution algorithm for dynamically controllable stnus. In *TIME*, 26–33.

Hunsberger, L. 2014. A faster algorithm for checking the dynamic controllability of simple temporal networks with uncertainty. In *ICAART*.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In *STACS*, 229–242.

Morris, P. H., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *AAAI*, 1193–1198.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In Nebel, B., ed., *IJCAI*, 494–499. Morgan Kaufmann.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *CP*. 375–389.

Morris, P. 2014. Dynamic controllability and dispatchability relationships. In *CPAIOR*, volume 8451, 464–479.

Peintner, B.; Venable, K. B.; and Yorke-Smith, N. 2007. Strong controllability of disjunctive temporal problems with uncertainty. In *CP*, 856–863.

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151:43–89.

Venable, K. B., and Yorke-Smith, N. 2005. Disjunctive temporal planning with uncertainty. In *IJCAI*, 1721–1722.

Venable, K. B.; Volpato, M.; Peintner, B.; and Yorke-Smith, N. 2010. Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *COPLAS workshop*, 50–59.

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