Disjunctive Temporal Networks with Uncertainty via SMT: Recent Results and Directions [1]

Andrea Micheli
Embedded Systems Unit, Fondazione Bruno Kessler, Via Sommarive 18, 38123 Trento, Italy
amicheli@fbk.eu

Abstract.
Many Planning and Scheduling systems are designed assuming that the system under control is able to decide the duration of all the activities being executed. However, in many application scenarios this assumption is not acceptable because the actual timing of actions is not under direct control of the plan executor. Hence, new Planning and Scheduling techniques are needed to deal with this temporal uncertainty explicitly.

In this paper, we summarize and systematize a series of works in which we addressed this uncertainty problem in the realm of temporal network scheduling. We show how Satisfiability Modulo Theory (SMT) solvers can be exploited to quickly solve different kinds of query in this setting. In particular, we focus on the framework of Disjunctive Temporal Networks with Uncertainty and address the three degrees of controllability for the fully-disjunctive class of problems, solving several open problems in the literature and experimentally showing the performance of the developed techniques. Finally, we outline and discuss several foreseeable directions of research in this field.

Keywords: Scheduling under Uncertainty, Temporal Problems with Uncertainty, Temporal Uncertainty, Satisfiability Modulo Theory

1. Introduction

Planning is the problem of synthesizing a strategy or a course of actions to control a known system in order to achieve a desired goal [27]. We focus on temporal planning and scheduling techniques, that are planning algorithms that can deal with systems having real-time constraints (such as synchronizations and deadlines) and that consider the precise timing of the activities. A lot of research has been devoted to these issues over the years, but many planning and scheduling systems still assume that the duration of each activity being planned as well as all the temporal constraints are under control of the plan executor that can freely schedule its activities, without any uncertainty. In real world scenarios, however, the duration of an activity is not always *controllable*. For example, the duration of a car trip from San Francisco to Los Angeles is not under the complete control of the driver: it also depends on traffic and weather conditions. In this kind of applications, different approaches are possible for planning. One possibility is to disregard uncertainty, estimating beforehand the duration of the trip: if, during execution, the trip is taking longer or shorter than expected, a new plan needs to be generated. Another idea is to come up with a plan that does not commit to a specific duration of the trip, but tries to be as general as possible in either a formal or a best-effort way.

[1]This paper is based on the PhD Thesis work in [41] titled "Planning and Scheduling in Temporally Uncertain Domains" that won the *"Marco Cadoli"* award for the best PhD dissertation granted by the Italian Association for Artificial Intelligence. The article summarizes the thesis contribution concerning temporal networks scheduling and discusses the current status on this line of research as well as several promising future directions.

Recently, we have been involved in a line of research aiming at relaxing this assumption to allow for more realistic and effective temporal planning. The idea is to explicitly model the uncertainty in the problem, allowing activities that have *uncontrollable* duration (assuming that minimal and maximal bounds for the trip duration are given). In the example above, this amounts to synthesize a strategy for the journey that is guaranteed to achieve the goal regardless of the traffic conditions, assuming that the duration of the trip will stay in the modeled bounds. Differently from other works, we are not interested in the probability distribution of the trip duration: we want to provide a plan that is guaranteed to work for every possible duration of the trip, not to maximize probabilistic expectation.

This controllability issue can be tackled at different levels; in this paper we focus on the problem of scheduling a set of activities having uncertain duration. This is the basic ingredient for the development of many planning techniques [18]. This paper reconstructs and summarizes our work in the realm of scheduling, collecting and presenting the main results in a coherent and holistic way.

The main formalism used in planning to model temporal knowledge is the Temporal Network (TN) [24]. In essence, a TN is a set of temporal constraints expressed over a set of time-valued variables that represent the time points that need to be scheduled. Over the years, the framework has been extended to represent temporal uncertainty [58] and the resulting Temporal Network with Uncertainty (TNU) is ideal to model activities having uncontrollable duration.

This paper discusses the following contributions. We first re-visit the problem of strong controllability of TNU, providing a set of encodings of the problem into the Satisfiability Modulo Theory (SMT) framework that are empirically evaluated to show their efficiency with respect to the state-of-the-art. Second, we address the weak controllability decision problem for the disjunctive class of TNUs by reducing it to SMT. This accounts for the first implementation of a decision procedure for this problem. Third, we address the open problem of weak strategy extraction. We provide a portfolio of algorithms for both the classes and show their empirical performance. Fourth, we tackle the dynamic controllability problem for the disjunctive class of TNUs. We provide a reduction of the problem to a reachability game in a linear-sized Timed Game Automata (TGA). This not only is the first solution technique for the open problem, but it also provides the first dynamic solution in closed form. Then, we exploit the ideas behind the formal TGA encoding to provide a more efficient, dedicated solution technique. Finally, we present a discussion on the status and the foreseeable next steps for this research line.

Structure of the Paper. The paper is structured as follows. Section 2 presents some background and notation needed for the rest of the paper. In section 3 we introduce the TNU formalism and the relevant queries. Sections 4 to 6 are concerned with strong, weak and dynamic controllability, respectively. In section 7 we survey the related literature while in section 8 we discuss the status of this research line and we outline several possible research directions. Finally, we conclude in section 9.

2. Background

Technical Preliminaries Our setting is standard first order logic [35]. The first-order signature is composed of constants, variables, function symbols, Boolean variables, and predicate symbols. A term is either a constant, a variable, or the application of a function symbol of arity n to n terms. A theory constraint (also called a theory atom) is the application of a predicate symbol of arity n to n terms. An atom is either a theory constraint or a Boolean variable. A literal is either an atom or its negation. A clause is a finite disjunction of literals. A formula is either true ($\top$), false ($\bot$), a Boolean variable, a theory constraint, the application of a propositional connective ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) of arity n to n formulae, or the application of a quantifier ($\forall$, $\exists$) to an individual variable and a formula. If t_1 and t_2 are terms, and ϕ is a formula, an if-then-else (ITE) term is $ite(\phi, t_1, t_2)$. The semantics of an ITE term is the usual if-then-else semantics from programming languages. For example, the term $ite(x > y, x, y)$ where x and y are numeric variables, corresponds to the maximum between x and y. An ITE term $ite(\phi, t_1, t_2)$ occurring in a formula ψ can be rewritten by substituting each occurrence with a fresh variable v and by conjoining $(\neg\phi \vee (v = t_1)) \wedge (\phi \vee (v = t_2))$. See [34] for a thorough discussion. We use $x, y, v, \dots$ for variables, and $\vec{x}, \vec{y}, \vec{v}, \dots$ for vectors of individual variables. Terms and formulae are referred to as expressions. Formulae are denoted with Greek letters: $\phi, \psi, \dots$. Let $\vec{x}$ be a vector of variables, we indicate the i-th variable in the vector with x_i. We write $\phi(x)$ to highlight the fact that x occurs in ϕ, and $\phi(\vec{x})$ to highlight the fact that the free variables of ϕ are variables in $\vec{x}$. We indicate with $Q\vec{x}.\phi(\vec{x})$ the formula $Qx_1.Qx_2.\ldots Qx_n.\phi(x_1,\ldots,x_n)$, where $Q \in \{\forall,\exists\}$.

Let $\phi(\vec{x}) \doteq \bigwedge_i \phi_i(\vec{x}_i)$ be a conjunction of formulae. We write $\phi(\vec{x})|_{\vec{y}}$ to represent the conjunction of the $\phi_i(\vec{x}_i)$ in which at least one variable of $\vec{y}$ occurs in $\vec{x}_i$.

Substitution is defined in the standard way [35]. We write $\phi[s/t]$ for the substitution of every occurrence of term t in ϕ with term s. Let $\vec{t}$ and $\vec{s}$ be vectors of terms having the same length, we write $\phi[\vec{s}/\vec{t}]$ for the parallel substitution of every occurrence of t_i in ϕ with s_i.

We use the standard semantic notions of interpretation and satisfiability. We call *satisfying assignment* or *model* of a formula $\phi(\vec{x})$ a total function μ that assigns to each x_i an element of its domain such that the formula $\phi[\mu(\vec{x})/\vec{x}]$ evaluates to $\top$ provided an interpretation of function symbols. A formula $\phi(\vec{x})$ is *satisfiable* if and only if it has a satisfying assignment and an interpretation. Following standard naming, we say that a formula is in Conjunctive Normal Form (CNF) if it is expressed as a conjunction of disjunctions of atoms or negations of atoms: $\bigwedge_{i=1}^{h} \bigvee_{j=1}^{k} l$, l being a literal. It is well known that each formula can be reduced to an equi-satisfiable CNF formula of linear size [22].

Satisfiability Modulo Theories Given a formula ϕ, satisfiability is the problem of finding a satisfying assignment for ϕ and an interpretation for the functional symbols in ϕ. This problem is approached in propositional logic with enhancements of the DPLL algorithm [21]: the formula is converted into an equi-satisfiable one in Conjunctive Normal Form (CNF); then, a satisfying assignment is incrementally built, until either all the clauses are satisfied, or a conflict is found, in which case back-jumping takes place (i.e. certain assignments are undone). Keys to efficiency are heuristics for the variable selection, and learning of conflicts [47].

Given a first-order formula ψ expressed in a decidable background theory T, *Satisfiability Modulo Theory* (SMT) [4] is the problem of deciding whether ψ is satisfiable. For example, consider the formula $(x \le y) \wedge ((x+3 = z) \vee (z \ge y))$ in the theory of real arithmetic ($x, y, z \in \mathbb{R}$, and the symbols $\le$, $+$, $=$ and $\ge$ are interpreted in the usual way). The formula is satisfiable and a satisfying assignment is $\{x := 5,\ y := 6,\ z := 8\}$. The theory of real arithmetic interprets "3" as the real number 3 and $+, =, <, >, \le, \ge$ as the usual mathematical functions and relations.

There exist several theories of practical interests: *Equality and Uninterpreted Functions*, *Linear Arithmetic* over the Reals and the Integers, *Non-Linear Arithmetic*, *Difference Logic*, *Bit Vectors*, *Arrays* and others. In this paper, we concentrate on the theory of linear arithmetic over the real numbers ($\mathcal{LRA}$) because it offers a natural and convenient way to model continuous, dense time. A formula in $\mathcal{LRA}$ is an arbitrary Boolean combination, a universal ($\forall$) or an existential ($\exists$) quantification, of atoms in the form $\sum_i a_i x_i \bowtie c$ where $\bowtie \in \{>, <, \le, \ge, \neq, =\}$, every x_i is a real variable, every a_i is a real constant and c is also a real constant. For brevity, given two real constants l, u such that $l \le u$, we denote with $t \in [l,u]$ the formula $l \le t \wedge t \le u$. With a slight abuse of notation, we allow l or u to be $-\infty$ or $+\infty$, respectively. The semantics is obtained by simply removing the constraint containing ∞ as it is tautological. For example, $t \in [2,\infty]$ becomes $t \ge 2$; $t \in [-\infty, 5]$ becomes $t \le 5$ and $t \in [-\infty,\infty]$ becomes $\top$. Real Difference logic ($\mathcal{RDL}$) is the fragment of $\mathcal{LRA}$ where all the atoms have the form $x_i - x_j \bowtie c$. We denote with $\mathcal{QF\_LRA}$ and $\mathcal{QF\_RDL}$ the quantifier-free fragments of $\mathcal{LRA}$ and $\mathcal{RDL}$, respectively.

The most efficient implementations of SMT solvers use the so-called "lazy approach", where a SAT solver is tightly integrated with a theory-specific solver for conjunctions of constraints (called T-solver). The role of the SAT solver is to enumerate the truth assignments to the Boolean abstraction of the first-order formula. The Boolean abstraction has the same Boolean structure of the first-order formula, but "replaces" the predicates which contain theory information with fresh Boolean variables. The Boolean abstraction of $(x \le y) \wedge ((x+3 = z) \vee (z \ge y))$ is $a \wedge (b \vee c)$, where a, b, c are fresh Boolean variables. The T-solver is invoked when the SAT solver finds a satisfying assignment for the Boolean abstraction: the satisfying assignment to Boolean abstraction maps directly to a conjunction of T atoms, which the T-solver can handle. If the conjunction is satisfiable also the original formula is satisfiable. Otherwise the T-solver returns a conflict set which identifies a reason for the unsatisfiability. Then, the negation of the conflict set is learned by the SAT solver in order to prune the search. Examples of solvers based on the "lazy approach" are MATHSAT [7,10], Z3 [23], YICES [25] and OPENSMT [8].

Traditionally, SMT solvers have been focused on solving quantifier-free formulae, but techniques to deal with quantifiers on selected theories have been recently developed and implemented. For the sake of this paper we are interested in dealing with quantifiers in the $\mathcal{LRA}$ theory. Some solvers (e.g. Z3) natively support

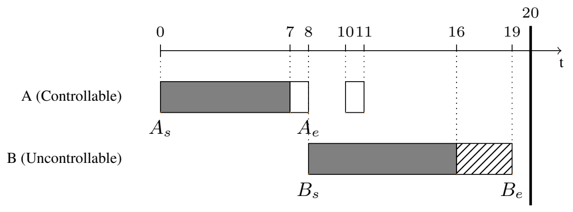


Fig. 1. Schema of a possible temporal allocation in the running example. Activities are depicted in time, filled regions are used to indicate the minimal guaranteed duration of an activity, the region in which uncontrollable event B_e can happen is striped, while the region in which A_e can be scheduled is the union of the two white rectangles. The problem deadline is indicated with the solid line at time 20.

$\mathcal{LRA}$ quantifiers, but others (e.g. MATHSAT) are still limited to the quantifier-free fragment.

A theory T is said to admit quantifier elimination, if for every quantified formula ϕ in T, there exist a quantifier-free formula ϕ' that is logically equivalent to ϕ. It has been proven that $\mathcal{LRA}$ admits quantifier elimination [51]. this means that from any $\mathcal{LRA}$ formula it is possible to derive a *logically-equivalent*, quantifier-free $\mathcal{LRA}$ formula (in $\mathcal{QF\_LRA}$). Several automated techniques for quantifier elimination exist for the $\mathcal{LRA}$ theory: Fourier-Motzkin [51] and Loos-Weispfenning [38] are two well-known examples.

3. Temporal Networks with Uncertainty

In this section, we formalize various classes of temporal networks proposed in the literature. In particular, we focus on disjunctive temporal networks with uncertainty (DTNU). We start by proposing an example.

Suppose we have two activities A and B. Activity A has duration of at least 7 units and at most 8 units or at least 10 units and at most 11 units, depending on a controllable decision. Activity B is uncontrollable, meaning that the actual duration is not decidable by the solver, but we can assume that it is at least 8 units and at most 11 units. We require that activity B must start after activity A and both activities must end within 20 units. The example is depicted in figure 1.

A Temporal Network (TN) is a formalism that is used to represent temporal constraints over time-valued variables representing time points. Two families of TNs have been presented in literature over the years: TN without uncertainty, in which all the time points can be freely assigned by the solver [24,53] and TN with uncertainty (TNU), in which only some of the time points can be assigned by the solver, while the others are intended to be assigned by an adversary [50,58]. As such, TNUs can be seen as a form of game between the solver and an adversarial environment. In this part we focus on the DTNU class, where disjunctions in constraints are allowed.

Definition 1 (DTNU [50]). *A DTNU is a tuple $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, where:*

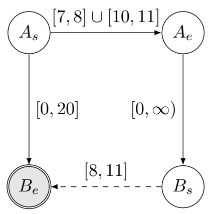
1. *$\mathcal{T}$ is a set of time points, partitioned into controllable ($\mathcal{T}_c$) and uncontrollable ($\mathcal{T}_u$);*
2. *$\mathcal{C}$ is a set of* free constraints*: each constraint c_i is of the form, $\bigvee_{j=1}^{D_i} t_{1,j} - t_{2,j} \in [l_{i,j}, u_{i,j}]$, for some $t_{1,j}, t_{2,j} \in \mathcal{T}$ and $l_{i,j}, u_{i,j} \in \mathbb{R} \cup \{+\infty, -\infty\}$; and*
3. *$\mathcal{L}$ 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 $\langle l_{i,j}, u_{i,j} \rangle$ such that $0 < l_{i,j} < u_{i,j} < \infty$, $j \in [1, E_i]$ (E_i being $|B_i|$); and for any distinct pairs, $\langle l_{i,j}, u_{i,j} \rangle$ and $\langle l_{i,k}, u_{i,k} \rangle$ in $\mathcal{B}_i$, either $l_{i,j} > u_{i,k}$ or $u_{i,j} < l_{i,k}$.*

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 will fulfill. As in previous work [50,58], we consider only contingent links that start with a controllable time point. Thus, 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 [1] of e (indicated with $\alpha(x)$).

Within the framework of DTNU, we can only express uncertainty on the duration of activities (i.e. we cannot express uncertainty on whether an activity could occur or not, nor on its discrete outcome). Contingent links are used to model the possible durations of the uncontrollable activities, while uncontrollable time points represent the uncontrollable ending time of activities. We remark that the Temporal Network model of time is continuous, and we explicitly avoid any discretization seeking for a real-valued solution.

Any temporal network is defined over a set of time points, namely variables representing time instants. A temporal allocation such as the one in our running example can be encoded in a temporal network by using two time points to represent the starting and ending

[1]This formulation assumes a complete independence between contingent links. This means that this formalism cannot express assumptions of interdependence between uncontrollable durations.



(a)

$$\begin{aligned}
&\mathcal{T}_c \doteq \{A_s, A_e, B_s\}\\
&\mathcal{T}_u \doteq \{B_e\}\\
&\mathcal{C} \doteq \{B_e - A_s \in [0,20], B_s - A_e \in [0,\infty),\\
&\qquad A_e - A_s \in [7,8] \vee A_e - A_s \in [10,11]\}\\
&\mathcal{L} \doteq \{\langle B_e, \{\langle 8,11\rangle\}, B_s\rangle\}
\end{aligned}$$

(b)

Fig. 2. The TNU model derived from the running example (b) and its graphical representation (a). Each node of the graph is a time point, doubly circled nodes are uncontrollable the others are controllable, solid arrows are free constraints and dashed arrows are contingent constraints.

time of each activity. Therefore, in order to model the running example with a temporal network we need a total of four time points A_s, A_e, B_s and B_e representing the start and end of activity A and B respectively (figure 1). B_e is the only uncontrollable time point, as we can control the starting and end time of A but we cannot control the duration of B. The only contingent constraint is the constraint on the duration of B. The rest of the network is composed of requirements that have to be always fulfilled, and can be translated in three free constraints. The resulting TNU is shown in figure 2.

For the sake of this paper, we define a *TN without uncertainty* as a TNU $\langle \mathcal{T}, \mathcal{C}, \emptyset \rangle$, in which the set of uncontrollable time points $\mathcal{T}_u$ is empty and the set of contingent links $\mathcal{L}$ is also empty. Excluding the $\emptyset$ in the tuple, this coincides with the Definitions of STN, TCSN and DTN.

We define an assignment to the time points as a total function from time points to real values. Given a TN without uncertainty, checking *consistency* corresponds to deciding the existence of an assignment that fulfills all the constraints of the network. We call such an assignment a *consistent schedule*, and we say that the TN is *consistent*. Checking the consistency of a TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ is defined as checking the consistency of the TN without uncertainty $\langle \mathcal{T}, \mathcal{C} \cup \rho(\mathcal{L}), \emptyset \rangle$, where $\rho(\mathcal{L})$ is the set of constraints obtained by considering each contingent link as a requirement constraint. Formally, $\rho(\mathcal{L}) \doteq \{\rho(x) \mid x \in \mathcal{L}\}$ and

$$\rho(\langle b, \mathcal{B}, e \rangle) \doteq \bigvee_{\langle l,u \rangle \in \mathcal{B}} e - b \geq l \wedge e - b \leq u \,.$$

The consistency problem can be directly and efficiently solved by SMT solvers equipped with the $\mathcal{QF\_RDL}$ theory [3,17].

Depending on the generality of the constraints in $\mathcal{C}$ and the maximal cardinality of the sets $\mathcal{B}_i$ of the elements of $\mathcal{L}$, three classes of TNUs are possible [50]. Definition 1 in its general form identifies *Disjunctive Temporal Network with Uncertainty* (DTNU) [50]. If each constraint contained in $\mathcal{C}$ is defined on at most two time points, the resulting network is a *Temporal Constraint Satisfaction Network with Uncertainty* (TCSNU).

If each constraint in $\mathcal{C}$ has exactly one disjunct and each $\mathcal{B}_i$ has exactly one element, we obtain a *Simple Temporal Network with Uncertainty* (STNU).

Similarly, we can define the corresponding TNs without uncertainty (DTN [53], TCSN, and STN [24]). Following the classification of Peintner et al. [50], we also say that a network is *simple-natured* if each $\mathcal{B}_i$ has exactly one element (An STNU is always simple-natured).

Given a TNU, values for controllable time points can be decided, namely they can be scheduled in time by an executor, while an uncontrollable time point e_i just happens after its activation time point b_i has been scheduled. The only assumption is that the i-th contingent link will be satisfied by the values of b_i and e_i. Given this intuitive meaning, we rephrased the concept of situation for a TNU [58] for the DTNU problem class.

Definition 2 (Situation)**.** *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU and let m be the number of uncontrollable time points ($|\mathcal{T}_u| = m$).*

The space of situations for P is a set of tuples $\Omega_P \doteq S_1 \times \cdots \times S_m$, where $S_i \doteq \bigcup_{j=1}^{E_i} [l^c_{i,j}, u^c_{i,j}]$. A situation is an element ω of Ω_P, and we write ω_{e_i} to indicate the value of the $i-th$ element of the tuple (i.e. the duration of the contingent link for e_i).

Intuitively, a situation is a choice of the actual duration for each activity with uncontrollable duration.

Given a situation, we define the *projection* of a TNU as the TN obtained fixing the duration of each contingent link.

Definition 3 (Projection)**.** *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU and let $\omega \doteq \langle \omega_1, \ldots, \omega_{|\mathcal{T}_u|} \rangle$ be a situation in Ω_P. The projection P_ω of the network P with respect to the situation ω is the TN $\langle \mathcal{T}, \mathcal{C}', \emptyset \rangle$, where $\mathcal{T} = \mathcal{T}_c$, $\mathcal{C}' \doteq \mathcal{C} \cup \{e_i - b_i \in [\omega_i, \omega_i] \mid \langle b_i, \mathcal{B}_i, e_i \rangle \in \mathcal{L}\}$.*

Intuitively, the projection P_ω is the network without uncertainty in which each uncontrollable duration has been fixed to a given value.

Interestingly, a number of possible queries are possible when TNU are concerned. In fact, depending on the amount of observation that we assume for the executor of the schedule, three different problems can be defined.

The first query is strong controllability that consists in deciding the existence of an assignment to controllable time points that fulfills the free constraints under any assignment of uncontrollable time points that satisfies the contingent constraints. Such an assignment is called a *strong schedule* of the network. A TNU for which there exists a strong schedule is said to be *strongly controllable*. Consider again the running example, the network is strongly controllable and a strong schedule is $\mu' = \{A_s = 0, A_e = 8, B_s = 8\}$.

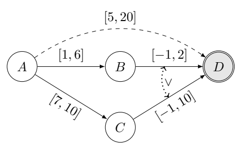
Definition 4 (Strong Controllability)**.** *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU. P is strongly controllable if there exists an assignment μ for $\mathcal{T}_c$ such that for each situation $\omega \in \Omega_P$, μ is a consistent schedule for the projection P_ω.*

Another query that can be addressed in the context of TNUs is weak controllability, that is concerned with the existence of a strategy that associates values to the controllable starting points of each activity, as a function of the uncontrollable durations. The values for the uncontrollable durations are not known at the moment of solving the problem; however, the executor is given the actual value of such durations just before the execution starts.

Definition 5 (Weak Controllability)**.** *Let $P \doteq (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be a TNU. P is weakly controllable if and only if for each situation $\omega \in \Omega_P$ the projection P_ω is consistent.*

In terms of games, weak controllability is the dual of strong controllability: in strong controllability, the executor is required to make its move (i.e. all its decisions) without observing the situation (i.e. the move of the environment); in weak controllability, the environment is required to make all its decisions before the executor. For example, consider the DTNU in figure 3. The problem is weakly controllable and a possible weak strategy is to schedule A at time 0 and B at time 6, if D happens before time 7 we schedule C at time 7, otherwise we schedule C at the same time as D.

The last kind of query that can be addressed given a TNU is dynamic controllability. Dynamic controllability is concerned with the existence of a strategy for executing the controllable time points that depends only on past observations of the outcomes of uncontrollable durations, and that guarantees that all relevant constraints will be satisfied no matter how the durations of the contingent links turn out. Essentially, a TNU is dynamically controllable if there exists a weak strategy that, in order to decide a controllable time point at time k, does not depend on any observation past time k. In the case of the DTNU in figure 3, the agent seeks a strategy for executing the controllable time points, A, B and C, that will guarantee that the constraints are satisfied, no matter what durations the environment happens to pick for the contingent link, $\langle A, \{\langle 5, 20\rangle\}, D\rangle$. For example, the agent might decide to execute A at time 0 and then wait. Should the environment happen to "choose" a duration of 5 for the contingent link, the agent would observe, at time 5, the execution of D. For example, the agent might then react executing B at time 6. Later, the agent will have to schedule C between time 7 and 10. In this example evolution, all the time points have been executed and all constraints in $\mathcal{C}$ are satisfied, hence the agent has succeeded. It can be checked that this DTNU is dynamically controllable (i.e., there exists a strategy for the agent that ensures success no matter how the environment behaves).



(a)

$\mathcal{T}_c \doteq \{A, B, C\}$
$\mathcal{T}_u \doteq \{D\}$
$\mathcal{C} \doteq \{B - A \in [1, 6], C - A \in [7, 10],$
$D - B \in [-1, 2] \vee D - C \in [-1, 10]\}$
$\mathcal{L} \doteq \{\langle A, \{\langle 5, 20\rangle\}\rangle, D\}$

(b)

Fig. 3. A dynamically controllable DTNU (b) and its graphical representation (a).

In the following, we will present the results and techniques we developed one query at the time for the general DTNU case.

4. Strong Controllability

Strong controllability is an important problem, because it results in a schedule that is satisfactory under all possible uncertainties. Clearly, a strong schedule can yield a longer time-span compared to a dynamic strategy. However, dynamic information may not be available, e.g. due to the lack of sensors. Furthermore, most algorithms for dynamic execution require runtime reasoning [30]. This may be incompatible with some operational settings: for example, in mission-critical systems, validating the run-time reasoner to the required level of assurance may be prohibitively hard. Furthermore, the computational resources available during execution may be too limited for a dynamic approach. Examples of such application domains can be found in production scheduling and in mission critical robotics, for which strong controllability is a very relevant problem. We also remark that, in the same domains, the expressiveness of disjunctive constraints (compared to simple temporal problems) is often necessary [48].

In this section, we present a comprehensive and effective approach for solving the strong controllability problem for TNUs in the most general form including arbitrary disjunctions.

We tackle the strong controllability problem of TNUs by reduction to SMT. The resulting problem can be then solved by efficient SMT solvers. This reduction has been published in [14] and [17], here we give an overview of the techniques and the results.

Given a TNU $P = \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, we derive a number of encodings of the problem using the expressive power of the SMT framework. We assume each time point is associated with an SMT variable: for every time point in $x \in \mathcal{T}$ we introduce a real SMT variable[2] x, and we denote with $\vec{T}$ the vector of such variables (imposing an arbitrary order); each constraint $c \in \mathcal{C}$ is directly mapped on the corresponding SMT formula (indicated by $[\![c]\!]$) by keeping the Boolean structure of the constraint and substituting each time point with the corresponding SMT variable.

The first encoding in equation (1) is a direct logical mapping of the notion of strong controllability; we call this encoding *direct encoding*. We indicate with $\vec{T}_c$ and $\vec{T}_u$ the SMT variables corresponding to $\mathcal{T}_c$ and $\mathcal{T}_u$, respectively.

$$\forall \vec{T}_u . [\![\rho(\mathcal{L})]\!] \rightarrow \bigwedge_{c_i \in \mathcal{C}} [\![c_i]\!] \tag{1}$$

Proposition 1. *The TNU P is strongly controllable if and only if equation* (1) *is satisfiable (and a model of equation* (1) *yields a strong schedule for P).*

Proposition 1 is directly obtained by formalizing the definition of strong controllability. Intuitively, equation (1) is satisfiable if and only if there exists an as-

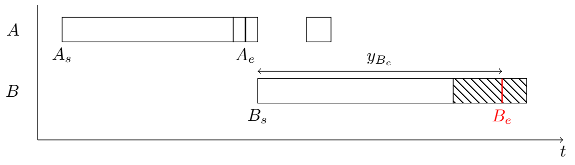


Fig. 4. The running example of figure 2 seen from an activities point of view to explain the encoding of the problem. The striped region is the uncontrollable space, namely where the uncontrollable time point B_e can be scheduled given the decision on the related controllable time point (B_s). The value of y_{B_e} in the shown scenario is seen as the actual duration of the B activity.

signment to the controllable variables $\mathcal{T}_c$ such that, for all assignments to the uncontrollable variables $\mathcal{T}_u$ (that is, for each situation) satisfying the contingent links $\mathcal{L}$, the free constraints $\mathcal{C}$ are also satisfied. In the above formula, the controllable variables are implicitly existentially quantified. In case of satisfiability, the SMT solver returns a satisfying assignment to the controllable variables that is exactly a strong schedule.

In order to enable further simplifications, we notice that contingent constraints depend both on controllable and uncontrollable time points, and we re-code the problem as follows. We encode each uncontrollable time point e_i in terms of the time difference with its starting time point $b_i \doteq \alpha(e_i)$ by means of an uncontrollable duration variable y_{e_i}. Intuitively, if we take an activity view, y_{e_i} measures the duration of the i-th activity. For every contingent link $l_i = \langle b_i, \mathcal{B}_i, e_i \rangle$ with $\mathcal{B}_i = \{\langle \ell_{i,1}, u_{i,1} \rangle, \langle \ell_{i,E_i}, u_{i,E_i} \rangle\}$, let $y_{e_i} \in \mathbb{R}$ be the uncontrollable offset variable associated to e_i such that $\bigvee_{j=1}^{E_i} (y_{e_i} \in [\ell_{i,j}, u_{i,j}])$. y_{e_i} represents the duration of the interval $[b_i, e_i]$ that is constrained by the i-th contingent link. We are thus symbolically encoding a situation $\omega \doteq (\omega_1, \ldots, \omega_{|\mathcal{T}_u|})$ in which y_{e_i} models the value of ω_i. Figure 4 gives a pictorial representation of this encoding interpreted at the activity level.

Definition 6 (TNU Encoding)**.** *Given a TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ with $\mathcal{T}_u \doteq \{e_1, e_2, \cdots, e_m\}$, let $\vec{Y}_u \doteq \langle y_{e_1}, y_{e_2}, \ldots, y_{e_m} \rangle$ be the vector of uncontrollable duration variables. We define the encoding of the problem as a tuple $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ where*

$$\Gamma(\vec{Y}_u) \doteq \bigwedge_{i=1}^{m} \bigvee_{j=1}^{E_i} (y_{e_i} \in [l_{i,j}, u_{i,j}])$$

and $\Psi(\vec{T}_c, \vec{Y}_u)$ is defined as:

$$\bigwedge_{c \in \mathcal{C}} c[(\alpha(e_1) + y_{e_1})/e_1] \ldots [(\alpha(e_m) + y_{e_m})/e_m].$$

[2]We use no graphical distinction for time points and SMT variables, because we always have a one-to-one correspondence. It shall be clear from the context whether we are referring to the time point (when we discuss about constraints) or to the SMT-variables (when we work with formulae).

Intuitively, $\Gamma(\vec{Y_u})$ is the formula representing the conjunction of all the contingent links after the re-coding, and $\Psi(\vec{T_c},\vec{Y_u})$ is the conjunction of all the free constraints rewritten in terms of $\vec{T_c}$ and $\vec{Y_u}$.

From here on, we assume an encoded problem $\langle\vec{T_c},\vec{Y_u},\Gamma(\vec{Y_u}),\Psi(\vec{T_c},\vec{Y_u})\rangle$ is given. In this setting, the strong controllability problem consists in finding a value for $\vec{T_c}$ that satisfies the free constraints $\Psi(\vec{T_c},\vec{Y_u})$ under any possible value of $\vec{Y_u}$ that satisfies $\Gamma(\vec{Y_u})$.

The strong controllability encoding in equation (1) can be re-coded as an $\mathcal{LRA}$ formula in the free variables $\vec{T_c}$ as follows.

$$\forall\vec{Y_u}.\big(\Gamma(\vec{Y_u})\rightarrow\Psi(\vec{T_c},\vec{Y_u})\big) \tag{2}$$

We call this encoding *Offset Encoding*. This formulation corresponds to a quantified SMT problem in $\mathcal{LRA}$, and still requires a solver that supports quantified formulae, but the part of the encoding representing the contingent link is now dependent on $\vec{Y_u}$ only.

The main problem in the previous encodings is the scope of the universal quantifier. Since the computational cost of quantification is very high, we can rewrite the offset encoding in equation (2) in order to obtain a more efficient encoding. Let us assume that $\Psi(\vec{T_c},\vec{Y_u})$ is written as a conjunction of formulae $\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}})$, where $\vec{T_{c_h}}\subseteq\vec{T_c}$ and $\vec{Y_{u_h}}\subseteq\vec{Y_u}$ are the variables used in the formula ψ_h. This assumption can be easily satisfied by converting $\Psi(\vec{T_c},\vec{Y_u})$ in CNF[3].

$$\Psi(\vec{T_c},\vec{Y_u})=\bigwedge_{h=1}^{H}\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}})$$

We have that $\bigwedge_h\forall\vec{Y_u}.(\neg\Gamma(\vec{Y_u})\vee\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}}))$ can be equivalently rewritten to $\bigwedge_h\forall\vec{Y_{u_h}}.(\neg\Gamma(\vec{Y_u})|_{Y_{u_h}}\vee\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}}))$, and we obtain the following *distributed encoding*. We recall that $\phi|_{\vec{x}}$ is a notation meaning the restriction of the conjunction ϕ to the conjuncts that are defined on at least one variable of $\vec{x}$.

$$\bigwedge_h\forall\vec{Y_{u_h}}.\Big(\neg\Gamma(\vec{Y_u})|_{Y_{u_h}}\vee\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}})\Big) \tag{3}$$

The size of the produced (quantified) formula is linear with respect to the original TNU. This encoding still requires a solver that supports quantified formulae, and contains as many quantifiers as conjuncts in $\Psi(\vec{T_c},\vec{Y_u})$.

In order to exploit solvers that do not support quantifiers, we propose an encoding of strong controllability into a quantifier-free SMT ($\mathcal{LRA}$) formula. This is obtained by resorting to an external procedure for quantifier elimination.

We rewrite equation (3) as $\bigwedge_h\neg\exists\vec{Y_{u_h}}.(\Gamma(\vec{Y_u})|_{Y_{u_h}}\wedge\neg\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}}))$, in order to apply a procedure for the elimination of existential quantifiers (e.g. Fourier-Motzkin [51]). In the following we refer to each conjunct after quantifier elimination as $\psi_h^{\Gamma}(\vec{T_{c_h}})$ (that is then a quantifier-free formula).

$$\psi_h^{\Gamma}(\vec{T_{c_h}})\leftrightarrow\neg\exists\vec{Y_{u_h}}.(\Gamma(\vec{Y_u})|_{Y_{u_h}}\wedge\neg\psi_h(\vec{T_{c_h}},\vec{Y_{u_h}}))$$

The resulting encoding, reported in equation (4), is called *eager for-all elimination encoding*.

$$\bigwedge_h\psi_h^{\Gamma}(\vec{T_{c_h}}) \tag{4}$$

Clearly, this approach moves most of the computation complexity from the solving of the resulting formula to the encoder. In fact, the encoder needs now to solve a number of costly quantifier eliminations.

For the simple-natured TCSNU class, it is not necessary to apply a general purpose quantifier elimination procedure. Given the specific nature of the constraints and the limitation to convex contingent constraints, only few cases are possible, and for each of them we use a pattern-based encoding, that in essence pre-computes the result of quantifier elimination. This result can be thought of as generalizing to simple-natured TCSNUs the result proposed by Fargier and Vidal [58] for the case of STNU. We highlight that for the STNU case, this special quantification technique yields (in polynomial time) an STN that can be solved in polynomial time by any SMT solver (when no disjunctions are present the problem degenerates in a single call to the theory solver that uses a single call to simplex solver to solve the entire problem).

There are eight possible clause patterns[4] shown in table 1. For unit clauses, we proceed as in the work by Fargier and Vidal [58]: the first four rows of table 1 report these results. The rest of the table present the results for the disjunctive binary clauses. The static

[3]If the used CNF transformation introduces additional variables, those are existentially quantified and extend the model of equation (1) by preserving the satisfiability and the strong schedules encoded in the models.

[4]This is because the problem can be encoded in 2-cnf using the "hole encoding" [17].

Clause pattern	**Quantification Result** ($\psi_h^\Gamma(\vec{T}_{c_h})$)
$(b_i - b_j) \geq k$	$(b_i - b_j) \geq k$
$(e_i - b_j) \geq k$	$(b_i - b_j) \geq k - L_i$
$(b_i - e_j) \geq k$	$(b_i - b_j) \geq k + U_j$
$(e_i - e_j) \geq k$	$(b_i - b_j) \geq k - L_i + U_j$
$(b_i - b_j) \leq k_1 \vee$ $(b_i - b_j) \geq k_2$	$(b_i - b_j) \leq k_1 \vee (b_i - b_j) \geq k_2$
$(e_i - b_j) \leq k_1 \vee$ $(e_i - b_j) \geq k_2$	$((b_i + L_i - b_j > k_1) \vee (b_i + U_i - b_j \leq k_1)) \wedge$ $((b_i + L_i - b_j < k_1) \vee (b_i + L_i - b_j \geq k_2))$
$(b_i - e_j) \leq k_1 \vee$ $(b_i - e_j) \geq k_2$	$((b_i - b_j - L_j < k_2) \vee (b_i - b_j - U_j \geq k_2)) \wedge$ $((b_i - b_j - L_j > k_2) \vee (b_i - b_j - L_j \leq k_1))$
$(e_i - e_j) \leq k_1 \vee$ $(e_i - e_j) \geq k_2$	$((b_i + U_i - b_j - U_j > k_1) \vee (b_i + U_i - b_j - L_j \leq k_1)) \wedge$ $((b_i + U_i - b_j - U_j < k_1) \vee (b_i + L_i - b_j - U_j \geq k_1)) \wedge$ $((b_i + L_i - b_j - L_j < k_2) \vee (b_i + L_i - b_j - U_j \geq k_2)) \wedge$ $((b_i + L_i - b_j - L_j > k_2) \vee (b_i + L_i - b_j - L_j \leq k_2))$

Table 1

Static quantification for simple-natured TCSNUs. For each clause pattern deriving from a hole-encoding of free constraints, the corresponding $\psi_h^\Gamma(\vec{T}_{c_h})$ is presented, assuming that if e_i is an uncontrollable time point, b_i is its corresponding controllable time point that relates to it with the i-th contingent link $\langle b_i, \{\langle L_i, U_i \rangle\}, e_i\rangle$.

quantification is possible by knowing that the contingent links are in the shape $e_i - b_i \in [L_i, U_i]$ and thus each possible free constraint clause can be parametrized and resolved upfront.

This specialized quantification technique results in a 2-CNF formula that has linear size in the original TCSNU. This encoding spares the computational cost of quantifier elimination and produces a highly optimized $\mathcal{QF\_LRA}$ formula.

Experimental Results. We experimentally evaluated our approach by developing a tool that automatically encodes the various classes of temporal problems as SMT problems; the SMT encodings are in turn solved by state-of-the-art solvers. We used a set of random benchmarks generated by means of an extension of the generator presented in [3], where uncertainty is randomly introduced: each constraint generated by the consistency problem generator is turned in a contingent link with a given probability, and its destination node is considered as uncontrollable. The benchmark set contains 1054 simple-natured instances for each TNU class (STNU, TCSNU and DTNU).

To the best of our knowledge, there are no available implemented solvers for strong controllability problems. Thus, we evaluated the different approaches we presented, to highlight the difference in performance and the respective merits. In addition, we also compare our approach with the PVYS algorithm described in [50]: we implemented our own version of the algorithm in a tool, written in Python, that and exploits the MATHSAT SMT solver to check the consistency of DTNs. The tool has been implemented in two variants. The first one directly follows the original pseudo-code; the second one exploits the incrementality feature of MATHSAT, to gain more efficiency: instead of checking the consistency of each problem separately, it reuses information derived from previous checks whenever possible. In order to test the effectiveness of the eager for-all elimination encodings (EFE) we experimented with different quantifier elimination techniques: the internal Z3 quantifier elimination (Z3QE), the MATHSAT5 Fourier-Motzkin implementation (M5FM) and the MATHSAT5 Loos-Weispfenning implementation (M5LW).

The results are reported in figure 5. We plotted in logarithmic scale the cumulative time in seconds to solve the considered set of benchmarks. The total time includes the encoding time, which may be significant in the case of quantifier-free encodings. The plots show that the OFFSET and DIRECT encodings quickly reach the resource limits, and are unable to solve all the instances. The behavior of the DISTRIBUTED encoding is slightly better than the eager for-all elimination approaches. The difference can be explained in purely technological terms: the quantifier elimination modules are called via pipe in our implementation, while Z3, on the DISTRIBUTED encoding, performs quantifier elimination "in-memory".

We notice that the static quantification techniques (EFE STATIC), when applicable (i.e. for STNU and simple-natured TCSNU), yield a substantial improvement in performance: the expensive quantifier elimination step is avoided altogether.

In the STNU problem class, the results of PVYS are comparable to the SMT-based approaches. This is expected, because the implementation of PVYS uses the same SMT-based calls to FARGIERVIDAL. In the disjunctive cases, PVYS performs dramatically worse than the SMT-based approaches, due to the enumerative treatment of disjunctions. Finally, we notice that incrementality improves the performance, even if the difference is not dramatic.

5. Weak Controllability

The second query of interest is weak controllability, that is concerned with the existence of a strategy that associates values to the controllable starting points of each activity, as a function of the uncontrollable durations.

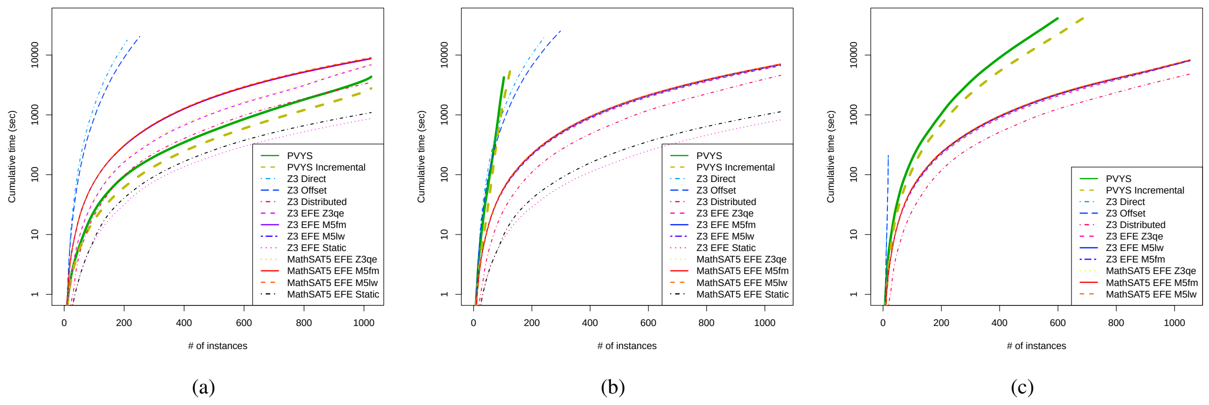


Fig. 5. Results for strong controllability experimental evaluation for the STNU (a) TCSNU (b) and DTNU (c) problem classes.

There are several reasons for studying weak controllability. From the temporal problems perspective, weak controllability is the conceptually interesting dual of the strong controllability problem. In addition, deciding whether a given TNU is weakly controllable serves as a pre-check for more complex problems such as dynamic controllability. In fact, weak controllability is a necessary condition for dynamic controllability [58]. From the practical standpoint, weak controllability allows for the modeling of a setting where a number of tasks is to be repeatedly executed, but with modalities that depend on some environmental parameters that become available just prior to execution.

We propose a general decision procedure for the problem of weak controllability for DTNUs based on a reduction to an SMT problem for the theory of Quantified Linear Real Arithmetic ($\mathcal{LRA}$). Then, we investigate the problem of on-line strategy execution, i.e. given a weakly controllable DTNU, how to repeatedly produce a suitable schedule for the controllable time points as a function of a valuation to the uncontrollable ones. This motivates the investigation of efficient runtime execution for weakly controllable TNUs. Finally, we address the synthesis problem: given a weakly controllable temporal problem, we algorithmically synthesize a function from an assignment to uncontrollable time points to an assignment to the controllable ones. We propose a number of algorithms for the synthesis of a strategy. These methods have been published in [15] and [16], here we give an overview of the techniques and the results.

Deciding Weak Controllability. Given a TNU P, we want a decision procedure that answer positively if and only if P is weakly controllable. We start by rewriting the problem as per in definition 6: we encode each uncontrollable time point e_i in terms of the time difference with its starting time point b_i by means of an uncontrollable duration variable y_{e_i}. Intuitively, a temporal problem is weakly controllable if there exists a strategy that maps every situation to a corresponding assignment to controllable time points, in such a way that all free constraints are satisfied. We can rephrase the concept of weak controllability presented in definition 5 as a satisfiability problem modulo the $\mathcal{LRA}$ theory as follows.

Proposition 2. *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU and let its encoding be $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$. P is weakly controllable if and only if the following formula is valid modulo the $\mathcal{LRA}$ theory.*

$$\forall \vec{Y}_u . \exists \vec{T}_c . (\Gamma(\vec{Y}_u) \rightarrow \Psi(\vec{T}_c, \vec{Y}_u)) \tag{5}$$

The formula in equation (5) is a direct formalization of the intuitive notion of weak controllability, and of the original definition in [58]. The universal quantifier captures the uncertainty in the decision of the duration variables. The implication ensures that free constraints are checked only when $\Gamma(\vec{Y}_u)$ is satisfied, that is only on assignments that encode situations of the original temporal problem. Equation (5) is a formula in $\mathcal{LRA}$ that is valid if and only if the problem is weakly controllable. Any SMT solver supporting $\mathcal{LRA}$ is able to deal with such a formula directly and it can correctly solve the problem. However, due to the high computational cost of directly handling quantifiers, an optimized encoding is required.

We first rewrite the formula encoding weak controllability in proposition 2 by transforming the external universal quantifier into the negation of an existential one, and we consider the negation of the resulting formula. We call the resulting formula *inverted encoding*.

$$\neg\exists\vec{T}_c.(\Gamma(\vec{Y}_u) \rightarrow \Psi(\vec{T}_c, \vec{Y}_u)) \tag{6}$$

If this formula is unsatisfiable, then the problem is weakly controllable, while if it is satisfiable, then the problem is not weakly controllable. Note that in equation (6) we dropped the outermost $\neg\exists\vec{Y}_u$ as any SMT problem is inherently an existential quantification and we consider the negation by reversing the interpretation of the result. Intuitively, we are searching for an assignment to the uncontrollable time points that can violate the free constraints under any possible strategy (it is a winning strategy for the environment). In fact, if the formula is satisfiable, each model corresponds to a situation for which no weak strategy to schedule the controllable time points exists. Therefore, differently from equation (5), this encoding is also helpful for debugging a non-weakly controllable problem.

A further improvement can be achieved by limiting as much as possible the scope of the existential quantifier. To this extent, we push the existential quantifier over the implication, and thus the quantification is limited to the free constraints only (ref. as *assumption-extraction* encoding):

$$\Gamma(\vec{Y}_u) \wedge \neg\exists\vec{T}_c.\Psi(\vec{T}_c, \vec{Y}_u). \tag{7}$$

Weak Strategies. We now consider the problem of actually executing a control strategy that is associated with a given weakly controllable TNU. A TNU is a modeling framework that represents a set of assumptions over the environment and imposes a set of requirements to be fulfilled. We consider the use-case in which a strategy for scheduling the controllable time points is repeatedly executed by reading the inputs from the environment in the form of a situation. Such a situation is generated by reading the parameters on which the uncontrollable durations depends, by means of appropriate sensors or estimators. The strategy computes an assignment to the controllable time points that fulfills the problem constraints and is then deployed to an actuator for execution.

The problem we tackle here is to automatically synthesize such a strategy: we discuss two approaches, namely *implicitly* and *explicit* strategies.

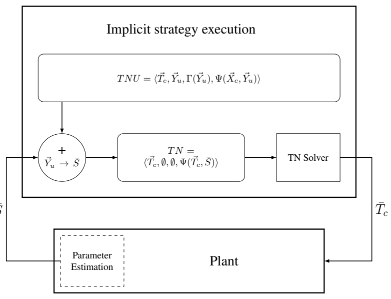

Implicit strategy execution
$TNU = \langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{X}_c, \vec{Y}_u) \rangle$
+
$\vec{Y}_u \rightarrow \bar{S}$
$TN = \langle \vec{T}_c, \emptyset, \emptyset, \Psi(\vec{T}_c, \bar{S}) \rangle$
TN Solver
$\bar{S}$
$\bar{T}_c$
Parameter Estimation
Plant

Fig. 6. Schematic view of implicit strategy mechanism. The strategy is repeatedly executed once a situation is obtained by estimating the relevant parameters in the Plant. The output of the strategy is a controllable schedule (i.e. an assignment $\bar{T}_c$ to all the controllable time points). The implicit strategy works by "projecting away" the uncertainty in the TNU: the uncontrollable durations $\vec{Y}_u$ are substituted with the actual values of the situation $\bar{S}$. Then, a TN is obtained and is solved using a TN solver, yielding the assignment ($\bar{T}_c$) to the controllable time points.

A way of obtaining a strategy for a weakly controllable TNU is given by definition 3 and depicted in figure 6: when a situation $\bar{S}$ is read[5], we eliminate the uncertainty by substituting the uncontrollable duration variables in the TNU formulation with the values obtained from the situation (obtaining a TN that is the projection of the TNU). Then, we solve the resulting temporal problem, that is now without uncertainty, and return the assignment to the controllable time points (indicated as $\bar{T}_c$) for execution. Formally, given the encoding of a TNU $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ and an assignment to all the uncontrollable durations $\bar{S}$ fulfilling $\Gamma(\bar{S})$ (a situation), we can find an assignment to the controllable variables $\vec{T}_c$ by finding a model for the formula $\Psi(\vec{T}_c, \bar{S})$. This strategy requires a solver to be executed once the situation $\bar{S}$ is known. In practice, we can implement this idea using any SMT solver by searching for a model for $\Psi(\vec{T}_c, \bar{S})$. However, this approach (called IMPLICIT-SMT) requires one to solve a separate SMT problem for each situation.

The main drawback of the implicit approach is the requirement of on-line reasoning. In fact, once the situation is known, a solver is invoked to discover the assignment for the controllable time points. Solving the TN resulting from the projection of a TNU is hard in general. If the problem belongs to the STNU problem class the resulting STN can be solved in polynomial time, but for the general case of DTNU, the projection results in a DTN that is, in general, NP-hard [53]. In

[5] $\bar{S}$ is a vector of $|\vec{Y}_u|$ rational numbers, one for each uncontrollable duration.

addition, having a solver as part of the run-time may require much more expensive platforms.

We avoid the burden of on-line reasoning by providing techniques for the synthesis of explicit strategies as functions that are simple and fast to execute. Consider the formalization in proposition 2. Interestingly, we can apply skolemization [35], thus replacing the existential quantifier by means of a fresh function symbol: i.e.

$$\forall \vec{Y}_u . \Gamma(\vec{Y}_u) \rightarrow \Psi(f(\vec{Y}_u), \vec{Y}_u). \qquad (8)$$

We transform the inner existential quantifier into a function f that models the weak strategy for the problem. In fact, in equation (8), the interpretation of the function f is exactly a strategy that solves the problem.

In the following, we focus on two types of explicit strategies: *linear strategies*, where each controllable variable is computed as a linear combination of the uncontrollable durations; and *piecewise-linear strategies*, where different linear strategies are executed depending on the input situation. From here on, we assume the encoding $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ of a TNU is given. In general, a weak strategy is a function that maps each assignment to uncontrollable durations satisfying $\Gamma(\vec{Y}_u)$ (i.e. each situation) into an assignment to the controllable time points, such that all the free constraints are satisfied.

Definition 7. *A weak strategy for a TNU is a function $f : \mathbb{R}^{|\vec{Y}_u|} \rightarrow \mathbb{R}^{|\vec{T}_c|}$ defined for every point $\vec{Y}_u$ in $\Gamma(\vec{Y}_u)$ and such that $\Psi(f(\vec{Y}_u), \vec{Y}_u)$ holds for every $\vec{Y}_u$ in $\Gamma(\vec{Y}_u)$.*

Note that, this definition does not impose any constraint (e.g. linearity, continuity) on f other than being a function. We modeled a weak strategy as a single function $f : \mathbb{R}^{|\vec{Y}_u|} \rightarrow \mathbb{R}^{|\vec{T}_c|}$, but we can equivalently consider a set of functions $f_1, \ldots, f_{|\vec{T}_c|}$ each computing a schedule for a single controllable time point given the situation. The two formalizations are equivalent because if there exists a unique function f, we can obtain the set of function by projection of f and vice-versa.

A *linear strategy* is such that the value of every controllable time point is obtained as a linear combination of $\vec{Y}_u$. Let $n \doteq |\vec{T}_c|$ and $m \doteq |T_u|$. A linear strategy can be represented as a matrix A of real coefficients of size $n \times m$ and a vector $\vec{c}$ of size n. Every controllable variable is scheduled according to a linear function of the uncontrollable durations. The strategy $f(\vec{Y}_u)$ can be then expressed as $A \cdot \vec{Y}_u + \vec{c}$ in which each $b_i \in \vec{T}_c$ can be computed as $A_{i,1} y_{A_e} + \ldots + A_{i,m} y_m + c_i$. Therefore, the matrix A must have one column for every duration and the vector $\vec{c}$ contains the constant additive terms. The problem of synthesizing a linear strategy is then equivalent to the problem of finding a suitable matrix A and vector $\vec{c}$.

A more general form of strategy is the *piecewise-linear strategy*, that is the composition of a finite number of linear strategies. A piecewise-linear strategy is defined by cases over a finite partition of the situations (a partition of the region represented by $\Gamma(\vec{Y}_u)$). For each case we have a linear strategy that is a valid weak strategy for that subset of the situations. We can compose these linear strategies by first checking in which element of the partition the observed situation belongs, and then applying the corresponding linear strategy. In this setting, a linear strategy is a particular case of a piecewise-linear strategy in which we have a partition of cardinality one.

Definition 8. *A* piecewise-linear strategy *is a function*

$$f(\vec{Y}_u) \doteq \begin{cases} f^1(\vec{Y}_u) & \textit{if } \eta^1(\vec{Y}_u) \\ f^2(\vec{Y}_u) & \textit{else if } \eta^2(\vec{Y}_u) \\ \ldots & \\ f^k(\vec{Y}_u) & \textit{else if } \eta^k(\vec{Y}_u) \end{cases}$$

where each f^i is a linear strategy and $\eta^i(\vec{Y}_u)$ are subregions of $\Gamma(\vec{Y}_u)$ such that $\Gamma(\vec{Y}_u) \subseteq (\bigcup_{i=1}^{k} \eta^i(\vec{Y}_u))$.

Linear strategies are very useful in practice: they are compact to represent and easy to evaluate. In fact, a linear strategy can be represented using just a matrix and a vector of real numbers; moreover, given an assignment to the uncontrollable duration, we can compute the schedule for the controllable variables by means of a single matrix multiplication. In general, unfortunately, a weakly controllable TNU is not guaranteed to have a linear strategy. In fact, even the STNU class of problems is not guaranteed to admit such a strategy for every weakly controllable instance. The following theorem (proven in [16]) states that there exists a weakly controllable STNU without any linear strategy.

Theorem 1. *There exists an STNU that is weakly controllable and does not have any linear strategy.*

On the contrary, we proved that a piecewise-linear strategy always exists for any weakly controllable TNU.

	Strategy Type	
	Linear	Piecewise-Linear
Convex (STNU)	VertexEncoding IncrementalWeakening	SimplexesDecomposition LazyExpansion
Disjunctive (DTNU)	NRA Encoding	SkinCrawler ConvexRegionEnumerator

Table 2
Overview of the developed algorithms.

Theorem 2. *For any given TNU P, if P is weakly controllable, then P admits a piecewise-linear strategy.*

We are interested in generating strategies that can be efficiently executed once the situation is known. Given this requirement, linear strategies are very helpful, because they are compact (the size is quadratic in the number of time points) and can be executed by performing a linear computation in the size of the strategy. Piecewise-linear strategies are also helpful because they can be executed in linear time in the size of the strategy as they require only a case switch before applying the linear executor.

Strategy Extraction. We now address the problem of synthesizing weak strategies. We classify the problem along two dimensions, distinguishing between (i) convex (STNU) vs. disjunctive (DTNU) temporal problems and (ii) linear vs. piecewise-linear strategies. Table 2 summarizes this classification and indicates the algorithms we developed for each problem class.

All the algorithms assume that the given problem is weakly controllable, but it is not known in advance whether the problem admits a linear strategy. Thus, the algorithms listed in the "Linear" column of table 2 return $\perp$ in case no linear strategy exists. The others are guaranteed to find a piecewise-linear strategy.

We do not describe the algorithms in detail here, we only give an overview of the ideas; all the details and the pseudo-codes for the algorithms are thoroughly presented in [16].

The algorithms for extracting linear strategies work by trying to synthesize the matrix of coefficients A and the bias vector $\vec{c}$. VertexEncoding encodes the problem for an STNU an a $\mathcal{QF\_LRA}$ formula using an SMT variable for each element of A and $\vec{c}$ and imposing an exponential number of constraints, while the NRA Encoding deals with DTNUs using a quantified non-linear formula using an SMT variable for each element of A and $\vec{c}$ and an SMT variable also for the elements of $\vec{Y}_u$. These approaches migth fail to build a linear strategy because it might not exists even for weakly controllable networks. The IncrementalWeakening algorithm is an optimization of the VertexEncoding that incrementally builds the strategy trying to limit the number of variables observed by the strategy.

The approaches for building piecewise-linear strategies work by splitting the region of the uncontrollables and by applying linear strategies to the obtained regions. SimplexesDecomposition decomposes the uncontrollable space in simplexes and for each of them a linear strategy is constructed; we proved that such a linear strategy always exists in [16]. LazyExpansion optimizes this approach by extending a linear strategy obtained by the SimplexesDecomposition approach even outside the original simplex, possibly limiting the number of iterations needed to cover the uncontrollable space. SkinCrawler constructs a piecewise-linear strategy for a DTNU by considering the skin of the polyhedra that are the geometric interpretation of the free constraints. Finally, the ConvexRegionEnumerator algorithm decomposes a DTNU in a number of STNU and for each of them it applies a piecewise-linear strategy construction. We highlight that these algorithms provide a correct weak strategy for the problem at hand, but no effort is put into minimizing the size of the strategy itself. This optimization topic is a clear objective for future research that is discusse din more detail in section 8.

Experimental Results. In order to empirically test the effectiveness of the proposed approaches, we implemented a tool for deciding weak controllability and synthesizing weak strategies for a TNU. Our tool reads a TNU problem, and applies our portfolio of encodings and algorithms. The tool can synthesize explicit strategies as C++ functions (taking in input a situation), that can be compiled and linked in any program.

We tested the decision problem encoding over a set of 2442 randomly generated DTNU, TCSNU and STNU instances, with a number of time points ranging from 6 to 20000. For the evaluation of strategy-extraction techniques, we used 1354 weakly controllable STNU benchmarks and 2112 weakly controllable DTNU instances ranging from 4 to 50 time points.

We remark that, as far as our knowledge is concerned, there are no competitor tools or solvers able to deal with the weak controllability decision problem, nor with the synthesis of a weak strategy. Thus, in the experimental evaluation, we do not compare with any other tool or approach.

The results of checking the decision problem over the set of TNUs are plotted in figure 7. The cactus

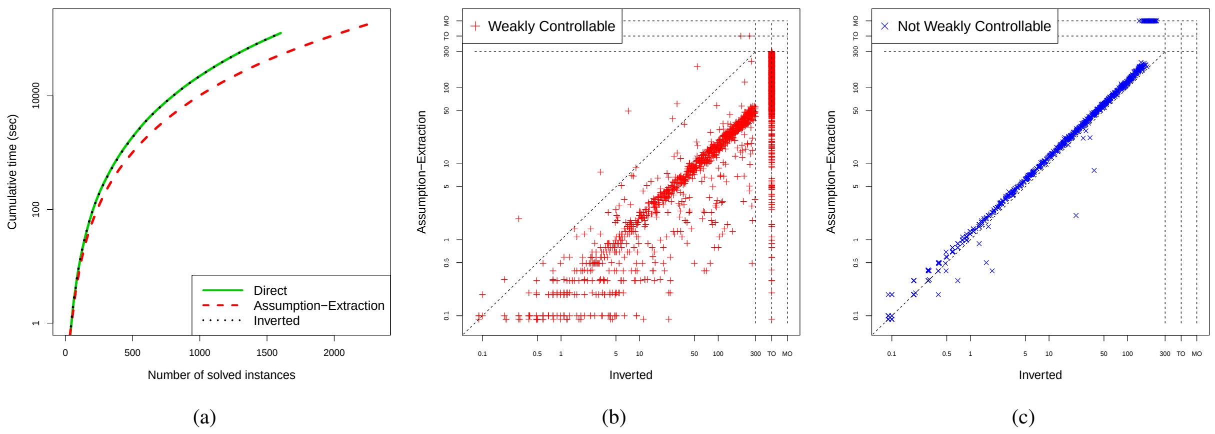


Fig. 7. Results for the decision problem encodings solved using the Z3 SMT solver. Figure (a) reports the cumulative time (in logarithmic scale) cactus plot; Figures (b) and (c) show the scatter plots of INVERTED vs. ASSUMPTION-EXTRACTION encodings divided in weakly controllable, and not weakly controllable, respectively. The TO line denotes the instances that reached the time out, while MO indicates instances that hit the memory limit.

plot (a) reports, in the horizontal axis, the number of solved instances and, on the vertical axis, the cumulative time, in logarithmic scale, taken by the SMT solver for each encoding. The figure highlights the fact that Z3 performs much better when the ASSUMPTION-EXTRACTION encoding of the problem is considered: in fact, this approach is able to solve, in less time, a higher number of instances with respect to the INVERTED and DIRECT encodings. In figures 7b and 7c, we reported the scatter plots comparing the performances of the ASSUMPTION-EXTRACTION with the INVERTED encodings, distinguishing between weakly controllable and non weakly controllable instances. We note that, in the weakly controllable case, the ASSUMPTION-EXTRACTION encoding outperforms the INVERTED encoding in most of the benchmarks. For non weakly controllable instances, the two encodings perform similarly in terms of speed. However, the INVERTED encoding is able to solve 86 instances that are unsolvable by the ASSUMPTION-EXTRACTION encoding due to the imposed memory limit.

The evaluation results for STNU strategy-extraction techniques are reported in figure 8. The plot considers only those benchmarks that admit a linear strategy, and compares the four different approaches. The plot clearly shows that for linear strategies, the INCREMENTALWEAKENING approach outperforms all the others. The SIMPLEXESDECOMPOSITION method quickly explodes due to the factorial complexity of simplexes enumeration. Although the techniques for piecewise-linear strategy extraction are penalized as

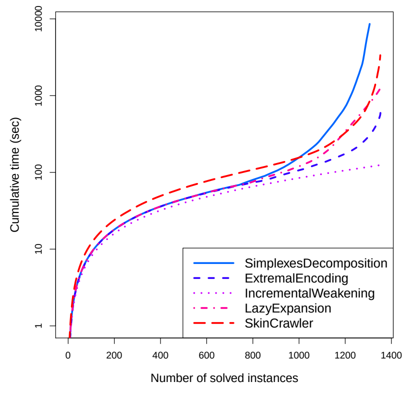


Fig. 8. Results for STNU linear strategy extraction problem: we plotted the cumulative cactus plot of the strategy extraction time for the different algorithms we propose.

they are strictly more general than the others, the plot shows that LAZYEXPANSION approach is much faster than the SIMPLEXESDECOMPOSITION.

In figure 9 we plotted the number of “pieces” of the strategies for the LAZYEXPANSION and SIMPLEXES-DECOMPOSITION methods. The plot shows that, although for small problems the LAZYEXPANSION approach generates additional, unneeded “pieces”, when the problem size increases the number of “pieces” identified by the LAZYEXPANSION method is much smaller than for the SIMPLEXESDECOMPOSITION one. In general, the LAZYEXPANSION approach has a huge gain in performance and in strategy size.

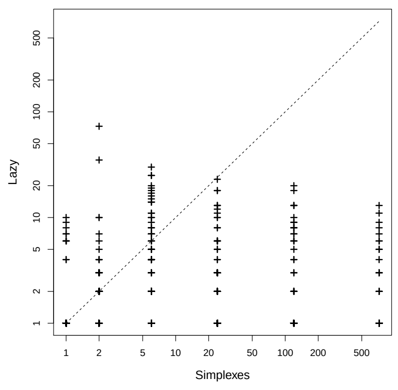


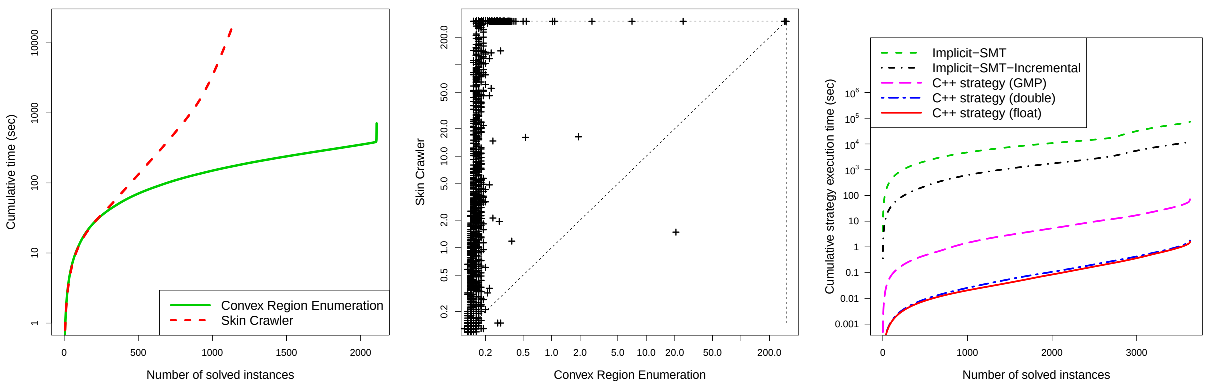
Fig. 9. Results for STNU linear strategy extraction problem: we compared the number of pieces for piecewise-linear algorithms expressed as the number of split regions.

In figure 10 we report the results on the DTNU problem class. The plots show that the ConvexRegionEnumerator algorithm performs better than the SkinCrawler one. This is because of two main reasons. First, the SkinCrawler approach solves a costly minimization problem and has to traverse all the faces of the space of free constraints while the ConvexRegionEnumerator algorithm applies the cheap LazyExpansion approach to each convex region that is generated by a single call to the SMT solver. Second, the linear strategy generated by the LazyExpansion approach is generalized and applied wherever possible, therefore if the problem allows for a linear strategy the ConvexRegionEnumerator algorithm is able to quickly synthesize it, while the SkinCrawler has to enumerate enough faces to cover the entire uncontrollable space.

There is an interesting peak on the rightmost part of the ConvexRegionEnumerator curve in the plot. This is due to a particular instance that is solved in 287.28 seconds generating a strategy with 3750 pieces (thus using the same number of iterations to terminate). This is one example in which the splitting done by the LazyExpansion approach gets lost in splitting the uncontrollable space in simplexes.

We proposed a number of approaches to synthesize weak strategies arguing that their execution is practically more efficient than solving the individual problems without uncertainty obtained by projecting the uncertainty away. We provide experimental evidence supporting this claim on a number of STNU and DTNU instances. For each TNU in our benchmark set, we randomly generated 1000 situations, represented as complete assignments for the uncontrollable durations. We implemented two variations of the implicit approach using the MathSAT5 SMT solver. In addition, we considered three ways to compile in machine code the problem-specific strategies generated by our algorithms. We translated the linear or piecewise-linear strategy synthesized by any of our algorithms into C++ code. The translation for linear strategies is straightforward: we create a function that takes in input a numeric value for each uncontrollable duration and we compute the output of the strategy. Given a piecewise-linear strategy, we translate it using a sequence of `if` statements, one for each piece. The condition of each `if` is the transposition in C++ syntax of the piece condition. Each conditional statement returns the value computed by the translation of the linear strategy relative to the particular piece. We used three different datatypes to represent numeric values and perform the arithmetic operations. In particular, we used the (finite-precision) C++ `float` and `double` and the GNU-MP library for arbitrary precision arithmetic [28]. The `float` and `double` datatypes are a finite-precision representation of rational numbers. As such, they suffer from both numeric stability and rounding problems that may, in principle, cause unsoundness in the strategy output. On the other hand, GNU-MP is the same library employed by the MathSAT5 SMT solver and does not suffer from any kind of numeric stability or rounding problems.

Figure 10c shows the results of the comparison: in our experiments the explicit strategies outperform projection-based implicit strategies. Implicit-SMT-Incremental performs better than Implicit-SMT, thanks to the incrementality feature of the SMT solver, but the explicit strategies bring a significant speedup on all the instances. Arbitrary-precision arithmetic (that fairly compares with the SMT precision) outperforms projection-based techniques by two orders of magnitude. The compiled strategies with native C++ datatypes perform even better, but the numerical stability problems can, in principle, lead to unsound results. We checked the output of each technique on each situation in order to assess the soundness, but all the results were correct. Nevertheless, studying under which condition we can guarantee that such finite-precision implementations are correct is subject of future work. We highlight that the compilations using native C++ datatypes can be translated to Boolean circuits, and this opens for the possibility of creating very efficient hardware implementations of these strategies.



(a) (b) (c)

Fig. 10. Results for strategy extraction problem in the DTNU problem class (a-b). In (a) we plotted the cumulative cactus plot of the solving time for the ConvexRegionEnumerator and the SkinCrawler algorithms while (b) is a scatter plot of the data. Results for strategy execution: for each problem, the generated strategy is executed on 1000 randomly generated situations (c). the plot considers all the STNU and DTNU randomly generated problems. The cactus plot shows the number of solved instances on the x axis and the accumulated time to solve them in the y axis.

6. Dynamic Controllability

The last kind of query that can be addressed given a TNU is dynamic controllability. Dynamic controllability is concerned with the existence of a strategy for executing the controllable time points that depends only on past observations of the outcomes of uncontrollable durations, and that guarantees that all relevant constraints will be satisfied no matter how the durations of the contingent links turn out. Essentially, a TNU is dynamically controllable if there exists a weak strategy that, in order to decide a controllable time point at time k, does not depend on any observation past time k. In order to keep the discussion limited in scope, we do not provide a formal definition of dynamic controllability here: such a definition needs the introduction of several formal concepts. The interested reader can see our previous work in [12] for a thorough formalization of the dynamic controllability for the DTNU case.

Polynomial algorithms to check the dynamic controllability of STNUs [33,43,44,45] and run-time algorithms for generating an execution strategy in real-time [31,32] have been presented in the literature. Although STNUs have been successful in some domains, many other domains require a richer set of constraints and features. Disjunctive constraints often arise in practice, for example, when two activities cannot be done simultaneously, but a dynamic controllability checking algorithm has only been presented for a subclass of DTNUs [55].

In this section, we present the work we contributed in the field of dynamic controllability. The contents of this section are derived from our previous papers [11,12,13,19]. We first define a syntax for expressing dynamic strategies (that is shown to be sufficient for any DTNU in [19]) and propose an algorithm to validate a given strategy against a TNU. Second, we present a novel approach for checking the dynamic controllability of DTNUs by translating the dynamic controllability problem into a reachability game on a Timed Game Automaton (TGA) [39]. The reachability game can be solved using off-the-shelf software that is able to synthesize a viable execution strategy or determine that no such strategy exists [5]. This results in the first sound-and-complete checking algorithm for the dynamic controllability of DTNUs. The encoding of such networks into TGA highlights important theoretical relationships between the different kinds of temporal reasoning frameworks and the TGA framework.

Third, we exploit the ideas behind the TGA encoding to develop a dedicated solving algorithm for the DTNU problem class. The algorithm is able to synthesize a strategy in form of an executable program. Finally, we present a comprehensive experimental evaluation of the proposed approaches.

Strategy Language. In the practical sense, the concept of dynamic strategy for a TNU is argument of discussion. Existing approaches that solve the dynamic controllability problem, produce strategies expressed as constraint networks that need to be scheduled at run-time by an executor. These networks encode a possibly infinite number of executions, but require a constraint solving algorithm running on-line with the system. This may or may not be acceptable, depending on the application at hand.

Some works aimed at reducing the amount of online reasoning as much as possible [44]. Here, we want to follow the idea we introduced for weak controllability: we want to automatically synthesize executable strategies. We define the following language to compactly express strategies in a readily executable form.

Definition 9 (Strategy Syntax)**.** *A strategy σ is recursively defined as follows.*

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

We defined a single wait operator that waits for a condition ψ to become true. Conditions are expressed as time regions defined over a set of clocks $\vec{T} \doteq \{\overline{x} \mid x \in \mathcal{T}\}$. Those are borrowed from the literature of Timed Automata [2,39]. For the sake of this paper, it suffices to say that given a set of clock variables X, we call "time region" any formula expressed by the following grammar:

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

where $\bowtie \in \{<, \leq, >, \geq, =\}$. We use a clock for each time point (controllable or uncontrollable). All the clocks start from time 0 and evolve as time passes. We encode the constraint that a clock is reset to 0 when the corresponding time point is executed. In this way, each clock measures the time passed since the corresponding 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.

A wait action 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. For example, if while executing a strategy $w(\psi, e_1 : \sigma_1, e_2 : \sigma_2, \dashv : \sigma_\dashv)$ the time point e_2 is observed, the wait is terminated and σ_2 is immediately executed.

In addition to wait statements, we also have the concatenation $(s(b); \sigma')$ construct that prescribes to immediately schedule the b time point (we assume no time elapses), and then proceed with the strategy σ'. The `noop` operator is a terminator signaling that the strategy is completed.

Theorem 3 (Sufficient Syntax)**.** *A DTNU is dynamically controllable if and only if it admits a solution strategy expressible as per definition 9.*

This strategy syntax 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. In order to simplify the exposition, 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 `noop` is a way of scheduling the time points in a specific order.

A strategy σ needs two characteristics for being a solution to the dynamic controllability problem: *dynamicity* and *validity*, defined as follows.

A strategy 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 the strategy syntax in definition 9, dynamicity can be checked syntactically: 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 time region ψ, as follows.

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

Proposition 3. *A strategy σ is dynamic if and only if $dyn(\emptyset, \sigma) = \top$.*

Intuitively, P keeps track of the time points that happened in the branch under analysis, and the check ensures that no time point outside P is used as a wait condition.

Strategy Validation. We now focus on the problem of validating a given strategy using the immediate-reaction semantics. We first present a search space that encodes every possible strategy for a given DTNU. 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 $Init \doteq \langle \emptyset, \bot, \top, \bot\rangle$. The transition relation defining the allowed moves in the space is E, defined as follows:

- $\langle P, \bot, \phi, \bot\rangle \xrightarrow{s(b)} \langle P \cup \{b\}, \bot, \rho(\phi, \overline{b}), \bot\rangle$ 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 the immediate start of 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.

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 resulting from starting a wait for condition ψ starting from any assignment that is compatible with condition ϕ.

For each uncontrollable time point e, we define the $uc(e)$ time region as $\bigvee_{\langle l,u\rangle \in \mathcal{B}} \overline{\alpha(e)} \geq l \wedge \overline{\alpha(e)} \leq u$ 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. We remark that the time region only depend on the clock corresponding to the activation time point $\alpha(e)$ because clocks measure the time since the corresponding time clock happened. In fact, this is a transliteration of a contingent link into a time region. This search space directly mimics the structure of our strategies, but 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.

The procedure for validating a strategy is reported in [19]: the algorithm navigates the search space $\mathbb{S}$, by 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 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 and branching to explore all possible uncontrollable outcomes in and-nodes. The algorithm takes a number of steps that is linear in the size of the strategy because at each step the strategy gets shortened.

Proposition 4. *The validation procedure is sound and complete for the immediate-reaction semantics.*

Strategy Synthesis. Checking dynamic controllability and synthesizing dynamic strategies for the DTNU problem class were open problems in the literature. We addressed this issue by providing a general schema to approach the dynamic controllability problem for the whole DTNU network class. The idea is to reduce the problem to a Reachability Game on a Timed Game Automaton (TGA) [39] obtained via a *linear* encoding procedure. Since the reachability problem for TGA is decidable and algorithms have been developed to solve this problem, this reduction constitutes a viable and novel solution approach for the open problem of dynamic controllability of DTNU. Moreover, this technique has been extended to also deal with discrete non-determinism [12].

We do not report the formal encoding here (the details have been published in [11] and [12]), for the sake of this paper it suffices to say that the encoding is linear in the size of the TNU and that the resulting TGA satisfied the reachability property if and only if the TNU is dynamically controllable. Moreover, each TGA controller guaranteeing the reachability property yields a valid dynamic strategy.

In addition to the TGA encoding, we developed a direct synthesis technique for DTNU. Differently from the TGA encoding, we directly synthesize dynamic strategies that are valid by construction without resorting to external TGA solvers. The details on this synthesis technique can be found in [19]. Intuitively, we combined our TGA encoding with the TGA solving algorithm in [9] removing useless checks and generating the search space on-demand. Moreover, we exploited the structure of the TNU to guide the search in the space of the reachability game and we devised an SMT-based algorithm to effectively early-prune the search. Finally, we devised two flavors of the algorithm one that considers different orderings as different TGA locations and one that disregards the ordering. The importance of the algorithm is twofold: first it performs much better than state-of-the-art TGA solvers applied to our TGA encoding, second it directly outputs an executable dynamic strategy using the syntax provided in definition 9.

Experimental Results. We evaluate the merits of the TGA encoding and the direct synthesis technique on a number of DTNU benchmarks, both considering the immediate-reaction semantics.

We implemented the DTNU-to-TGA using an automated encoder that takes in input a DTNU and outputs a TGA in the input language of the state-of-the-art TGA solver Uppaal-TIGA [5]. We implemented both the flavors of the DTNU encodings discussed in [11]. In the following, we refer to the DNF encoding as TIGA-DNF and to the NNF encoding as TIGA-NNF. We also implemented the validation and synthesis algorithms in a tool called PyDc. We analyze four versions of the 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 is the same we used for weak controllability strategy synthesis.

The results are shown in figure 11. We first notice that the direct synthesis techniques are vastly superior to the TGA-based approaches as shown in the cactus plot of figure 11a. The direct synthesis algorithm is able to solve 2543 instances, while the best TGA based approach can only solve 799 instances. We observe run-times differences between the two approaches of up to three orders of magnitude, as shown in figure 11b.

The cactus plot in figure 11a, also shows that the SMT-based pruning yields a significant performance boost, both for the unordered case (from 1531 to 2289 solved instances), and for the ordered case (from 1552 to 2543). Nonetheless we observe how the pruning is much more effective for the ordered case, thanks to the additional strength of the pruning constraints.

Finally, the scatter plot in figure 11c shows that the ordered case is almost always superior to the unordered one when the SMT pruning is enabled. 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 than in the unordered case.

7. Related Work

In the literature, many works focused on the STNU framework to model and reason on temporal uncertainty, with very few approaches being able to deal with TCSNU or DTNUs. In this section we provide an overview of the state-of-the-art concerning temporal networks with uncertainty.

STNUs have been successfully used in many application contexts [20,26,46]; in general, they are useful to model a fixed set of activities or tasks (some of which having uncontrollable duration) subject to constraints. For example, STNUs can be used to represent temporal plans in AI planning.

The STNU framework has been presented in [59], where the authors prove that the strong controllability problem is tractable and give an algorithm for reducing any given STNU to an STN having only controllable time points, in such a way that any consistent schedule for the STN is a solution for the strong controllability problem. This approach is based on removing all the uncontrollable time points and all the contingent links, and to substitute each occurrence of an uncontrollable time points in the free constraints according to a substitution table. In our SMT-based approach we reconstruct this result when using the eager for-all elimination encoding extending it to the case of TCSNU. The strong controllability problem for DTNU has been addressed in [50]. In this work, we developed several encodings into the SMT framework that are experimentally shown to be much more efficient than the theoretical technique pioneered in [50].

Concerning weak controllability, [59] gives a co-NP algorithm for deciding the problem for STNUs, but no synthesis algorithm for weak strategies is present in the

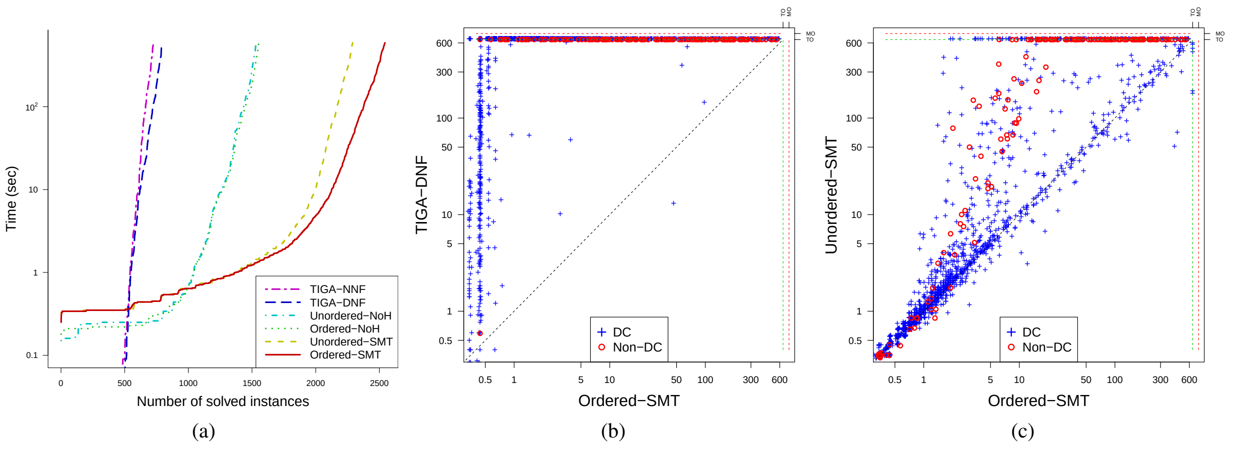


Fig. 11. Results for the experimental evaluation. The logarithmic scale cactus plot in (a) shows the different approaches (with and without SMT pruning) against the theoretical encoding solved by the Uppaal-TIGA TGA solver. The scatters compare the ordered SMT approach with the theoretical encoding (b) and with the unordered solver (c).

literature. We extended these results giving constructive algorithm that are able to produce weak strategies and we also tackled the full DTNU problem class.

Dynamic controllability has been widely studied for the STNU class. Starting from the original formulation in [59] that proposed an exponential game-based solution, Morris, Muscettola and Vidal showed a pseudo-polynomial algorithm in [46]. Then, Morris and Muscettola refined the idea into a polynomial algorithm [45] that has been further optimized by Morris in [43] obtaining an $O(n^4)$ algorithm. The semantics of dynamic controllability has been studied by Hunsberger [29] that proposed a view in which the solution strategy cannot react to uncontrollable decisions in zero time, but needs a non-null time to elaborate and react to the observation. These works are based on the definition of enriched networks of constraints that are propagated till a fixed point is reached following carefully designed rules. If the network does not enter in an inconsistent state, then the problem is provably dynamically controllable but no strategy synthesis is performed by these algorithms. [43] and [33] show how to write a reasoning algorithm that takes in input a propagated network and decides when to execute controllable time points at run-time. The problem with such algorithms is that they propagate constraints at run-time each time a time point is scheduled or observed. The complexity of the technique is polynomial. Finally, we mention [44] in which Morris proposed an algorithm for transforming a dynamically controllable STNU in an executable form (called dispatchable) that can be used for on-line execution with a minimal reasoning effort. No algorithms or techniques exist in the literature for solving the dynamic controllability problem of the DTNU problem class. [55] presents an algorithm for dynamic controllability that is limited to the TCSNU class and is based on a Meta-CSP exploration and a decision procedure for the weak controllability of DTNUs. Both these approaches are mainly theoretical and are limited to the decision problem. This means that they are not concerned with the synthesis of a strategy (in our execution model, a plan) for scheduling the time points, but only in verifying that such a plan exists at all. In our approach, we followed the idea of exploiting an encoding in TGA pioneered by Vidal [56,57] by providing a polynomial sized encoding that also works for the DTNU case and by constructing a dedicated solving algorithm that improves the performance over a general-purpose TGA solver. Our approaches are constructive: they produce a closed-form execution strategy if they find that the input network is dynamically controllable. Finally, other approaches exploited the relation between TGA and dynamic controllability of temporal networks. In particular [49] and [40] studied this relationship in the context of flexible plans, while [1] explores the use of TGA for the problem of planning (not scheduling) using dynamic controllability.

8. Future Directions

In this paper, we presented our efforts pursuing a more comprehensive coverage of the problems arising in the context of temporal uncertainty for temporal network scheduling. We believe that the techniques we presented can serve in many practical applications and we actively work in pushing these techniques in research projects. However, in this work we focused on covering the different controllability queries for DTNs disregarding several orthogonal aspects that might be useful in practical applications. In this section, we elaborate on different directions to extend this work, motivating the expected impact and discussing possible ideas.

Discrete Non-Determinism. A natural extension is to adapt the techniques to work also for the non-deterministic case, i.e. when activities have unpredictable effects on the world and these effects can be used to express the constraints. An example formalism in this setting is the Conditional Temporal Network (CTN) [54] in which part of the constraints are activated or de-activated depending on a Boolean, non-deterministic run-time observation. With this respect, we started working [12] by combining the disjunctions of the DTNU framework with conditionals in the CTN formalism, obtaining the Conditional Disjunctive Temporal Problem with Uncertainty (CDTNU). We extended the DTNU-to-TGA approach for solving the dynamic controllability to this case. Dealing with strong controllability in CDTNU is straight-forward because we can project away the Boolean variables as explained in [54] effectively reducing the problem to the strong controllability of a linear-sized DTNU suitable for our techniques. Weak Controllability is an open problem when uncertainty is considered.

Optimality. All the approaches we presented are concerned in finding a solution for the controllability problems, in the from of a strong schedule or a strategy: no distinction is imposed between solutions. An important direction to explore is the optimization of solutions in order to find those that maximize some desired property. This could be very useful in practice: for example, one may look for the strategy that minimizes the time-span while respecting all the constraints, reducing the operational costs. For strong controllability we envisage the possibility of exploiting our encoding in $\mathcal{QF\_LRA}$ and apply an optimizing SMT solver (e.g. [52]). For the other controllability levels the problem is open and less obvious. In particular, we believe that the synthesis of optimal dynamic strategies could be extremely useful in practice. Finally, for the case of strategy synthesis there is a clear multi-objective direction of optimization: on one hand one may want to optimize the performance of the strategy in terms of make-span or by prioritizing activities to be executed, on the other hand one may want to minimize the size of the strategy itself yielding the minimal number of cases to be considered and the minimal memory footprint.

Resources. In the scheduling literature resource constraints are considered in many works. Temporal networks cannot directly express complex resource constraints like for example the consumption of fuel by one activity or the workers needed to perform a task. With this respect, some authors managed to encode particular kind of resources in TNU [37], and an extension of the TN framework for resources has been presented [36]. Extending our scheduling techniques for dealing with resources is possible, but presents some challenges: in some parts of our reasoning we exploit the fact that temporal constraints are limited to a specific form in $\mathcal{QF\_RDL}$, but we often use very expressive SMT theories, such as $\mathcal{QF\_LRA}$, that can accommodate at least some classes of resources. For example, dealing with linear-continuous resources in the context of strong controllability is possible exploiting our encoding and adding the relevant resource constraints. However, a dedicated study is in order to understand the exact class of resources that can be addressed. Moreover, extending TNU adding resources can be done in different ways (e.g. only activity durations can be uncertain vs. resources consumption and production can be uncertain too) so we think that some work is definitely need in this respect to define a proper modeling framework and to address the different possible queries.

Partial Observability. It would be interesting to explore the middle-ground between strong and dynamic controllability and between dynamic controllability and weak controllability. In particular, an important issue arising when non-determinism of any kind is considered is partial observability: in a real system, not all the relevant variables are directly observed by appropriate sensors, either for cost and implementation reasons or simply because some quantities are not measurable in a sufficiently small time. Partial observability limits the entities that can be observed and forces the execution to take decision with incomplete information. In the TNU framework we could divide uncontrollable activities in observable and non-observable and provide only the observable durations to the executor. In this scenario, the dynamic and weak controllability problems get hybridized with strong controllability: in fact, some of the time points become not observable and the relative constraints must be solved in a "strong" way, the others can be observed and the strategy can rely on this information to make decisions. A first work in this direction is [6] in which the STNU framework is extended with partial observability and dynamic controllability is studied.

Planning. Finally, in this paper we focused on the problem of scheduling DTNUs, but the basic motivation behind this study is temporal planning with temporal uncertainty. In fact, just like TNs are used in temporal planning as the natural back-end to assess the consistency of partial plans, TNUs can be used to plan in domains where temporal uncertainty arises. Following this direction, we started a working on the problem of Strong temporal Planning with Duration Uncertaity that is the lifting of strong controllability to the planning level [18,42]. However, a lot of possible extensions are still not covered by the current state-of-the-art. the lifting of dynamic controllability to the planning case has been partially addressed by planners such as IxTeT [26] that generate plans in the form of a STNU that is guaranteed to be dynamic controllable, but the general problem of generating policies that are allowed to observe the duration of activities (possibly changing the course of actions) has never been studied. Another interesting issue is to consider partial observability at the planning level when temporal uncertainty is concerned.

9. Conclusion

In this paper, we presented a coherent vision on the research line on temporal networks with uncertainty we have been pursuing in the last years. Starting from the temporal network scheduling problem with uncertainty, we described the different queries arising in the context of temporal uncertainty and we presented dedicated techniques that are able to answer the controllability queries for the expressive disjunctive case. Finally, we also detailed a number of future directions to continue the line of research.

The PhD Thesis [41] from which this work is derived also discusses a detailed assessment of the state of the art concerning uncertainty in Planning and in Scheduling and also presents some results in which the technology explained in this paper is applied to the case of planning.

Acknowledgments

I am profoundly grateful to Alessandro Cimatti (advisor of the thesis) and Marco Roveri (co-advisor) for their patient guidance during the PhD. Moreover, I would like to thank David E. Smith, Minh Do and Jeremy Frank for the support and guidance during the internship period at NASA Ames. I am deeply grateful and honored for the "Marco Cadoli" award granted to this thesis by the Italian Association for Artificial Intelligence.

References

[1] Yasmina Abdeddaïm, Eugene Asarin, Matthieu Gallien, Félix Ingrand, Charles Lesire, and Mihaela Sighireanu. Planning robust temporal plans: A comparison between CBTP and TGA approaches. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 2–9, 2007.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In *ECP*, pages 97–108, 1999.

[4] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.

[5] Gerd Behrmann, AgnÃĺs Cougnard, Alexandre David, Emmanuel Fleury, KimG. Larsen, and Didier Lime. Uppaal-Tiga: Time for playing games! In *CAV*, pages 121–125. 2007.

[6] Arthur Bit-Monnot, Malik Ghallab, and Félix Ingrand. Which contingent events to observe for the dynamic controllability of a plan. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3038–3044, 2016.

[7] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In *CAV*, pages 299–303, 2008.

[8] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In *TACAS*, pages 150–153, 2010.

[9] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, pages 66–80, 2005.

[10] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *TACAS*, 2013.

[11] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *TIME*, pages 27–36, 2014.

[12] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Inf.*, 53(6-8):681–722, 2016.

[13] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, and Marco Roveri. Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In *AAAI*, pages 2242–2249, 2014.

[14] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving temporal problems using SMT: strong controllability. In *CP*, pages 248–264, 2012.

[15] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving temporal problems using SMT: weak controllability. In *AAAI*, 2012.

[16] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. An SMT-based approach to weak controllability for disjunctive temporal problems with uncertainty. *Artificial Intelligence*, 224:1–27, 2015.

[17] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving strong controllability of temporal problems with uncertainty using SMT. *Constraints*, 2015.

[18] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Strong temporal planning with uncontrollable durations: A state-space approach. In *AAAI*, pages 3254–3260, 2015.

[19] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *AAAI*, page to appear, 2016.

[20] Jing Cui, Peng Yu, Cheng Fang, Patrik Haslum, and Brian C. Williams. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *ICAPS*, pages 52–60, 2015.

[21] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of ACM*, 5(7):394–397, 1962.

[22] Thierry de la Tour. Minimizing the number of clauses by renaming. In Mark Stickel, editor, *CADE*, volume 449 of *LNCS*, pages 558–572. Springer, 1990.

[23] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

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

[25] Bruno Dutertre and Leonardo Mendonça de Moura. The Yices SMT solver. Tool paper at `http://yices.csl.sri.com/tool-paper.pdf`, 2006.

[26] Malik Ghallab and Hervé Laruelle. Representation and control in ixtet, a temporal planner. In *AIPS*, pages 61–67, 1994.

[27] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[28] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012. `http://gmplib.org/`.

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

[30] Luke Hunsberger. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *TIME*, pages 121–128, 2010.

[31] Luke Hunsberger. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *TIME*, pages 121–128, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[32] Luke Hunsberger. A faster execution algorithm for dynamically controllable stnus. In *TIME*, pages 26–33, 2013.

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

[34] Hyondeuk Kim, Fabio Somenzi, and HoonSang Jin. Efficient Term-ITE conversion for satisfiability modulo theories. In *SAT*, pages 195–208, 2009.

[35] Stephen C. Kleene. *Mathematical Logic*. J. Wiley & Sons, 1967.

[36] Philippe Laborie. Resource temporal networks: Definition and complexity. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 948–953, 2003.

[37] Michele Lombardi and Michela Milano. Constraint based scheduling to deal with uncertain durations and self-timed execution. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, pages 383–397, 2010.

[38] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *Computer Journal*, 36(5):450–462, 1993.

[39] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS*, pages 229–242, 1995.

[40] Marta Cialdea Mayer and Andrea Orlandini. An executable semantics of flexible plans in terms of timed game automata. In *22nd International Symposium on Temporal Representation and Reasoning, TIME 2015, Kassel, Germany, September 23-25, 2015*, pages 160–169, 2015.

[41] Andrea Micheli. *Planning and Scheduling in Temporally Uncertain Domains*. PhD thesis, University of Trento, 1 2016. Fulltext available at `http://www.mikand.net/thesis/`.

[42] Andrea Micheli, Minh Do, and David E. Smith. Compiling away uncertainty in strong temporal planning with uncontrollable durations. In *IJCAI*, 2015.

[43] Paul Morris. A structural characterization of temporal dynamic controllability. In *CP*, pages 375–389, 2006.

[44] Paul Morris. Dynamic controllability and dispatchability relationships. In Helmut Simonis, editor, *CPAIOR*, volume 8451 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2014.

[45] Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *AAAI*, pages 1193–1198, 2005.

[46] Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI*, pages 494–502, 2001.

[47] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *DAC*, pages 530–535, 2001.

[48] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[49] Andrea Orlandini, Alberto Finzi, Amedeo Cesta, and Simone Fratini. Tga-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*, pages 233–245, 2011.

[50] Bart Peintner, Kristen B. Venable, and Neil Yorke-Smith. Strong controllability of disjunctive temporal problems with uncertainty. In *CP*, pages 856–863, 2007.

[51] Alexander Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1998.

[52] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Transactions on Computational Logic*, 16(2):12:1–12:43, 2015.

[53] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[54] Iohannis Tsamardinos, Thierry Vidal, and Martha Pollack. Ctp: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003.

[55] Kristen Brent Venable, Michele Volpato, Bart Peintner, and Neil Yorke-Smith. Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *ICAPS - COPLAS Workshop*, pages 50–59, 2010.

[56] Thierry Vidal. Controllability characterization and checking in contingent temporal constraint networks. In *KR*, pages 559–570, 2000.

[57] Thierry Vidal. A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pages 395–402, 2000.

[58] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[59] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.