Timeline-Based Planning over Dense Temporal Domains with Trigger-less Rules is NP-Complete

Timeline-Based Planning over Dense Temporal Domains with Trigger-less Rules is NP-Complete*

Laura Bozzelli[1], Alberto Molinari[2], Angelo Montanari[2], Adriano Peron[1], and Gerhard Woeginger[3]

[1] University of Napoli "Federico II", Napoli, Italy
lr.bozzelli@gmail.com, adrperon@unina.it
[2] University of Udine, Udine, Italy
molinari.alberto@gmail.com, angelo.montanari@uniud.it
[3] RWTH-Aachen University, Aachen, Germany
woeginger@algo.rwth-aachen.de

Abstract. In timeline-based planning—an approach which is more declarative than standard, action-based planning—the domain is described by a finite set of independent, but interacting, state variables. The temporal behavior of each variable is governed by a transition function, and the sequence of values it takes over time is represented by a timeline. A set of temporal constraints, called synchronization rules, impose suitable conditions on the evolution of the values of variables.
The temporal domain is commonly assumed to be discrete, and the dense case is dealt with by introducing an artificial discretization. Here, we address the problem of timeline-based planning over dense temporal domains, without discretizing them. However, since the unrestricted version of the problem has been recently proved to be *undecidable*, we focus on the case in which all synchronization rules are trigger-less, and prove its **NP**-completeness.

Keywords: Timeline-Based Planning · Computational Complexity · Timed Automata · NP-completeness.

1 Introduction

In this paper, we study the problem of *timeline-based planning*. Unlike the standard action-based planning, that focuses on the series of actions an agent has to perform to reach a goal from the initial state of the world, timeline-based one describes in a more declarative fashion what has to happen in order to satisfy the goal. The domain is represented by a set of state variables, each one

* The work by Bozzelli, Molinari, Montanari, and Peron has been supported by the GNCS project *Formal methods for verification and synthesis of discrete and hybrid systems.* The work by Molinari and Montanari has also been supported by the project *(PRID) ENCASE—Efforts in the uNderstanding of Complex interActing SystEms.*

modeling a component. State variables are independent of each other, but a set of synchronization rules constrain the temporal relations among them. The evolution of the value of each state variable over time is controlled by a transition function, and described by means of a timeline (a sequence of tokens).

Timeline-based planning has been applied in several contexts [2,4,5,10], but a systematic study of its expressiveness and complexity has been undertaken only recently. The temporal domain is typically assumed to be discrete; the dense case is commonly dealt with by forcing a discretization of the domain. In [7,8] Gigante et al. proved that timeline-based planning is **EXPSPACE**-complete.

In this paper, we address the timeline-based planning problem over dense temporal domains, without resorting to any form of discretization. Since the problem in its full generality is known to be *undecidable* over dense domains [3], we restrict ourselves to a constrained form of synchronization rules, namely, trigger-less ones, and give an optimal **NP** planning algorithm for this case.

Organization of the paper. In Section 2, we give some background knowledge about timeline-based planning. Then, we show that if we allow only trigger-less synchronization rules, the problem is decidable (unlike the general case). First, in Section 3 we provide a **PSPACE** algorithm by encoding planning instances into timed automata. Next, by exploiting a pair of fundamental bounds on plans obtained from timed automata, in Section 4 we prove that timeline-based planning with trigger-less rules is in fact **NP**-complete: we give an **NP** algorithm and then a matching complexity lower bound. Conclusions give an assessment of the achieved results and outline future research themes.

2 The Timeline-Based Planning Problem

In this section, we give an account of timeline-based planning (TP). We refer the reader to [6,7] for details. Let $\mathbb{N}$, $\mathbb{R}_+$, and $\mathbb{Q}_+$ denote the naturals, non-negative reals, and non-negative rationals, respectively. Let *Intv* be the set of intervals in $\mathbb{R}_+$ whose endpoints are in $\mathbb{Q}_+ \cup \{\infty\}$.
In TP, domain knowledge is encoded by a set of state variables, whose behaviour over time is described by transition functions and synchronization rules.

Definition 1. *A* state variable x *is a triple* $x = (V_x, T_x, D_x)$*, where* V_x *is the* finite domain *of the variable* x*,* $T_x : V_x \to 2^{V_x}$ *is the* value transition function*, which maps each* $v \in V_x$ *to the (possibly empty) set of successor values, and* $D_x : V_x \to Intv$ *is the* constraint function *that maps each* $v \in V_x$ *to an interval.*

A *token* for a variable x is a pair (v, d) consisting of a value $v \in V_x$ and a duration $d \in \mathbb{R}_+$ such that $d \in D_x(v)$. Intuitively, a token for x represents an interval of time where x takes value v. In the following we will also denote (v, d) as (x, v, d) to make x explicit. The behavior of x is specified by means of a *timeline* which is a non-empty sequence of tokens $\pi = (v_0, d_0) \cdots (v_n, d_n)$ consistent with T_x, i.e., $v_{i+1} \in T_x(v_i)$ for all $0 \leq i < n$. The *start time* $\mathsf{s}(\pi, i)$ and

the *end time* $\mathsf{e}(\pi, i)$ of the i-th token ($0 \leq i \leq n$) of the timeline π are defined as $\mathsf{e}(\pi, i) = \sum_{h=0}^{i} d_h$ and $\mathsf{s}(\pi, i) = 0$ if $i = 0$, and $\mathsf{s}(\pi, i) = \sum_{h=0}^{i-1} d_h$ otherwise.

Given a finite set SV of state variables, a *multi-timeline* of SV is a mapping Π assigning to each state variable $x \in SV$ a timeline for x. Multi-timelines of SV can be constrained by a set of *synchronization rules*, which relate tokens, possibly belonging to different timelines, through temporal constraints on the start/end times of tokens (time-point constraints) and on the difference between start/end times of tokens (interval constraints). The synchronization rules exploit an alphabet $\Sigma = \{o, o', \ldots\}$ of *token names* to refer to the tokens along a multi-timeline, and are based on the notions of *atom* and *existential statement*.

Definition 2. *An* atom *is either a clause of the form* $o_1 \leq_I^{e_1,e_2} o_2$ (interval atom)*, or of the forms* $o_1 \leq_I^{e_1} t$ *or* $o_1 \geq_I^{e_1} t$ (time-point atom)*, where* $o_1, o_2 \in \Sigma$*,* $I \in Intv$*,* $t \in \mathbb{Q}_+$*, and* $e_1, e_2 \in \{\mathsf{s}, \mathsf{e}\}$*.*

An atom ρ is evaluated with respect to a *Σ-assignment λ_Π for a given multi-timeline Π* which is a mapping assigning to each token name $o \in \Sigma$ a pair $\lambda_\Pi(o) = (\pi, i)$ such that π is a timeline of Π and $0 \leq i < |\pi|$ is a position along π (intuitively, (π, i) represents the token of Π referenced by the name o). An interval atom $o_1 \leq_I^{e_1,e_2} o_2$ *is satisfied by* λ_Π if $e_2(\lambda_\Pi(o_2)) - e_1(\lambda_\Pi(o_1)) \in I$. A point atom $o \leq_I^{e} t$ (resp., $o \geq_I^{e} t$) *is satisfied by* λ_Π if $t - e(\lambda_\Pi(o)) \in I$ (resp., $e(\lambda_\Pi(o)) - t \in I$).

Definition 3. *An* existential statement *$\mathcal{E}$ for a finite set SV of state variables is a statement of the form $\mathcal{E} := \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\mathcal{C}$, where $\mathcal{C}$ is a conjunction of atoms, $o_i \in \Sigma$, $x_i \in SV$, and $v_i \in V_{x_i}$ for each $i = 1, \ldots, n$. The elements $o_i[x_i = v_i]$ are called* quantifiers*. A token name used in $\mathcal{C}$, but not occurring in any quantifier, is said to be* free*. Given a Σ-assignment λ_Π for a multi-timeline Π of SV, we say that λ_Π* is consistent with the existential statement *$\mathcal{E}$ if for each quantified token name o_i, we have $\lambda_\Pi(o_i) = (\pi, h)$ where $\pi = \Pi(x_i)$ and the h-th token of π has value v_i. A multi-timeline Π of SV* satisfies *$\mathcal{E}$ if there exists a Σ-assignment λ_Π for Π consistent with $\mathcal{E}$ such that each atom in $\mathcal{C}$ is satisfied by λ_Π.*

Definition 4. *A* synchronization rule *$\mathcal{R}$ for a finite set SV of state variables is a rule of one of the forms $o_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, or $\top \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, where $o_0 \in \Sigma$, $x_0 \in SV$, $v_0 \in V_{x_0}$, and $\mathcal{E}_1, \ldots, \mathcal{E}_k$ are* existential statements*. In rules of the first form* (trigger rules)*, the quantifier $o_0[x_0 = v_0]$ is called* trigger*, and we require that only o_0 may appear free in $\mathcal{E}_i$ (for $i = 1, \ldots, n$). In rules of the second form* (trigger-less rules)*, we require that no token name appears free.*

Intuitively, a trigger $o_0[x_0 = v_0]$ acts as a universal quantifier, which states that *for all* the tokens of the timeline for the state variable x_0, where x_0 takes the value v_0, at least one of the existential statements $\mathcal{E}_i$ must be satisfied. Trigger-less rules simply assert the satisfaction of some existential statement.

The semantics of synchronization rules is formally defined as follows.

Definition 5. *Let Π be a multi-timeline of a set SV of state variables. Given a* trigger-less rule $\mathcal{R}$ *of SV, Π* satisfies $\mathcal{R}$ *if Π satisfies some existential statement of $\mathcal{R}$. Given a* trigger rule $\mathcal{R}$ *of SV with trigger $o_0[x_0 = v_0]$, Π* satisfies $\mathcal{R}$ *if for every position i of the timeline $\pi = \Pi(x_0)$ for x_0 such that $\pi(i) = (v_0, d)$, there is an existential statement $\mathcal{E}$ of $\mathcal{R}$ and a Σ-assignment λ_Π for Π consistent with $\mathcal{E}$ such that $\lambda_\Pi(o_0) = (\pi, i)$ and λ_Π satisfies all the atoms of $\mathcal{E}$.*

A TP domain $P = (SV, S)$ is specified by a finite set SV of state variables and a finite set S of synchronization rules modeling their admissible behaviors. Trigger-less rules can be used to express initial and intermediate conditions, and the goals of the problem, while trigger rules are useful to specify invariants and response requirements. A *plan for P* is a multi-timeline of SV satisfying all the rules in S. The end time of the token ending last in P is said the *horizon* of P. The TP problem is to decide if, given a TP domain P, there is a plan for P.

3 Timed Automata and Finite Bounds for Plans

TP over dense temporal domains is, in its full generality, *undecidable* [3]. However, if we restrict to trigger-less synchronization rules only, we get a decidable (in fact, **NP**-complete) problem. To show this, we start by encoding planning into a parallel composition of timed automata (TA). The intuition is that each timeline can be seen as a timed word "described" by the TA associated with the corresponding variable. A plan for k variables is then a timed k-multiword, i.e., a timed word over a structured alphabet featuring a component for each variable.

We call k-MWTA the composition of TAs accepting k-multiwords encoding plans. We will show at the end of the section how to derive from k-MWTAs a pair of bounds, on the number of tokens of a plan, and on its horizon, which will be at the basis of the **NP** algorithm of the next section.

Let us recall some basic notions. Let w be a finite or infinite word over some alphabet. An *infinite timed word* w over a finite alphabet Σ is an infinite word $w = (a_1, \tau_1)(a_2, \tau_2)\cdots$ over $\Sigma \times \mathbb{R}_+$ (intuitively, τ_i is the time at which the event a_i occurs) such that the sequence $\tau = \tau_1, \tau_2, \ldots$ of timestamps satisfies: (1) $\tau_i \leq \tau_{i+1}$ for all $i \geq 1$ (monotonicity), and (2) for all $t \in \mathbb{R}_+$, $\tau_i \geq t$ for some $i \geq 1$ (divergence/progress). The timed word w is also denoted by the pair (σ, τ), where σ is the (untimed) infinite word $a_1 a_2 \cdots$ and τ is the sequence of timestamps. An *ω-timed language* over Σ is a set of infinite timed words over Σ.

Let C be a set of clocks. A clock valuation $val : C \to \mathbb{R}_+$ is a function assigning a real value to each clock in C. A *clock constraint* over C is a *Boolean combination* of atomic formulas of the form $c \bullet c' + cst$ or $c \bullet cst$, where $\bullet \in \{\geq, \leq\}$, $c, c' \in C$, and $cst \in \mathbb{Q}_+$ is a constant. We will often use the interval-based notation, for instance, $c \in [2, 7.4]$. We denote by $\Phi(C)$ the set of clock constraints over C. Given a clock valuation val for C and a clock constraint θ over C, we say that val satisfies θ if θ evaluates to true replacing each occurrence of a clock c in θ by $val(c)$, and interpreting Boolean connectives in the standard way. Given $t \in \mathbb{R}_+$, $(val + t)$ denotes the valuation such that, for all $c \in C$, $(val + t)(c) = val(c) + t$. For $Res \subseteq C$, $val[Res](c) = 0$ if $c \in Res$, and $val[Res](c) = val(c)$ otherwise.

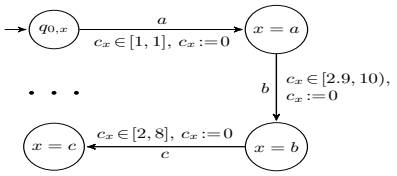

Fig. 1. (Part of) TA $\mathcal{A}_x$ for $x=(V_x,T_x,D_x)$, with $V_x = \{a, b, c, \ldots\}$, $b \in T_x(a)$, $c \in T_x(b)$, $D_x(a) = [2.9, 10)$, $D_x(b) = [2, 8], \ldots$.

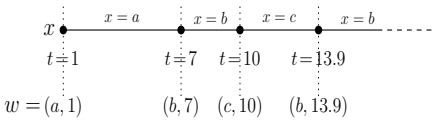

Fig. 2. An example of timeline for the state variable x of Fig. 1, with the timed word encoding it, and accepted by $\mathcal{A}_x$.

Definition 6 ([1]). A (Büchi) timed automaton (TA) *over Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, C, \Delta, F)$, where Q is a finite set of (control) states, $Q_0 \subseteq Q$ is the set of initial states, C is a finite set of clocks, $F \subseteq Q$ is the set of accepting states, and $\Delta \subseteq Q \times \Sigma \times \Phi(C) \times 2^C \times Q$ is the transition relation.*

A configuration of $\mathcal{A}$ is a pair $(q, sval)$, where $q \in Q$, and $sval$ is a clock valuation for C. A run π of $\mathcal{A}$ on $w = (\sigma, \tau)$ is an infinite sequence of configurations $\pi = (q_0, sval_0)(q_1, sval_1) \cdots$ such that $q_0 \in Q_0$, $sval_0(c) = 0$ for all $c \in C$ (*initialization* requirement), and the following constraint holds (*consecution*): for all $i \geq 1$, for some $(q_{i-1}, \sigma_i, \theta, Res, q_i) \in \Delta$, $sval_i = (sval_{i-1} + \tau_i - \tau_{i-1})[Res]$ and $(sval_{i-1} + \tau_i - \tau_{i-1}) \models \theta$ (we let $\tau_0 = 0$). The run π is *accepting* if there are infinitely many $i \geq 0$ such that $q_i \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is the set of infinite timed words w over Σ s.t. there is an accepting run of $\mathcal{A}$ on w.

Let us now introduce the encoding of a timeline by a timed word, and the TA for a state variable. Hereafter, we will assume that, for every x and every $v \in V_x$, we have $T_x(v) \neq \emptyset$; however, this constraint can easily be relaxed [3].

Definition 7. *Let $x = (V_x, T_x, D_x)$ be a state variable. The* timeline for x encoded by a timed word $(a_1, \tau_1)(a_2, \tau_2) \cdots$ *is the sequence of tokens (x, a_1, t_1) $(x, a_2, t_2) \cdots$, where, for $i \geq 1$, $a_i \in V_x$, $a_{i+1} \in T_x(a_i)$, and $t_i = \tau_{i+1} - \tau_i \in D_x(a_i)$.*

Definition 8. A TA for a state variable $x = (V_x, T_x, D_x)$ *is a tuple $\mathcal{A}_x = (V_x, Q, \{q_{0,x}\}, \{c_x\}, \Delta, Q)$, where $Q = V_x \cup \{q_{0,x}\}$ ($q_{0,x} \notin V_x$), and $\Delta = \{(v', v, c_x \in D_x(v'), \{c_x\}, v) \mid v', v \in V_x,\ v \in T_x(v')\} \cup \{(q_{0,x}, v, c_x \in [1,1], \{c_x\}, v) \mid v \in V_x\}$.*

Intuitively, $\mathcal{A}_x$ accepts all timed words encoding a timeline for the state variable x. Let us note that all states are accepting. Moreover, the constraints of a transition $\delta \in \Delta$ on the unique clock c_x are determined by the (value of the) source state of δ. The reason why we set $c_x \in [1, 1]$ on all the transitions from $q_{0,x}$ is technical and will be clear later. See Fig. 1 and 2 for some examples.

In the following, we introduce the formalism of k-multiword TA (k-MWTA). A k-MWTA accepts a language of timed k-multiwords, formally defined as follows. For $k \geq 1$ pairwise disjoint alphabets $\Sigma_1, \ldots, \Sigma_k$, and the symbol $\epsilon \notin \bigcup_{1 \leq i \leq k} \Sigma_i$, k-Σ denotes the multialphabet $\{(a_1, \ldots, a_k) \mid a_i \in \Sigma_i \cup \{\epsilon\},\ \text{for } 1 \leq i \leq k\} \setminus \{(\epsilon, \ldots, \epsilon)\}$. A *$k$-multiword* is a word over k-Σ. Intuitively, a k-multiword $\overrightarrow{a}_1 \cdot \overrightarrow{a}_2 \cdots$ is a synchronization of k words over the alphabets $\Sigma_1, \ldots, \Sigma_k$; in

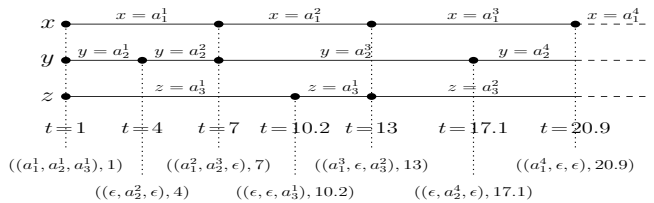

Fig. 3. An example of timelines for x, y, z with the timed 3-multiword encoding them.

$\vec{a}_j = (a_j^1, \ldots a_j^k)$, for $j \geq 1$, all the symbols a_j^i such that $a_j^i \neq \epsilon$, with $1 \leq i \leq k$, occur at the same instant of time, and if $a_j^i = \epsilon$, the intended meaning is that no symbol (event) of the i-th alphabet Σ_i occurs at that time. A timed k-multiword is just a timed word (σ, τ) where the untimed word σ is a k-multiword.

The notion of timeline encoded by a timed word can be extended to the multi-timeline encoded by a timed k-multiword.

Definition 9. *Let $x_1, \ldots, x_k$, with $x_i = (V_{x_i}, T_{x_i}, D_{x_i})$, be k state variables. The timeline for x_j encoded by a timed k-multiword $(\vec{a}_1, \tau_1)(\vec{a}_2, \tau_2) \cdots$ is the timeline encoded by $d\epsilon\big((\vec{a}_1[j], \tau_1)(\vec{a}_2[j], \tau_2) \cdots\big)$, where $d\epsilon((\sigma, \tau))$ is the timed word obtained from (σ, τ) removing occurrences of symbols (ϵ, τ'), for any $\tau' \in \mathbb{R}_+$.*

See Fig. 3 for an example. A k-MWTA can be viewed as a suitable parallel composition of TAs communicating via shared clocks.

Definition 10. *For $1 \leq i \leq k$, let $\mathcal{A}_i = (\Sigma_i, Q_i, Q_{0,i}, C_i, \Delta_i, F_i)$ be a TA. A k-multiword TA (k-MWTA) for $\mathcal{A}_1, \ldots, \mathcal{A}_k$ is a TA k-$\mathcal{A} = (k\text{-}\Sigma, Q, Q_0, C, \Delta, F)$:*

- *$Q = (Q_1 \times \ldots \times Q_k) \cup \{q_f\}$, q_f being an (optional) auxiliary final state,*
- *$Q_0 = Q_{0,1} \times \ldots \times Q_{0,k}$, $C = C_1 \cup \ldots \cup C_k$, $F = (F_1 \times \ldots \times F_k) \cup \{q_f\}$,*
- *if $\Big((q_1, \ldots, q_k), (a_1, \ldots, a_k), \Theta, \mathcal{C}l, (q'_1, \ldots, q'_k)\Big) \in \Delta$ and $(q_1, \ldots, q_k), (q'_1, \ldots, q'_k) \in Q \setminus \{q_f\}$, then $\Theta = \bigwedge_{i=1}^k \theta_i$, $\mathcal{C}l = \bigcup_{i=1}^k Res_i$, and, for all $i = 1, \ldots, k$,*
  - *if $a_i \neq \epsilon$, then there exists $(q_i, a_i, \theta_i, Res_i, q'_i) \in \Delta_i$,*
  - *if $a_i = \epsilon$, then it holds $q_i = q'_i$, $\theta_i = \top$, and $Res_i = \emptyset$.*

Notice that more than one k-MWTA can be generated from given $\mathcal{A}_1, \ldots, \mathcal{A}_k$. The ones we are interested in are those of Definition 11 and Definition 13 below.

We now instantiate Definition 10 over the TA for $x_1, \ldots, x_k$.

Definition 11. *Let $x_1, \ldots, x_k$ be k state variables and, for $1 \leq i \leq k$, let $\mathcal{A}_{x_i} = (\Sigma_i = V_{x_i}, Q_i, \{q_{0,i}\}, \{c_i\}, \Delta_i, Q_i)$. A k-MWTA for $\mathcal{A}_{x_1}, \ldots, \mathcal{A}_{x_k}$ is the k-MWTA k-$\mathcal{A}_{x_1,\ldots,x_k} = (k\text{-}\Sigma, Q, \{q_0\}, C, \Delta' \cup \Delta'', F)$, where:*

- *$Q = Q_1 \times \ldots \times Q_k$, $q_0 = (q_{0,1}, \ldots, q_{0,k})$, $C = \{c_1, \ldots, c_k\}$, $F = Q_1 \times \ldots \times Q_k$,*
- *$\Big((q_1, \ldots, q_k), (a_1, \ldots, a_k), \Theta, \mathcal{C}l, (q'_1, \ldots, q'_k)\Big) \in \Delta'$ iff $(q_1, \ldots, q_k) \neq q_0$, $\Theta = \bigwedge_{i=1}^k \theta_i$, $\mathcal{C}l = \bigcup_{i=1}^k Res_i$, and, for all $i = 1, \ldots, k$,*
  - *if $a_i \neq \epsilon$, then there exists $(q_i, a_i, \theta_i, Res_i, q'_i) \in \Delta_i$,*
  - *if $a_i = \epsilon$, then it holds $q_i = q'_i$, $\theta_i = \top$, and $Res_i = \emptyset$.*
- *$\Delta'' = \{(q_0, (a_1, \ldots, a_k), \bigwedge_{i=1}^k c_i \in [1, 1], \bigcup_{i=1}^k \{c_i\}, (a_1, \ldots, a_k)) \mid (a_1, \ldots, a_k) \in \Sigma_1 \times \ldots \times \Sigma_k\}$*

Proposition 1. *Let $x_1, \ldots, x_k$ be k state variables, with $k \geq 1$. $\mathcal{L}_T(k\text{-}\mathcal{A}_{x_1,\ldots,x_k})$ is a set of k-multiwords, each one encoding a timeline for each of $x_1, \ldots, x_k$.*

Let us now introduce the k-MWTA for the *synchronizations rules*. Since each rule has the form $\mathcal{E}_1 \vee \ldots \vee \mathcal{E}_n$, we focus on a single $\mathcal{E}_i$, the automaton for the rule being just the union of the automata for $\mathcal{E}_1, \ldots, \mathcal{E}_n$. $\mathcal{E}_i$ has the form $\exists o_1[x_1 = v_{1,j_1}] \cdots \exists o_n[x_n = v_{n,j_n}].\mathcal{C}$, where $\mathcal{C}$ is a conjunction of atoms. We associate two clock variables with each quantifier $o_i[x_i = v_{i,j_i}]$—named $c_{o_i,S}$ and $c_{o_i,E}$—which, intuitively, are reset when the token chosen for o_i starts and ends, respectively. In order to select a suitable token along the timeline, $c_{o_i,S}$ and $c_{o_i,E}$ are non-deterministically reset when x_i takes the value $v_{i,j_i} \in V_{x_i}$. Moreover, to deal with atoms involving a time constant (time-point atoms), we introduce a clock variable c_{glob}, which measures the current time and is *never reset*. For technical reasons, we assume that the start of activities is at time 1 and, consequently, the reset of any $c_{o_i,S}$ and $c_{o_i,E}$ cannot happen *before* 1 time unit has passed from the beginning of the timed word/plan. In fact, in Definition 8, we have $c_x \in [1, 1]$ on all the transitions from $q_{0,x}$ (for this reason, we must also add 1 to all time constants in all time-point atoms). This assumption implies that the value of $c_{o_i,S}$ is equal to that of c_{glob} if $c_{o_i,S}$ has never been reset, and less otherwise. Since only one token for each quantifier is chosen in a timeline, $c_{o_i,S}$ must be reset only once: a transition resetting $c_{o_i,S}$ is enabled only if the constraint $c_{o_i,S} = c_{glob}$ is satisfied (likewise for $c_{o_i,E}$).

We now define a TA for a state variable x, suitably resetting the clocks associated with all the quantifiers over x. For a set of token names O, we denote by $\Lambda(O, x, v)$ the subset of names $o \in O$ such that $o[x = v]$ for a variable x and a value $v \in V_x$, and by $\Lambda(O, x) = \bigcup_{v \in D_x} \Lambda(O, x, v)$. Moreover $C_S(O)$ (resp., $C_E(O)$) represents the set $\{c_{o,S} \mid o \in O\}$ (resp., $\{c_{o,E} \mid o \in O\}$).

Definition 12. *Given $x = (V_x, T_x, D_x)$ and quantifiers $o_1[x = v_1], \ldots, o_\ell[x = v_\ell]$, a TA for $x, o_1, \ldots, o_\ell$ is $\mathcal{A}_{x,o_1,\ldots,o_\ell} = (V_x, Q, \{q_0\}, C, \Delta' \cup \Delta'', \emptyset)$, where:*

- *$Q = V_x \cup \{q_0\}$ and $C = \{c_{glob}\} \cup \{c_{o_i,S}, c_{o_i,E} \mid i = 1, \ldots, \ell\}$,*
- *Δ' is the set of tuples $\Big(v, a, \bigwedge_{o \in P} c_{o,S} = c_{glob} \wedge \bigwedge_{o \in R} (c_{o,S} < c_{glob} \wedge c_{o,E} = c_{glob}) \wedge$*

$$\bigwedge_{o \in \Lambda(\{o_1,\ldots,o_k\},x,v) \setminus R} (c_{o,S} = c_{glob} \vee c_{o,E} < c_{glob}), C_S(P) \cup C_E(R), a\Big)$$

  where $v \in V_x$, $P \subseteq \Lambda(\{o_1, \ldots, o_\ell\}, x, a)$ and $R \subseteq \Lambda(\{o_1, \ldots, o_\ell\}, x, v)$;
- *$\Delta'' = \{(q_0, a, c_{glob} \in [1, 1], C_S(P), a) \mid a \in V_x, P \subseteq \Lambda(\{o_1, \ldots, o_\ell\}, x, a)\}$.*

In Definition 12, R (resp., $\Lambda(\{o_1, \ldots, o_k\}, x, v) \setminus R$) is the set of token names whose end clock *must (resp., must not) be reset*. The next k-MWTA is a synchronization of TA (each one for the tokens over the same variable) for quantifiers in $\mathcal{E}$.

Definition 13. *Let $x_1, \ldots, x_k$ be k variables and $\mathcal{E} = \exists o_1[x_{j_1} = v_1] \cdots \exists o_n[x_{j_n} = v_n].\mathcal{C}$, with $\{j_1, \ldots, j_n\} \subseteq \{1, \ldots, k\}$. A k-MWTA for $x_1, \ldots, x_k, o_1, \ldots, o_n$, and $\mathcal{E}$ is a k-MWTA for the TA $\mathcal{A}_{x_i,\Lambda(\{o_1,\ldots,o_n\},x_i)} = (\Sigma_i = V_{x_i}, Q_i, \{q_{0,i}\}, C_i, \Delta_i, \emptyset)$, with $1 \leq i \leq k$, $k\text{-}\mathcal{A}_{x_1,\ldots,x_k,\mathcal{E}} = (k\text{-}\Sigma, Q, \{q_0\}, C, \Delta' \cup \Delta'' \cup \Delta''' \cup \Delta'''', \{q_f\})$, where:*

- $Q = (Q_1 \times \ldots \times Q_k) \cup \{q_f\}$, $q_0 = (q_{0,1}, \ldots, q_{0,k})$, $C = \{c_{glob}\} \cup \{c_{o_i,S}, c_{o_i,E} | i \in [1,n]\}$,
- Δ' *is the set of tuples* $\Big((v_1, \ldots, v_k), (a_1, \ldots, a_k), \bigwedge_{i=1}^{k} \Big(\bigwedge_{o \in P_i} c_{o,S} = c_{glob} \wedge$

$$\bigwedge_{o \in R_i} (c_{o,S} < c_{glob} \wedge c_{o,E} = c_{glob}) \wedge \bigwedge_{o \in \overline{R}_i} (c_{o,S} = c_{glob} \vee c_{o,E} < c_{glob})\Big), \bigcup_{i=1}^{k} (C_S(P_i) \cup C_E(R_i)), (v'_1, \ldots, v'_k)\Big)$$

  satisfying, for all $i = 1, \ldots, k$, *the following conditions:*
  - *if* $a_i = \epsilon$, *then* $v_i = v'_i$, $P_i = \emptyset$, *and* $R_i = \overline{R}_i = \emptyset$,
  - *if* $a_i \neq \epsilon$, $a_i = v'_i \in D_{x_i}$, $P_i \subseteq \Lambda(\{o_1, \ldots, o_n\}, x_i, v'_i)$, $R_i \dot{\cup} \overline{R}_i = \Lambda(\{o_1, \ldots, o_n\}, x_i, v_i)$
- $\Delta'' = \{(q_0, (a_1, \ldots, a_k), c_{glob} \in [1,1], \bigcup_{i=1}^{k} C_S(P_i), (a_1, \ldots, a_k)) \mid (a_1, \ldots, a_k) \in V_{x_1} \times \ldots \times V_{x_k}, P_i \subseteq \Lambda(\{o_1, \ldots, o_n\}, x_i, a_i)$ *for* $1 \leq i \leq k\}$,
- $\Delta''' = \{(q, (a_1, \ldots, a_k), \Phi_{\mathcal{C}} \wedge \bigwedge_{i=1}^{k} (c_{o_i,S} < c_{glob} \wedge c_{o_i,E} < c_{glob}), \emptyset, q_f) \mid q \in V_{x_1} \times \ldots \times V_{x_k}, (a_1, \ldots, a_k) \in k\text{-}\Sigma\}$, *where* $\Phi_{\mathcal{C}}$ *is the translation of* $\mathcal{C}$ *into a (conjunction of)* TA *clock constraint(s),*
- $\Delta'''' = \{(q_f, (a_1, \ldots, a_k), \top, \emptyset, q_f) \mid (a_1, \ldots, a_k) \in k\text{-}\Sigma\}$.

Note that k-$\mathcal{A}_{x_1,\ldots,x_k,\mathcal{E}}$ can enter q_f only when all token clocks have been reset ($\bigwedge_{i=1}^{k} (c_{o_i,S} < c_{glob} \wedge c_{o_i,E} < c_{glob})$) and all $\mathcal{C}$'s conditions are verified.

The TA $\tilde{\mathcal{A}}$ for a TP domain $P = (\{x_1, \ldots, x_k\}, S)$ is built by exploiting the standard union and intersection operations for TA: $\tilde{\mathcal{A}}$ is obtained by intersecting (i) the k-MWTA k-$\mathcal{A}_{x_1,\ldots,x_k}$ for the state variables (Definition 11) with (ii) a TA k-$\mathcal{A}_{x_1,\ldots,x_k,\mathcal{R}}$ for each trigger-less rule $\mathcal{R} = \top \rightarrow \bigvee_{1 \leq i \leq m} \mathcal{E}_i$ in S, which is the disjunction of m k-MWTAk-$\mathcal{A}_{x_1,\ldots,x_k,\mathcal{E}_i}$ (Definition 13), one for each $\mathcal{E}_i$.

Proposition 2. $\mathcal{L}_T(\tilde{\mathcal{A}})$ *is the set of plans for* $P = (\{x_1, \ldots, x_k\}, S)$.

Theorem 1. *Let* $P = (SV, S)$ *be a TP domain with trigger-less rules only. The problem of checking the existence of a plan for* P *is in* ***PSPACE****.*

The theorem is proved by inspecting the standard **PSPACE** emptiness checking algorithm for TA, in order to verify that the space complexity remains polynomial also for k-MWTA. For details we refer the reader to [3]. The check involves building the so-called *region automaton* [1] and concludes by an emptiness test on a Büchi automaton of $g = O(|C| \cdot r \cdot \mathcal{V})$ states, where C is the clock set of $\tilde{\mathcal{A}}$, $r = O(|C|! \cdot 2^{|C|} \cdot (2K+2)^{|C|})$ is the number of regions (K is the maximum constant occurring in P [4]), and $\mathcal{V} = O(|S| \cdot (V^k \cdot d)^{|S|+1})$ is the number of states of $\tilde{\mathcal{A}}$, with $V = \max_{i=1}^{k} |V_{x_i}|$ and d the number of disjuncts in the longest S rule.

Exploiting this result, we can derive a finite horizon for the plans of a (any) problem, that is, if a TP domain $P = (SV, S)$ has a solution plan, then P also

[4] I.e., an upper/lower bound of an interval of a token duration, a time constant in an atom, or a bound at the subscript of an atom. We assume they are encoded in binary.

has a solution plan whose horizon is no greater than a given bound. Analogously, we calculate a bound on the maximum number of tokens in a plan.

In order to check the emptiness of the previous Büchi automaton of g states, it is enough to find a finite word uv, where: (i) $|u|, |v| \leq g$, and (ii) there is a run of the automaton that, from an initial state, upon reading u, reaches a state q, and upon reading v from q reaches a final state and gets ultimately back to q. Finally, we observe that each transition of the Büchi automaton corresponds to the start point of at least a token in some timeline of (a plan for) P, and at most a token for each timeline (when all these tokens start simultaneously). This yields a bound on the number of tokens: $2 \cdot g \cdot |SV|$. We can also derive a bound on the horizon of the plan: $2 \cdot g \cdot |SV| \cdot (K+1)$. As a matter of fact, every transition taken in the timed automaton may let at most $K+1$ time units pass, as K accounts in particular for the maximum constant to which a (any) clock is compared (clearly, an unbounded quantity of time units may pass, but after $K+1$ the last region of the region automaton will certainly have been reached).

By exploiting this pair of bounds, we now describe an **NP** algorithm.

4 TP with Trigger-less Rules is NP-complete

To start with, we provide an example showing that there is no *polynomial-size* plan for some TP domains. Thus, an explicit enumeration of all tokens across all timelines does not represent a suitable polynomial-size certificate. Let $p(i)$ be the i-th prime number, assuming $p(1) = 1$, $p(2) = 2$, $p(3) = 3$, $p(4) = 5$, and so on. For $1 \leq i \leq n$, let $x_i = (\{v_i\}, \{(v_i, v_i)\}, D_{x_i})$, with $D_{x_i}(v_i) = [p(i), p(i)]$. The rule $\top \rightarrow \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n]. \bigwedge_{i=1}^{n-1} o_i \leq^{\mathsf{e},\mathsf{e}}_{[0,0]} o_{i+1}$ is asking for the existence of a "synchronization point", where n tokens (one for each variable) have their ends aligned. Due to the allowed token durations, the first such time point is $\prod_{i=1}^{n} p(i) \geq 2^{n-1}$. Hence, in any plan, the timeline for x_1 features at least 2^{n-1} tokens: no explicit polynomial-time enumeration of such tokens is possible. It follows that there is no trivial guess-and-check **NP** algorithm.

We now present a non-deterministic polynomial-time algorithm, proving that the TP problem with trigger-less rules is in **NP**. At the end of the section, we will see that such problem is in fact **NP**-complete. The algorithm is structured in 3 phases, whose description follows.

(Phase 1) Preprocessing. As a preliminary preprocessing phase, we consider all rational values occurring in the input TP domain $P = (SV, S)$—be either upper/lower bounds of an interval of a token duration, a time constant in an atom, or upper/lower bounds (u or ℓ) at the subscript of an atom—and convert them into integers by multiplying them by the lcm γ of all denominators. This involves a quadratic blowup in the input size, being constants encoded in binary. Given a plan for P' (where all values are integers), we can obtain one for the original P by dividing the start/end times of all tokens in each timeline by γ.

(Phase 2) Non-deterministic token positioning. The algorithm then non-deterministically guesses, for every trigger-less rule in S, a disjunct—and deletes all the others. Moreover, for each (left) quantifier $o_i[x_i = v_i]$, it guesses the

integer part of both the start and the end time of the token for x_i to which o_i is mapped. We call such time instants, respectively, $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$.[5] We observe that all start/end times $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$, being less than or equal to $2 \cdot g \cdot |SV| \cdot (K+1)$ (the finite horizon bound), have an integer part that can be encoded with polynomially many bits (and thus can be generated in polynomial time). Let us now consider the fractional parts of the start/end time of the tokens associated with quantifiers (we denote them by $\mathsf{s}_{frac}(o_i)$ and $\mathsf{e}_{frac}(o_i)$). The algorithm non-deterministically guesses an order of all such fractional parts. It has to specify, for every token start/end time, whether it is integer ($\mathsf{s}_{frac}(o_i) = 0$, $\mathsf{e}_{frac}(o_i) = 0$) or not ($\mathsf{s}_{frac}(o_i) > 0$, $\mathsf{e}_{frac}(o_i) > 0$). Every possibility can be generated in polynomial time. Some trivial tests should be performed, i.e., for all o_i, $\mathsf{s}_{int}(o_i) \leq \mathsf{e}_{int}(o_i)$, each token is assigned an end time equal or greater than its start time, and no two tokens for the same variable are overlapping.

It is routine to check that if we change the start/end time of (some of the) tokens associated with quantifiers, but we leave unchanged (*i*) all the integer parts, (*ii*) zeroness/non-zeroness of fractional parts, and (*iii*) the order of the fractional parts, then the satisfaction of the (atoms in the) trigger-less rules does not change. This is due to all the constants being integers, as a result of the preprocessing step.[6] Therefore we can now check whether all rules are satisfied.

(Phase 3) Enforcing legal token durations and timeline evolutions. We now conclude by checking that: (*i*) all tokens associated with a quantifier have an admissible duration, and that (*ii*) there exists a legal timeline evolution between pairs of adjacent such tokens over the same variable (two tokens are *adjacent* if there is no other token associated with a quantifier in between). We will enforce all these requirements as constraints of a *linear problem*, which can be solved in deterministic polynomial time (e.g., using the ellipsoid algorithm). When needed, we use *strict inequalities*, which are not allowed in linear programs. We will show later how to convert these into non-strict ones.

We start by associating non-negative variables $\alpha_{o_i,s}, \alpha_{o_i,e}$ with the fractional parts of the start/end times $\mathsf{s}_{frac}(o_i)$, $\mathsf{e}_{frac}(o_i)$ of every token for a quantifier $o_i[x_i = v_i]$. First, we add the linear constraints $0 \leq \alpha_{o_i,s} < 1,\ 0 \leq \alpha_{o_i,e} < 1$. Then, we also need to enforce that the values of $\alpha_{o_i,s}, \alpha_{o_i,e}$ respect the decided order of the fractional parts, e.g., $0 = \alpha_{o_i,s} = \alpha_{o_j,s} < \alpha_{o_k,s} < \cdots < \alpha_{o_j,e} < \alpha_{o_i,e} = \alpha_{o_k,e} < \cdots$. To enforce requirement (*i*), we set, for all $o_i[x_i = v_i]$, $a \leq (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e}) - (\mathsf{s}_{int}(o_i) + \alpha_{o_i,s}) \leq b$, where $D_{x_i}(v_i) = [a, b]$. Clearly, strict ($<$) inequalities must be used for a left/right open interval. To enforce (*ii*), i.e., the existence of a legal timeline evolution between each pair of adjacent tokens for the same state variable, say $o_i[x_i = v_i]$ and $o_j[x_i = v_j]$, we proceed as follows (analogous considerations hold for an evolution between $t = 0$ and the first token).

[5] W.l.o.g., we can assume that all quantifiers refer to distinct tokens in timelines. As a matter of fact, the algorithm can non-deterministically choose to force two (or more) quantifiers $o_i[x_i = v_i]$ and $o_j[x_i = v_i]$, over the same variable and value, to refer to the same token just by rewriting all occurrences of o_j as o_i in the atoms of the rules.

[6] We observe that, by leaving unchanged all integer parts, fractional parts' zeroness and order, the region of the region graph of the timed automaton does not change.

Let us consider each state variable $x_i = (V_i, T_i, D_i)$ as a directed graph $G = (V_i, T_i)$, where D_i associates with each vertex $v \in V_i$ a duration interval. We have to decide whether there is (a) a path in G, possibly with repeated vertices and edges, $v_0 \cdot v_1 \cdots v_{n-1} \cdot v_n$, where $v_0 \in T_i(v_i)$ and v_n, with $v_j \in T_i(v_n)$, are non-deterministically generated, and (b) a list of non-negative reals $d_0, \ldots, d_n$ s.t. $\sum_{i=0}^{n} d_i = (\mathsf{s}_{int}(o_j) + \alpha_{o_j,s}) - (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e})$ and, for all $0 \leq s \leq n$, $d_s \in D_i(v_s)$.

To this aim, we guess a set of integers $\{\alpha'_{u,v} \mid (u,v) \in T_i\}$. Intuitively, $\alpha'_{u,v}$ is the number of times the solution path traverses (u,v). Since every time an edge is traversed a new token starts, each $\alpha'_{u,v}$ is bounded by the number of tokens, i.e., by $2 \cdot g \cdot |SV|$. Hence the binary encoding of $\alpha'_{u,v}$ can be generated in polynomial time. Then, we perform the following deterministic steps.

1. We consider the set $E' := \{(u,v) \in T_i \mid \alpha'_{u,v} > 0\}$ (subset of the edges of G), and we check whether it induces an (undirected) connected subgraph of G.
2. Check (a) $\sum_{(u,v)\in E'} \alpha'_{u,v} = \sum_{(v,w)\in E'} \alpha'_{v,w}$ for $v \in V_i \setminus \{v_0, v_n\}$ (b) $\sum_{(u,v_0)\in E'} \alpha'_{u,v_0} = \sum_{(v_0,w)\in E'} \alpha'_{v_0,w} - 1$ (c) $\sum_{(u,v_n)\in E'} \alpha'_{u,v_n} = \sum_{(v_n,w)\in E'} \alpha'_{v_n,w} + 1$.
3. For all $v \in V_i \setminus \{v_0\}$, we define $y_v := \sum_{(u,v)\in E'} \alpha'_{u,v}$ (y_v is the number of times the solution path gets into v); moreover, $y_{v_0} := \sum_{(v_0,u)\in E'} \alpha'_{v_0,u}$.
4. We define the real non-negative variables z_v, for every $v \in V_i$ (z_v is the total waiting time of the path on the node v), subject to the following constraints: $a \cdot y_v \leq z_v \leq b \cdot y_v$, where $D_i(v) = [a,b]$ (the constraint is analogous for open intervals). Finally, we set $\sum_{v\in V_i} z_v = (\mathsf{s}_{int}(o_j) + \alpha_{o_j,s}) - (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e})$.

Steps 1. and 2. together check that the values $\alpha'_{u,v}$ for the edges specify a directed *Eulerian path* from v_0 to v_n in a multigraph. The following indeed holds.

Theorem 2 ([9]). *Let $G' = (V', E')$ be a directed multigraph (E' is a multiset). G has a (directed) Eulerian path from v_0 to v_n iff: 1) the undirected version of G' is connected; 2) $|\{(u,v) \in E'\}| = |\{(v,w) \in E'\}|$, for all $v \in V' \setminus \{v_0, v_n\}$; 3) $|\{(u,v_0) \in E'\}| = |\{(v_0,w) \in E'\}| - 1$; 4) $|\{(u,v_n) \in E'\}| = |\{(v_n,w) \in E'\}| + 1$.*

Steps 3. and 4. evaluate the waiting times of the path in some vertex v with duration interval $[a,b]$. If the solution path visits v y_v times, then each single visit must take at least a and at most b units of time. Hence, the overall visitation time is in between $a \cdot y_v$ and $b \cdot y_v$. Vice versa, if the total visitation time is in between $a \cdot y_v$ and $b \cdot y_v$, then it can be split into y_v intervals each falling into $[a,b]$.

The algorithm concludes by solving the linear program given by the variables $\alpha_{o_i,s}$ and $\alpha_{o_i,e}$ for each quantifier $o_i[x_i = v_i]$, and, for each pair of adjacent tokens in the same timeline, say for x_i, for each $v \in V_i$, the variables z_v subject to their constraints. However, in order to conform to linear programming, we have to replace all strict inequalities with non-strict ones. All constraints involving strict inequalities we have written so far are of (or can easily be converted into) the following forms: $\xi s < \eta q + k$ or $\xi s > \eta q + k$, where s and q are variables, and ξ, η, k are constants. We replace them, respectively, by $\xi s - \eta q - k + \beta_t \leq 0$ and $\xi s - \eta q - k - \beta_t \geq 0$, where β_t is an additional fresh non-negative variable, which is *local* to a single constraint. The original inequality and the new one are equivalent if and only if β_t is a small enough *positive* number. Moreover, we add another non-negative variable, say r, which is subject to a constraint $r \leq \beta_t$, for

each of the introduced variables β_t (i.e., r is less than or equal to the *minimum* of all β_t's). Finally, we maximize the value of r when solving the linear program. We have that $\max r > 0$ if and only if there is an admissible solution where the values of all β_t's are positive (and thus the original strict inequalities hold true).

We conclude with the next result (that holds even for a single state variable).

Theorem 3. *Let P be a TP domain with trigger-less rules only. Checking the existence of a plan for P is an **NP**-complete problem.*

Proof. For **NP**-hardness, there is a reduction from the problem of the existence of a Hamiltonian path in a directed graph G. Given $G=(V,E)$, with $|V|=n$, we define $x=(V,E,D_x)$, where $D_x(v)=[1,1]$ for each $v \in V$. We add the following trigger-less rules, one for each $v \in V$: $\top \rightarrow \exists o[x = v].o \geq^{\mathsf{s}}_{[0,n-1]} 0$. The rule for $v \in V$ requires that there is a token $(x, v, 1)$ on the timeline for x, starting no later than $n-1$. G has a Hamiltonian path iff there is a plan for the planning problem. □

5 Conclusions and Future Work

In this paper, we studied TP restricted to trigger-less rules, over dense temporal domains, proving its **NP**-completeness. We analyzed this case as the unrestricted version of the problem is known to be undecidable [3].

In future work, we will focus on decidability and complexity of intermediate fragments of TP, namely, where forms of synchronization rules in between the general ones and trigger-less are considered. For instance, we may allow only rules for future, and/or non-singular intervals in the atoms of trigger-rules.

References

1. Alur, R., Dill, D.: A theory of timed automata. Th. Comp. Sci. **126**, 183–235 (1994)
2. Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., Smith, D.: EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In: ICKEPS (2012)
3. Bozzelli, L., Molinari, A., Montanari, A., Peron, A.: Decidability and Complexity of Timeline-based Planning over Dense Temporal Domains. In: KR (2018), extended version at https://www.uniud.it/it/ateneo-uniud/ateneo-uniud-organizzazione/dipartimenti/dmif/assets/preprints/1-2018-molinari
4. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A., Policella, N.: An Innovative Product for Space Mission Planning: A Posteriori Evaluation. In: ICAPS. pp. 57–64 (2007)
5. Chien, S., Tran, D., Rabideau, G., Schaffer, S., Mandl, D., Frye, S.: Timeline-based space operations scheduling with external constraints. In: ICAPS. pp. 34–41 (2010)
6. Cialdea Mayer, M., Orlandini, A., Umbrico, A.: Planning and Execution with Flexible Timelines: a Formal Account. Acta Informatica **53**(6–8), 649–680 (2016)
7. Gigante, N., Montanari, A., Cialdea M., M., Orlandini, A.: Timelines are expressive enough to capture action-based temporal planning. In: TIME. pp. 100–109 (2016)
8. Gigante, N., Montanari, A., Cialdea Mayer, M., Orlandini, A.: Complexity of timeline-based planning. In: ICAPS. pp. 116–124 (2017)
9. Jungnickel, D.: Graphs, Networks and Algorithms. Springer (2013)
10. Muscettola, N.: HSTS: Integrating Planning and Scheduling. In: Intelligent Scheduling, pp. 169–212. Morgan Kaufmann (1994)