Decidability and Complexity of Action-Based Temporal Planning over Dense Time

Nicola Gigante,[1] Andrea Micheli,[2] Angelo Montanari,[1] Enrico Scala[3]
[1]University of Udine, Italy
{nicola.gigante, angelo.montanari}@uniud.it
[2]Fondazione Bruno Kessler, Trento, Italy
amicheli@fbk.eu
[3]University of Brescia, Italy
enrico.scala@unibs.it

Abstract

This paper studies the computational complexity of temporal planning, as represented by PDDL 2.1, interpreted over *dense* time. When time is considered discrete, the problem is known to be EXPSPACE-complete. However, the official PDDL 2.1 semantics, and many implementations, interpret time as a dense domain. This work provides several results about the complexity of the problem, studying a few interesting cases: whether a minimum amount ε of separation between mutually exclusive events is given, in contrast to the separation being simply required to be non-zero, and whether or not actions are allowed to overlap already running instances of themselves. We prove the problem to be PSPACE-complete when self-overlap is forbidden, whereas, when allowed, it becomes EXPSPACE-complete with ε-separation and undecidable with non-zero separation. These results clarify the computational consequences of different choices in the definition of the PDDL 2.1 semantics, which were vague until now.

1 Introduction

Domain-independent planning (Ghallab, Nau, and Traverso 2016) is one of the classical fields of Artificial Intelligence and received considerable attention throughout the years. One of the most active research directions in this context is *temporal planning*, which represents and reasons about the flow of time explicitly. A popular modeling language for such problems is PDDL 2.1 (Fox and Long 2003), an action-centered formalism that extends classical planning by explicitly modeling the *duration* of actions. A temporal planning problem in PDDL 2.1 consists of looking for a sequence of actions that is not only *causally* executable (as in *classical* planning), but also *schedulable*, in accordance to given action duration constraints, along a timeline of unbounded length. Several planning systems (Coles et al. 2010; Eyerich, Mattmüller, and Röger 2012; Gerevini, Saetti, and Serina 2003; Rankooh and Ghassem-Sani 2015) as well as various international planning competitions (Vallati et al. 2015; Coles et al. 2012) adopt or have adopted PDDL 2.1 for the specification of the temporal planning problem.

Here, we study the computational complexity of PDDL 2.1 temporal planning problems over a *dense* temporal domain.

To the best of our knowledge, to date only Rintanen (2007), Cushing et al. (2007), and Cushing (2012) approached temporal planning from a theoretical point of view, focusing their attention, however, on a temporal model that is substantially discrete. In particular, Rintanen (2007) proves the problem to be EXPSPACE-complete over *discrete* time in the general case, and PSPACE-complete when actions are disallowed to self-overlap with already running instances of themselves. Such results apply to the dense setting if a specific ε is given as the minimum amount of time separating mutual exclusive (*mutex*) events, and if actions are given only a specific fixed duration, whereas PDDL 2.1 generally specifies actions with an interval of admissible values for the duration.

The computational complexity arising from using a dense temporal model with no ε value given upfront remains still poorly understood. It is worth noticing that the formal specification of PDDL 2.1 (Fox and Long 2003, Section 8) only requires mutex events to be separated by a non-zero amount of time; the idea of accepting an ε-separation value as input comes later in the text as an expedient to facilitate plan validation (Section 10 - Plan Validation). The very same authors do however admit that this ambiguity was at that time problematic and they did not find a definitive and principled way to account for it. Clarifying this aspect is relevant not only because it could be at times impractical to provide the right ε value upfront, but also because many planners do not use a discrete temporal model at all (Shin and Davis 2005; Coles et al. 2010); practice and theory behind temporal planning problems seem unnecessarily distant.

In order to work out these issues, this paper analyzes the computational complexity of temporal planning problems over dense time that takes into account, in a comprehensive manner, a number of different variants: the case with ε-separation, where an ε value of separation between mutex events is given upfront, and the case with non-zero separation, where mutex events are only required to not appear at the same time. Both cases are studied when either allowing or forbidding self-overlap of actions.

The outcomes can be summarized as follows: when self-overlap of actions is forbidden temporal planning over dense time is not harder than classical planning (PSPACE-complete), regardless of the mutex separation criterion. On

the other hand, allowing actions to self-overlap makes the problem harder: EXPSPACE-complete with ε-separation and, perhaps most surprisingly, *undecidable* in the non-zero separation case. We prove these results by studying the problems with and without self-overlapping separately. For the case with no self-overlap, we give a novel polynomial reduction to *updatable timed automata* (Alur and Dill 1994; Bouyer et al. 2004), which gives us the PSPACE upper bound for both the ε-separation and non-zero separation cases. For the case with self-overlap, we provide a reduction from two specific variants of the *corridor tiling problem* (van Emde Boas 1997), known to be EXPSPACE-complete and undecidable, to temporal planning problems with ε-separation and non-zero separation semantics, respectively. We work with actions with a non-fixed duration, thus extending the bound found by Rintanen (2007) to a more general setting. Table 1 summarizes the main results.

Outline We formally define the problem in Section 2. Then, Section 3 provides a complexity analysis of the problem when self-overlap of actions is forbidden, Section 4 when it is allowed. Section 5 surveys related work, and Section 6 concludes the paper with some final remarks.

2 Dense time temporal planning

Following Fox and Long (2003), this section introduces the temporal planning problem we are interested in. Our analysis focuses on the STRIPS fragment, and uses a set-theoretical representation (Ghallab, Nau, and Traverso 2004).

Definition 2.1 (Planning problem)**.** A planning problem is a tuple $\mathcal{P} = \langle P, A, I, G\rangle$, where P is a set of propositions, A is a set of durative actions, $I \subseteq P$ is the initial state, and $G \subseteq P$ is the goal condition. A snap (instantaneous) action is a tuple $h = \langle \mathrm{pre}(h), \mathrm{eff}^+(h), \mathrm{eff}^-(h)\rangle$, where $\mathrm{pre}(h) \subseteq P$ is the set of preconditions and $\mathrm{eff}^+(h), \mathrm{eff}^-(h) \subseteq P$ are two disjoint sets of *positive* and *negative* effects (we write $\mathrm{eff}(h)$ for $\mathrm{eff}^+(h) \cup \mathrm{eff}^-(h)$), respectively. A durative action is a tuple $a = \langle a_\vdash, a_\dashv, \mathrm{pre}^\leftrightarrow(a), [L_a, U_a]\rangle$, where $a_\vdash$ and $a_\dashv$ are the *start* and *end* snap actions, respectively, $\mathrm{pre}^\leftrightarrow(a) \subseteq P$ is the *over-all condition*, and $L_a \in \mathbb{Q}_{>0}$ and $U_a \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$ are the bounds on the action duration.

Definition 2.2 (Plan)**.** Let $\mathcal{P} = \langle P, A, I, G\rangle$ be a planning problem. A plan for $\mathcal{P}$ is a set of tuples $\pi = \{\langle a_1, t_1, d_1\rangle, \cdots, \langle a_n, t_n, d_n\rangle\}$, where, for each $1 \leq i \leq n$, $a_i \in A$ is a durative action, $t_i \in \mathbb{Q}_{\geq 0}$ is the start time of a_i, and $d_i \in \mathbb{Q}_{>0}$ is its duration.

The semantics of temporal plans is thoroughly defined by Fox and Long (2003). For space reasons, we only informally report the aspects that are shared among all planners using PDDL2.1, and describe more rigorously the semantics variants that are the main object of our study. Intuitively, a *solution plan* π for a planning problem $\mathcal{P}$ can be simulated by applying the effects of all snap actions $a_\vdash$ at time t and all effects of $a_\dashv$ at time $t + d$ for all $\langle a, t, d\rangle \in \pi$. The induced trace assigns a value to each predicate at each time and must be such that the preconditions of all snap actions and the over-all condition of all actions in π are satisfied, the duration of actions respects the given bounds, and the plan execution yields a final state with no pending running actions and where the goal condition holds.

Definition 2.2 admits more than one action starting or ending at the same time. However, PDDL 2.1 imposes all snap actions to not interfere with each other.

Definition 2.3 (Mutex snap actions)**.** Two snap actions a and b are *mutually exclusive* (mutex), denoted as $a \not\downarrow b$, if either $\mathrm{pre}(a) \cap \mathrm{eff}(b) \neq \varnothing$, or $\mathrm{pre}(b) \cap \mathrm{eff}(a) \neq \varnothing$, or $\mathrm{eff}^+(a) \cap \mathrm{eff}^-(b) \neq \varnothing$, or $\mathrm{eff}^+(b) \cap \mathrm{eff}^-(a) \neq \varnothing$.

Intuitively, we constrain mutex snap actions to not appear at the same time in a plan. How this requirement is defined can differ depending on whether we want to enforce a minimum amount ε of separation between any pair of mutex snap actions or whether any separation suffices. Two different notions of *plan validity* can be defined consequently.

Definition 2.4 (Non-zero separation plan validity)**.**
A solution plan $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$ for a planning problem $\mathcal{P}$ is valid under *non-zero separation* if, for each $1 \leq i, j \leq n$, with $i \neq j$, $a_i \not\downarrow a_j$ implies $t_i \neq t_j$.

Definition 2.5 (ε-separation plan validity)**.** Let $\varepsilon \in \mathbb{Q}_{>0}$. A solution plan $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$ for a planning problem $\mathcal{P}$ is valid under *ε-separation* if, for all $1 \leq i, j \leq n$, with $i \neq j$, $a_i \not\downarrow a_j$ implies $|t_i - t_j| \geq \varepsilon$.

A further distinction has important consequences from a computational point of view, both in discrete time domains (Rintanen 2007) and in dense ones.

Definition 2.6 (Actions self-overlap)**.**
Given a planning problem $\mathcal{P} = \langle P, A, I, G\rangle$ and a plan $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$ for $\mathcal{P}$, an action $a \in A$ is said to *self-overlap* in π if there exist any $1 \leq i, j \leq n$ such that $a = a_i = a_j$ and $t_i \leq t_j \leq t_i + d_i$.

This paper studies the computational complexity of deciding whether a solution plan exists for a given planning problem $\mathcal{P}$, under both the above-defined notions of plan validity, both allowing and disallowing actions to self-overlap.

3 Forbidding self-overlap of actions

In this section, we determine the complexity of temporal planning over dense time when actions are not allowed to overlap with themselves. Since temporal planning extends classical planning, which is known to be PSPACE-complete (Bylander 1994), it is trivially PSPACE-hard. We prove that the problem can be solved in polynomial space by encoding it into a particular kind of *timed automata*.

A *timed automaton* (TA) is a finite automaton augmented with a set of *clocks* (Alur and Dill 1994), which explicitly track the flow of *time*. Each transition in a TA may include temporal constraints, called *guards*, that disable the transition if not satisfied by the current clock values. Each transition may also include *clock resets* that cause specified clocks to be reset to zero whenever the transition is taken. Finally, each location may include an *invariant*—that is, a constraint specifying the conditions under which the automaton may stay in that location. Here, we use the more general *updatable timed automata* (Bouyer et al. 2004), that allow clocks to be reset to any constant rational value, not only to zero.

	ε-separation ($L_a = U_a$)	ε-separation ($[L_a, U_a]$)	non-zero separation ($[L_a, U_a]$)
w/o self-overlap	PSPACE-complete	**PSPACE-complete**	
self-overlap	EXPSPACE-complete	**EXPSPACE-complete**	**undecidable**

Table 1: Complexity bounds for the different considered cases. $L_a = U_a$ is the case where duration is fixed, while $[L_a, U_a]$ is the case where the duration can be any value in the interval. Bold font indicates novel results.

Given a set $\mathcal{X}$ of elements called *clocks*, the set $\mathcal{C}(\mathcal{X})$ of *constraints* over the clocks in $\mathcal{X}$ contains conjunctions of constraints of the form $x \bowtie k$ or $y - x \bowtie k$, where $x, y \in \mathcal{X}$, $k \in \mathbb{Q}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. The set $\mathcal{U}(\mathcal{X})$ of *updates* on the clocks in $\mathcal{X}$ is the set of conjunctions of basic statements of the form $x := k$, with $x \in \mathcal{X}$ and $k \in \mathbb{Q}$.

Definition 3.1 (Updatable timed automaton)**.** An *updatable timed automaton* is a tuple $\mathcal{T} = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, \mathcal{I} \rangle$ where: Σ is the alphabet; $\mathcal{L}$ is the finite set of locations; $\ell_0 \in \mathcal{L}$ is the initial location; $\mathcal{X}$ is the finite set of *clocks*; $\Delta \subseteq \mathcal{L} \times \mathcal{C}(\mathcal{X}) \times \Sigma \times 2^{\mathcal{U}(\mathcal{X})} \times \mathcal{L}$ is the transition relation; $\mathcal{I} : \mathcal{L} \to \mathcal{C}(\mathcal{X})$ maps each location to its *invariant*.

The full semantics of TAs has been defined in details by Alur and Dill (1994). For the sake of this paper, we only informally define the reachability problem. The reachability problem for a TA $\mathcal{T} = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, \mathcal{I} \rangle$ and an objective $\mathcal{G} \subseteq \mathcal{L}$ consists in deciding whether there exists an execution of $\mathcal{T}$ that starts from ℓ_0 and ends in a location $\ell^* \in \mathcal{G}$. The problem is PSPACE-complete with standard resets and constant updates, too (Bouyer et al. 2004).

To simplify the exposition, we use the concept of *urgent* locations, *i.e.*, locations where time is stationary. Urgent locations are encoded adding an extra clock that is reset to zero in each incoming transition and forced to be zero in the location invariant. The problem complexity remains unchanged.

The intuition behind the encoding comes from decision-epoch planners (Do and Kambhampati 2003): at each step, the automata can either execute a set of snap actions (by checking the preconditions and applying the effects) or decide to wait a certain amount of time (delta-transition). Crucially, to keep the size of the resulting automaton polynomial, there cannot be one location for each propositional state of the planning problem. Instead, we symbolically encode the predicates using clocks that maintain a truth value recognizable in the guards of the automaton. We use constant updates to apply the effects of actions on such clocks. As we will see, the encoding can be adapted to support either the ε-separation or the non-zero separation semantics, hence proving the complexity of both cases, without self-overlap.

Theorem 3.1. *Temporal planning over dense time, without self-overlap of actions, is PSPACE-complete.*

Proof. As anticipated, PSPACE-hardness is trivial, as temporal planning includes classical planning as a special-case and the latter is PSPACE-complete (Bylander 1994).

Let $\mathcal{P} = \langle P, A, I, G \rangle$ be a temporal planning problem, and let $M = |A|$, $N = |P|$, $A = \{a_1, \ldots, a_M\}$, and $P = \{p_1, \ldots, p_N\}$. We prove that the problem belongs to PSPACE by encoding any such temporal planning problem $\mathcal{P}$ into an equivalent TA $\mathcal{T}[\mathcal{P}] = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, \mathcal{I} \rangle$, of size polynomial in the size of $\mathcal{P}$, with a location ℓ^* such that a solution plan for $\mathcal{P}$ exists iff ℓ^* is reachable from ℓ_0.

The automaton is defined as follows:

1. the alphabet is an arbitrary singleton $\Sigma = \{\textvisiblespace\}$; the words accepted by the automaton are irrelevant in this encoding;
2. the set of locations $\mathcal{L}$ is composed of the three locations ℓ_0 (the *initial* location), ℓ^* (the *goal* location), ℓ_δ (the *main* location), of a set $\{s_0, s_1, \ldots, s_{M+N}\}$ of $M + N$ *state decoding* locations, of a set $\{d_0, d_1, \ldots, d_{2M}\}$ of $2M$ *decision making* locations and of a set $\{e_0, e_1, \ldots, e_{2M}\}$ of $2M$ *execution* locations. All locations but ℓ_δ are *urgent*;
3. the set of clock variables $\mathcal{X}$ is composed of:
   - a clock $\mathtt{c}\delta$, called the *global* clock;
   - a clock $\mathtt{cp_i}$ for each $p_i \in P$;
   - five clocks, $\mathtt{cx_{a\vdash}}$, $\mathtt{cx_{a\dashv}}$, $\mathtt{cr_a}$, $\mathtt{cs_a}$, $\mathtt{ce_a}$, for each $a \in A$.
4. no invariant conditions are needed (except the ones that are implicitly defined by urgent states), hence $\mathcal{I} = \varnothing$.

We now define the transition relation Δ, explaining how the automaton works and how the locations defined above are connected. A schema of the construction can be seen in Figure 1. The main location ℓ_δ is the only location where time can pass, all the other locations being urgent. The initial location immediately transitions to the main location, setting some of the $\mathtt{cp_i}$ clocks to one, depending on the *initial state*:

$$\langle s_0, \top, \textvisiblespace, \{\mathtt{cp_i} := 1 \mid p_i \in I\}, \ell_\delta \rangle \in \Delta$$

Let $B = \{\mathtt{cp_i} \mid p_i \in P\} \cup \{\mathtt{cr_a} \mid a \in A\}$ be a subset of the clocks, that we call the *binary* clocks, and let us denote them, with an arbitrary order, as $B = \{\mathtt{b}_1, \ldots, \mathtt{b}_{M+N}\}$. The initial transition establishes an invariant that is kept by construction throughout the automaton: when the execution *enters* ℓ_δ, it holds that $\mathtt{c}\delta = 0$ and, for $\mathtt{b}_k \in B$, either $\mathtt{b}_k = 0$ or $\mathtt{b}_k = 1$. Since the clocks advance together while the automaton stays in the main location, the *difference* between any $\mathtt{b}_k \in B$ and $\mathtt{c}\delta$ will always be either zero or one, accordingly. In this way, these clocks can be used as *binary variables*, and, in particular, the $\mathtt{cp_i}$ clocks can be used to represent the propositional state of the planning problem propositions.

In the main location, the automaton can decide at any time to make a transition to the first *state decoding* location s_0:

$$\langle \ell_\delta, \top, \textvisiblespace, \varnothing, s_0 \rangle \in \Delta$$

The s_0 location starts a chain of $M + N$ locations, that goes up to s_{M+N}, called the *state decoding* path. Its purpose is to reset each clock $\mathtt{b}_k \in B$ to a binary value, to allow the subsequent transitions to use such clocks as binary variables: at each step of the state decoding path, say in the transition

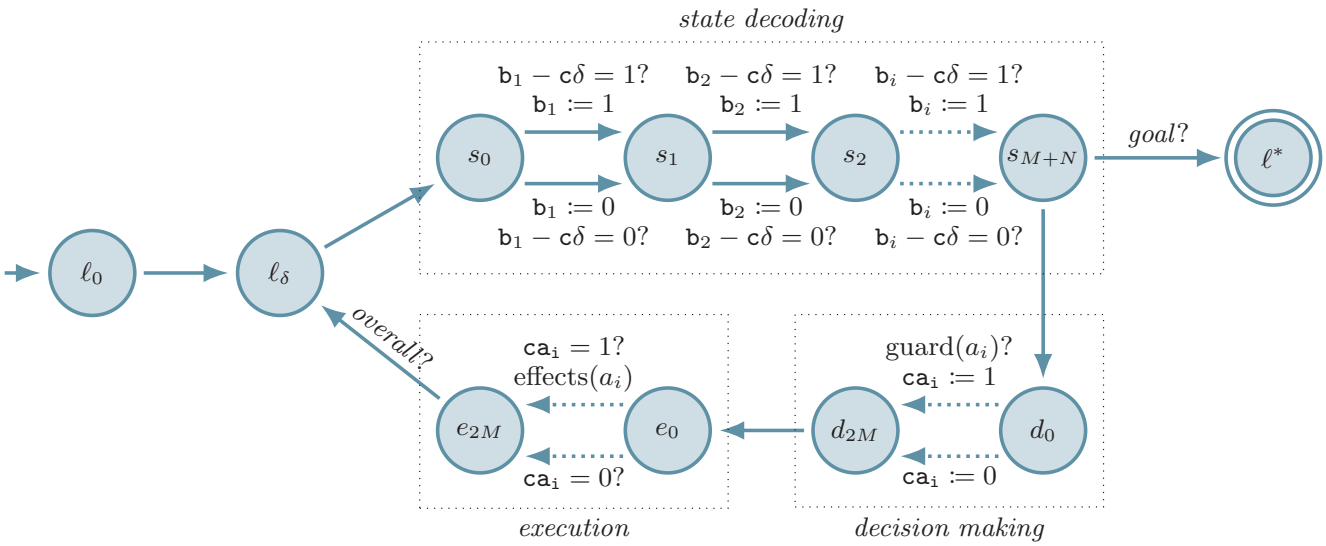

Figure 1: The *updatable timed automaton* used in Theorem 3.1; a_i indicates any snap action (*i.e.*, either $a_\vdash$ or $a_\dashv$ for some $a \in A$).

from s_{k-1} to s_k, the $\mathtt{b}_k$ clock is reset to zero if $\mathtt{b}_k - \mathtt{c}\delta = 0$, and it is reset to one if $\mathtt{b}_k - \mathtt{c}\delta = 1$, hence we have:

$$\langle s_{k-1}, \mathtt{b}_k - \mathtt{c}\delta = 0, \llcorner, \{\mathtt{b}_k := 0\}, s_k \rangle \in \Delta$$
$$\langle s_{k-1}, \mathtt{b}_k - \mathtt{c}\delta = 1, \llcorner, \{\mathtt{b}_k := 1\}, s_k \rangle \in \Delta$$

for all $k \in \{1, \ldots, M+N\}$. By the time the automaton reaches s_{M+N}, the value of the p_i clocks directly corresponds to the binary values of the planning problem propositions at the current time in the encoded plan. Hence, the guards of later transitions can directly encode any propositional formula over those propositions. The other binary clocks $\mathtt{cr}_\mathtt{a}$, instead, will be used to keep track of whether the a action is being executed (i.e. "running") at the current time.

The automaton traverses the state decoding path either at the start or the end of an action or when the execution ends because the goal was reached. In the former case, we move to the location d_0 unconditionally:

$$\langle s_{M+N}, \top, \llcorner, \varnothing, d_0 \rangle \in \Delta$$

Starting from d_0, the automaton can decide which subset of the snaps to execute in parallel by setting the $\mathtt{cx}_\mathtt{h}$ execution clock, for each snap h, to either 1 or to 0. This can be done polynomially by a construction analogous to the state decoding path. For each $a_k \in A$ we have the following transitions:

$$\langle d_{2k-2}, \text{guard}(a_{k\vdash}), \llcorner, \{\mathtt{cx}_{\mathtt{a}_{k\vdash}} := 1\}, d_{2k-1} \rangle \in \Delta$$
$$\langle d_{2k-2}, \top, \llcorner, \{\mathtt{cx}_{\mathtt{a}_{k\vdash}} := 0\}, d_{2k-1} \rangle \in \Delta$$
$$\langle d_{2k-1}, \text{guard}(a_{k\dashv}), \llcorner, \{\mathtt{cx}_{\mathtt{a}_{k\dashv}} := 1\}, d_{2k} \rangle \in \Delta$$
$$\langle d_{2k-1}, \top, \llcorner, \{\mathtt{cx}_{\mathtt{a}_{k\dashv}} := 0\}, d_{2k} \rangle \in \Delta$$

Intuitively, the automaton can either take a transition that sets to zero the clock relative to a snap action signal that such action is not to be executed at the current time, or take the transition checking the guard of the snap action and setting the clock to one. The guard of a starting snap action is:

$$\text{guard}(a_\vdash) = \bigwedge_{p_i \in \text{pre}(a_\vdash)} \mathtt{cp}_\mathtt{i} = 1 \wedge \mathtt{cr}_\mathtt{a} = 0 \wedge \text{sep}(a_\vdash)$$

where:

$$\text{sep}(a_\vdash) = \bigwedge_{b_\vdash \not\downarrow a_\vdash} \mathtt{cs}_\mathtt{b} > 0 \wedge \bigwedge_{b_\dashv \not\downarrow a_\vdash} \mathtt{ce}_\mathtt{b} > 0$$

The condition expressed by $\text{guard}(a_\vdash)$ checks that the preconditions of $a_\vdash$ are satisfied, and that the action is not already running. Then, $\text{sep}(a_\vdash)$ encodes the time separation between mutex snap actions. By checking that the corresponding clocks of each mutex snap actions are positive, we enforce the non-zero separation condition. By replacing the $\mathtt{cs}_\mathtt{b} > 0$ and $\mathtt{ce}_\mathtt{b} > 0$ conditions with $\mathtt{cs}_\mathtt{b} \geq \varepsilon$ and $\mathtt{ce}_\mathtt{b} \geq \varepsilon$, we can easily capture the ε-separation semantics.

The guard for the end of an action is very similar, but it checks that the action is actually already running, that the over-all conditions are satisfied and that the action duration is compatible with its duration constraints:

$$\text{guard}(a_\dashv) = \bigwedge_{p_i \in \text{pre}(a_\dashv)} \mathtt{cp}_\mathtt{i} = 1 \wedge \mathtt{cr}_\mathtt{a} = 1 \wedge \text{sep}(a_\dashv) \wedge \text{dur}(a_\dashv)$$

where:

$$\text{dur}(a_\dashv) = \mathtt{cs}_\mathtt{a} \geq L_a \wedge \mathtt{cs}_\mathtt{a} \leq U_a$$

At this point, the value of the $\mathtt{cx}_\mathtt{a}$ execution clocks is either 0 or 1 depending on whether the snap action a must be executed or not. To apply the effects we traverse another sequence of locations $e_0, \cdots, e_{2M}$ that apply the effects of each snap action if the relative execution clock is set to 1.

$$\langle d_{2M}, \top, \llcorner, \varnothing, e_0 \rangle \in \Delta$$
$$\langle d_{2k-2}, \mathtt{cx}_{\mathtt{a}_{k\vdash}} = 1, \llcorner, \text{effects}(a_\vdash), d_{2k-1} \rangle \in \Delta$$
$$\langle d_{2k-2}, \mathtt{cx}_{\mathtt{a}_{k\vdash}} = 0, \llcorner, \varnothing, d_{2k-1} \rangle \in \Delta$$
$$\langle d_{2k-1}, \mathtt{cx}_{\mathtt{a}_{k\dashv}} = 1, \llcorner, \text{effects}(a_\dashv), d_{2k} \rangle \in \Delta$$
$$\langle d_{2k-1}, \mathtt{cx}_{\mathtt{a}_{k\dashv}} = 0, \llcorner, \varnothing, d_{2k} \rangle \in \Delta$$

Note that $\text{guard}(\cdot)$ and $\text{effects}(\cdot)$ use the $\mathtt{cp}_\mathtt{i}$ clocks to, respectively, enforce the preconditions and execute the effects

of the snap action at hand. The effects of the start of an action are defined as follows:

$$\begin{aligned}\text{effects}(a_\vdash) = &\{\mathtt{cr_a} \coloneqq 1, \mathtt{cs_a} \coloneqq 0\} \\ &\cup \{\mathtt{cp_i} \coloneqq 1 \mid p_i \in \text{eff}^+(a_\vdash)\} \\ &\cup \{\mathtt{cp_i} \coloneqq 0 \mid p_i \in \text{eff}^-(a_\vdash)\}\end{aligned}$$

Hence, $\text{effects}(a_\vdash)$ sets $\mathtt{cr_a}$ to record that a is executing and resets the $\mathtt{cs_a}$ clock, which is used to record the time passed by the last time the a action was started. It also sets the binary clocks $\mathtt{cp_i}$ according to the positive or negative effects of the action. The effects for the *end* of actions, $\text{effects}(a_\dashv)$, are defined similarly, but the $\mathtt{ce_a}$ clock is reset instead of $\mathtt{cs_a}$, to record the time since the last time the a action ended, and $\mathtt{cr_a}$ is set to zero.

When all effects have been applied, in the e_{2M} location, we can return in location ℓ_δ by resetting the clock $\mathtt{c}\delta$. In this way, we reset the invariant for binary clocks, and the automaton can decide how much time to wait until the next snap action is executed. In this transition we perform a final check to ensure that the over-all conditions of each running action are respected:

$$\langle e_{2M}, \bigwedge_{a \in A} \text{oc}(a), \llcorner, \{\mathtt{c}\delta \coloneqq 0\}, \ell_\delta \rangle \in \Delta$$

where:

$$\text{oc}(a) = \bigwedge_{p_i \in \text{pre}^{\leftrightarrow}(a)} \mathtt{cr_a} - \mathtt{cp_i} \leq 0$$

The $\text{oc}(a)$ formula, intuitively checks that if a is running (*i.e.*, $\mathtt{cr_a} = 1$), then the over-all conditions of a must be true (*i.e.*, each $\mathtt{cp_i}$ must be 1 for each $p_i \in \text{pre}^{\leftrightarrow}(a)$). This implication is captured by the difference above that is false only when $\mathtt{cr_a} = 1$ and $\mathtt{cp_i} = 0$. This ensures that, in any accepted run, the guard cannot be false due to a condition being violated.

Finally, if the automaton takes the state decoding path when the goal condition is satisfied and no action is running, then it can transition to the goal state, reaching its objective:

$$\langle s_{M+N}, \bigwedge_{p_i \in G} \mathtt{cp_i} = 1 \wedge \bigwedge_{a \in A} \mathtt{cr_a} = 0, \llcorner, \varnothing, \ell^* \rangle \in \Delta$$

This completes the definition of the $\mathcal{T}[\mathcal{P}]$ automaton. The number of locations and transitions is easily seen to be polynomial in the size of $\mathcal{P}$, and it can be checked that ℓ^* is reachable from ℓ_0 if and only if $\mathcal{P}$ admits a solution plan. □

4 Allowing self-overlap of actions

This section focuses on the problem of dense-time temporal planning in the case where actions are allowed to overlap with themselves. Here, the distinction between the ε-separation and non-zero separation semantics play an important role. Indeed, the problem turns out to be EXPSPACE-complete in the former case and *undecidable* in the latter.

The two results are based on reductions from a well-known class of combinatorial problems known as *tiling problems*. For $n \geq 1$, let $[n]$ denote the set $\{1 \dots n\}$.

Definition 4.1 (Tiling structures)**.** A *tiling structure* is a tuple $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$, where T is a finite set of elements called *tiles*, $t_0 \in T$ and $t_* \in T$ are, respectively, the *initial* and *final* tiles, and $H, V \subseteq T \times T$, are the *horizontal* and *vertical adjacency relations*.

Given some $n > 0$, an *n-tiling* for a tiling structure $\mathcal{T}$ is a function $f : [n] \times [h] \to T$, for some $h > 0$, mapping any pair $(i, j) \in [n] \times [h]$ to a tile $f(i, j) \in T$ such that:

1. $f(0, 0) = t_0$
2. $f(n, h) = t^*$
3. for all $x \in [n-1]$ and $y \in [h]$, $f(x, y)\ H\ f(x+1, y)$
4. for all $x \in [n]$ and $y \in [h-1]$, $f(x, y)\ V\ f(x, y+1)$

Several interesting combinatorial problems can be defined, with complexities ranging from NP to highly undecidable, depending on which portion of the plane we are asked to tile. The resulting *tiling problems* have been used extensively as a tool for reductions in logics and combinatorics. A detailed survey on the topic is provided by van Emde Boas (1997).

The *unbounded corridor tiling problem* asks whether there exists $n > 0$ such that a given tiling structure $\mathcal{T}$ admits an n-tiling. The problem can be shown to be undecidable by a direct reduction from the halting problem of Turing machines.

By restricting the width of the corridor beforehand we obtain decidable problems. Given a tiling structure $\mathcal{T}$ and an $n > 0$, the *exponential corridor tiling problem* asks whether $\mathcal{T}$ admits an m-tiling for some $m \leq n$. Here, *exponential* refer to the fact that n is encoded in binary, hence the upper bound on the corridor width is exponential in the size of the input. For this reason, the problem can be shown to be EXPSPACE-complete (as opposed to the PSPACE-complete *corridor tiling* problem shown by van Emde Boas).

Intuitively, we reduce temporal planning to tiling problems, encoding tilings into plans. Each durative action represents a tile, and the propositional state remembers the tiles of previous steps. Actions' preconditions ensure the tiles satisfy the tiling structure. The reduction from the unbounded variant employs an unbounded number of self-overlaps, which is the source of undecidability in the non-zero separation case.

Theorem 4.1 (Undecidability)**.** *Temporal planning over dense time, with* non-zero separation *semantics and* self-overlap *of actions, is* undecidable*.*

Proof. The proof goes by reduction from the aforementioned *unbounded corridor tiling problem*. Given a tiling structure $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$, we can build a temporal planning problem $\mathcal{P} = \langle P, A, I, G \rangle$ that admits a solution plan if and only if $\mathcal{T}$ admits a tiling.[1] The problem encodes the tiling structure in such a way that a plan for $\mathcal{P}$ describes a tiling.

The problem is built upon the following propositions P:

- a set $\{\tau_0, \dots, \tau_{n-1}\}$ of $n = \lceil \log_2(|T|) \rceil$ propositions, whose truth value evolves to represent the tile placed at each position of the tiling, in a row-major layout;
- a set $\{\pi_0, \dots, \pi_{n-1}\}$ of $n = \lceil \log_2(|T|) \rceil$, that for each tile represent the one placed above it in the previous row;

[1] For ease of exposition, we use *conditional effects*, and allow the preconditions of actions to be given as generic boolean formulas. This can be reduced to the simple syntax of Definition 2.1 by paying an exponential size increase, in general, but the formulas used here all have fixed depth, ensuring the increase in size is only polynomial.

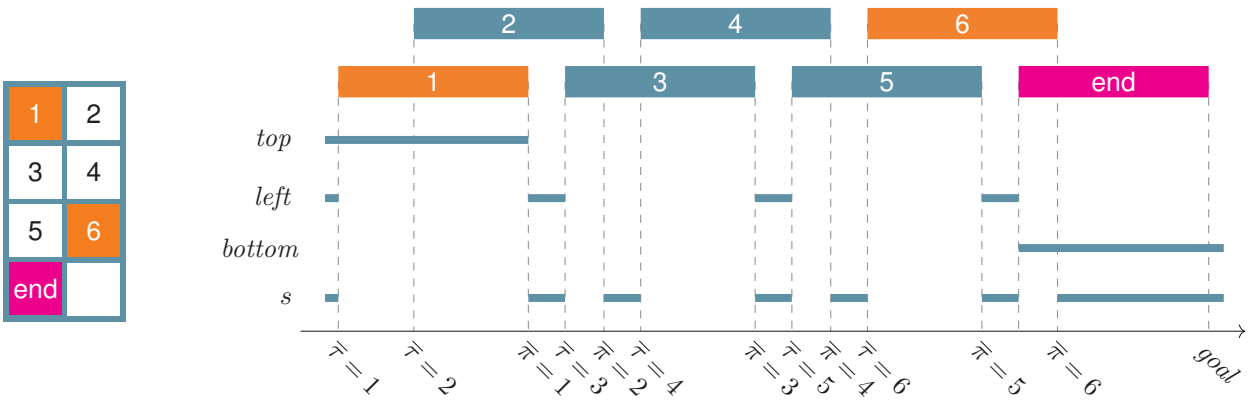

Figure 2: Depiction of the proof of Theorem 4.1. On the left, a rectangular tiling of six tiles $\{1, \ldots, 6\}$, with $t_0 = 1$ and $t^* = 6$. On the right, the corresponding temporal plan, with the executed actions on top, and the values of auxiliary propositions below.

- three propositions *top*, *bottom*, and *left* used to track whether the current time point belongs, respectively, to the first or the last row, or the leftmost column of the tiling;
- a flag bit s (for ***start***), with a specific role described below;
- a flag bit g that indicates the goal of the problem.

Then, the problem includes an action a^t for each $t \in T$, and an additional action *end*. All the actions have a fixed duration of *one time unit*. To describe their preconditions and effects, we need a bit of notation. The propositions τ_i and π_i represents tiles in T by adopting a binary encoding, hence for each $t \in T$, a formula $\overline{\tau} = t$ (resp., $\overline{\pi} = t$) can be defined as a simple conjunction stating the truth value of the τ_i (resp., π_i) fluents corresponding to t. Similarly, the shorthands $\overline{\tau} := t$ and $\overline{\pi} := t$ are used to denote the effect of setting the τ_i and π_i fluents to the tuple of values corresponding to t. Moreover, we use the notation $(\overline{\tau}, t) \in H$ and $(\overline{\pi}, t) \in V$ to mean that the current values of the τ_i and π_i fluents, respectively, are related to t by the H and V relations.

Formally, they can be written as follows:

$$(\overline{\tau}, t) \in H \leftrightarrow \bigvee_{\substack{t' \in T \\ (t', t) \in H}} \overline{\tau} = t' \qquad (\overline{\pi}, t) \in V \leftrightarrow \bigvee_{\substack{t' \in T \\ (t', t) \in V}} \overline{\pi} = t'$$

With this notation in place, the preconditions and effects of the problem's actions are defined as follows:

$$\begin{aligned}
\mathrm{pre}(a^t{}_{\vdash}) &= \neg \mathit{bottom} \\
&\quad \wedge \neg(\mathit{top} \wedge \mathit{left}) \qquad \leftarrow \text{except for } a^{t_0} \\
&\quad \wedge \neg \mathit{top} \rightarrow s \\
&\quad \wedge \neg \mathit{top} \rightarrow (\overline{\pi}, t) \in V \\
&\quad \wedge \neg \mathit{left} \rightarrow (\overline{\tau}, t) \in H \\
\mathrm{eff}(a^t{}_{\vdash}) &= \neg s \wedge \neg \mathit{left} \wedge \overline{\tau} := t \\
\mathrm{pre}(a^t{}_{\dashv}) &= \neg \mathit{bottom} \rightarrow \neg s \\
\mathrm{eff}(a^t{}_{\dashv}) &= s \wedge \neg \mathit{top} \wedge \overline{\pi} := t \\
&\quad \wedge \text{ when at-start}(\mathit{left}) \text{ then } \mathit{left}
\end{aligned}$$

Then, the *end* action is:

$$\begin{aligned}
\mathrm{pre}(\mathit{end}_{\vdash}) &= \mathit{left} \\
\mathrm{eff}(\mathit{end}_{\vdash}) &= \mathit{bottom} \\
\mathrm{pre}(\mathit{end}_{\dashv}) &= (\overline{\pi} := t^*) \\
\mathrm{eff}(\mathit{end}_{\dashv}) &= g
\end{aligned}$$

In the initial state, s, *top* and *left* hold, and an arbitrary value is held by $\overline{\tau}$ and $\overline{\pi}$. The goal requires g to hold.

The actions above define how a solution plan for the problem represents a valid tiling. An example plan representing a small tiled rectangle is depicted in Figure 2. Each tile of a row is determined by the effects of the *start* of an action, which sets the τ_i fluents accordingly. As soon as the first action ends, $\neg \mathit{top}$ is set, effectively starting the second row, and determining the number of columns as the number of the actions started until that point. The density of the time domain here plays a key role, since any arbitrary number of actions can be started in one time unit, during the execution of the first action. The self-overlap is also crucial, since there is no way to know in advance how many concurrent actions (how many columns) will be needed. After $\neg \mathit{top}$ is set, a strict alternation between starts and ends of actions is enforced by requiring s to hold at the start and $\neg s$ to hold at the end of each action, unless the bottom row is reached. This ensures that the number of actions of each row is the same as the one chosen on the first row. When a complete tiling has been built, the *end* action can be executed to signal the termination of the tiling. The start of *end* sets the *bottom* proposition, and since no other action can be started if *bottom* holds, this effectively stops the construction, leaving just the time for the started actions of the last row to end. The goal condition is satisfied as soon as the plan manages to start *end*. It can be checked that the plans obtained in this way correctly represent a rectangular grid of tiles.

It remains to check that such a grid actually corresponds to a correct tiling. The *left* proposition holds at any position that belongs to the first column: it is set in the initial state and cleared at the start of any action, but the end of an action will set it again if it was set when the action started. In this way the first action of each row sets *left* again when it ends, ready for the start of the first action of the subsequent row.

With this machinery in place, we enforce the adjacency

relations, and confirm that the plan encoded in this way represents a correct tiling, as per Definition 4.1:

1. the initial tiling t_0 is the only one that can be placed first, since only the start of a^{t_0} does not require $\neg(top \wedge left)$;
2. the final tiling t^* is the only one that can be placed last, since $\mathrm{pre}(end_\dashv)$ requires that $\overline{\pi} := t^*$;
3. since the τ_i fluents are set to the value of the current tile, the starting precondition of each action a^t ensures the horizontal adjacency relation, by requiring that $(\tau, t) \in H$, excepting for the first column, where *left* holds;
4. because of the strict alternation between starts and ends, the start of each action is preceded by the end of the one at the corresponding position in the previous row of the tiling; the π_i fluents, which are set by the end of each action, hence represent the tile placed exactly above the current position; in this way, the vertical adjacency relation can be enforced at the start of each action by requiring that $(\pi, t) \in V$, excepting for the first row, where *top* holds.

It can be seen that such a planning problem admits a solution plan iff the original tiling structure admits a tiling. □

The above argument can be adapted to the *ε-separation* semantics, reducing from the easier corridor tiling problem.

Theorem 4.2. *Temporal planning over dense time, under* ε-separation *semantics, and with* self-overlap *of actions, is* EXPSPACE*-complete.*

Proof. Let $\mathcal{P} = (P, A, I, G)$ be a dense-time temporal planning problem. An equivalent problem $\mathcal{P}' = (P', A', I', G')$ can be obtained, of exponential size $|\mathcal{P}'| \in \mathcal{O}(2^{|\mathcal{P}|})$, such that $\mathcal{P}$ has a solution plan admitting self-overlap of actions iff $\mathcal{P}'$ admits a plan without any self-overlap. Then, $\mathcal{P}'$ in turn can be solved in space polynomial in $|\mathcal{P}'|$, hence exponential in $|\mathcal{P}|$, by Theorem 3.1. If D_{max} is the maximum duration allowed for any action in A, then the $\mathcal{P}'$ problem can be obtained by duplicating each action $a \in A$ into k copies $a_1, \ldots, a_k$, where $k = D_{max}/\varepsilon$, which corresponds to the maximum number of overlapping instances of the same action, noticing that the start of an action is mutex with itself.

Hence we showed that the problem belongs to EXPSPACE. To show the EXPSPACE-hardness, we proceed by reduction from the *exponential corridor tiling problem*. Given a tiling structure $\mathcal{T}$ and an integer $n > 0$, the structure can be encoded into a temporal planning problem $\mathcal{P}$ that admits a solution plan if and only if $\mathcal{T}$ admits an m-tiling for some $m \leq n$. The encoding is the same as the one employed in the proof of Theorem 4.1, hence we do not repeat it here. However, since the maximum number of columns in the tiling is now known in advance, the resulting planning problem can be solved with ε-separation by choosing a suitable value for ε. At first, note that all snap actions produced in the reduction are *mutex*, since their effects and preconditions share at least the s flag bit. Then, at worst, the action corresponding to the first tile of the row has to contain the end of the m actions of the previous row, and the start of the other $m - 1$ tiles of the row. This divides the unit time interval of the action into $2m - 2$ subintervals, hence $\varepsilon = 1/(2n - 2)$ ensures enough granularity for any tiling with only and at most n columns. □

5 Related works

Other works in the literature have analyzed temporal planning from a theoretical perspective before. Cushing (2012) discusses at length the philosophical subtleties of some semantic aspects. He discusses the impact of the non-zero *vs.* ε-separation issue in PDDL 2.1. However, he diverges significantly from the PDDL 2.1 modeling formulation, and gives no complexity results. Relevant to our discussion is also the work by Shin and Davis (2005), who raise the ambiguity between ε-separation and non-zero separation, and favor the latter in their encoding to SMT. Note that, despite the confusion on this matter, PDDL 2.1 semantics does not prescribe to accept an ε separation value as input to the planner. The ε input is only suggested by Fox and Long (2003) as a way to alleviate the burden of a potentially complex plan validation task. Rintanen (2007) focuses on temporal planning over discrete time, showing the problem to be EXPSPACE-complete. Then, he proceeds showing that with constant action durations ($L_a = U_a$), forbidding action self-overlap makes the problem PSPACE-complete, *i.e.*, reducible to classical planning. The result improves the quite restrictive conditions of *temporally simple languages* previously found by Cushing et al. (2007). Transferring these results to the dense-time case is *a priori* possible only assuming ε-separation, since then problems can be suitably scaled and discretized at will. As we show, the discretization is not always possible under non-zero separation, as the problem becomes *undecidable* when self-overlap is allowed. Moreover, while the restriction to fixed action durations is just syntactic convenience in the discrete case, as an action with an interval $[L_a, U_a]$ of possible durations can be replaced by a finite number of copies, with dense time this is in general not possible. Our results extend those above, in the dense-time setting, by naturally handling actions of non-constant durations and by covering both ε-separation and non-zero separation semantics, with or without self-overlap.

The computational complexity of other planning formalisms has been studied before, from classical (Bylander 1994) to hierarchical (Erol, Hendler, and Nau 1996), to timeline-based planning (Gigante et al. 2016; 2017). When comparing our results with the timeline-based planning paradigm in particular, it is interesting to note a similar complexity jump, with a problem that is EXPSPACE-complete over discrete time (Gigante et al. 2017), but becomes undecidable over dense time (Bozzelli et al. 2018).

The idea of exploiting timed automata to handle temporal planning problems has been recently investigated by other authors as well (Bogomolov et al. 2015; Heinz et al. 2019), but, as far as we know, temporal planning had never been previously captured with *polynomial-size* timed automata.

6 Conclusions

The paper studies the computational complexity of temporal planning, as specified by PDDL 2.1. Our theoretical analysis provides a comprehensive picture of the computational complexity of temporal planning under different semantic interpretations. In particular, mutually exclusive events can be constrained to be separated by a given minimum quan-

tum of time (ε-separation), or just any positive amount of time (non-zero separation). We furthermore consider both the cases where actions are allowed or disallowed to overlap with running instances of themselves.

Our analysis reveals that dense-time temporal planning is PSPACE-complete—no harder than classical planning—when self-overlap is forbidden, while when allowed the problem becomes EXPSPACE-complete with ε-separation and even *undecidable* with non-zero separation. These results clarify the computational consequences of different choices in the PDDL 2.1 semantics, which were vague until now.

Furthermore, our proofs employ the first polynomial-sized encoding of temporal planning into *timed automata*. Analogously or in combination with other works that have tried to exploit timed automata or their extensions (Wang and Williams 2015; Bogomolov et al. 2015), this may lead to practically interesting novel approaches.

Acknowledgements

Nicola Gigante and Angelo Montanari have been supported by the PRID project *ENCASE - Efforts in the uNderstanding of Complex interActing SystEms*, and by the iNdAM GNCS project *Formal Methods for Combined Verification*.

Andrea Micheli and Enrico Scala acknowledge the support by the Autonomous Province of Trento in the scope of L.P. n.6/1999 with grant MAIS (*Mechanical Automation Integration System*) n. 2017-D323-00056 del. n. 941 of 16/06/2017 and by EIT Digital within the *AWARD* project.

References

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

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *Proc. of the 25th International Conference on Automated Planning and Scheduling*, 42–46.

Bouyer, P.; Dufourd, C.; Fleury, E.; and Petit, A. 2004. Updatable timed automata. *Theoretical Computer Science* 321(2):291 – 345.

Bozzelli, L.; Molinari, A.; Montanari, A.; and Peron, A. 2018. Decidability and complexity of timeline-based planning over dense temporal domains. In *Proc. of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, 627–628. AAAI Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artif. Intell.* 69(1-2):165–204.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proc. of the 20th International Conference on Automated Planning and Scheduling*, 42–49. AAAI.

Coles, A. J.; Coles, A.; Olaya, A. G.; Celorrio, S. J.; Linares López, C.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magaz.* 33(1).

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In Veloso, M. M., ed., *IJCAI 2007, Proc. of the 20th International Joint Conference on Artificial Intelligence*, 1852–1859.

Cushing, W. A. 2012. *When is Temporal Planning Really Temporal?* Ph.D. Dissertation, Arizona State University.

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20:155–194.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics in Artificial Intelligence* 18(1):69–93.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2012. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Towards Service Robots for Everyday Environments*, volume 76 of *Springer Tracts in Advanced Robotics*. Springer. 49–64.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *J. Artif. Intell. Res.* 20:239–290.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning - Theory and Practice*. Elsevier.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Gigante, N.; Montanari, A.; Cialdea Mayer, M.; and Orlandini, A. 2016. Timelines are expressive enough to capture action-based temporal planning. In *Proc. of the 23rd International Symposium on Temporal Representation and Reasoning*, 100–109. IEEE Computer Society.

Gigante, N.; Montanari, A.; Cialdea Mayer, M.; and Orlandini, A. 2017. Complexity of timeline-based planning. In *Proc. of the 27th International Conference on Automated Planning and Scheduling*, 116–124. AAAI Press.

Heinz, A.; Wehrle, M.; Bogomolov, S.; Magazzeni, D.; Greitschus, M.; and Podelski, A. 2019. Temporal planning as refinement-based model checking. In *Proc. of the 29th International Conference on Automated Planning and Scheduling*, 195–199. AAAI Press.

Rankooh, M. F., and Ghassem-Sani, G. 2015. ITSAT: an efficient sat-based temporal planner. *Journal of Artificial Intelligence Research* 53:541–632.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *Proc. of the 17th International Conference on Automated Planning and Scheduling*, 280–287.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artif. Intell.* 166(1-2):194–253.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *Ai Magazine* 36(3):90–98.

van Emde Boas, P. 1997. The convenience of tiling. In Sorbi, A., ed., *Complexity, Logic and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics*. Marcel Dekker Inc. 331–363.

Wang, D., and Williams, B. C. 2015. tburton: A divide and conquer temporal planner. In *AAAI*, 3409–3417. AAAI Press.