Dealing with Numeric and Metric Time Constraints in PDDL3 via Compilation to Numeric Planning

Luigi Bonassi, Alfonso Emilio Gerevini, Enrico Scala

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
{luigi.bonassi, alfonso.gerevini, enrico.scala}@unibs.it

Abstract

This paper studies an approach to planning with PDDL3 constraints involving mixed propositional and numeric conditions, as well as metric time constraints. We show how the whole PDDL3 with instantaneous actions can be compiled away into a numeric planning problem without PDDL3 constraints, enabling the use of any state-of-the-art numeric planner that is agnostic to the existence of PDDL3. Our solution exploits the concept of regression. In addition to a basic compilation, we present an optimized variant based on the observation that it is possible to make the compilation sensitive to the structure of the problem to solve; this can be done by reasoning on the interactions between the problem actions and the constraints. The resulting optimization substantially reduces the size of the planning task. We experimentally observe that our approach significantly outperforms existing state-of-the-art planners supporting the same class of constraints over known benchmark domains, settling a new state-of-the-art planning system for PDDL3.

Introduction

PDDL3 (Gerevini et al. 2009) is a standard and popular planning formalism for expressing planning problems with trajectory constraints, temporal properties that every valid plan must satisfy. As other languages to express trajectory constraints (e.g., LTL by Pnueli (1977)), it has been proven to be useful for controlling the structure of the desired solution plans (Bacchus and Kabanza 1998; Baier and McIlraith 2006a,b; Gerevini et al. 2009; De Giacomo, Masellis, and Montali 2014). In this paper, we study planning problems extended with PDDL3 trajectory constraints involving mixed propositional and numeric conditions (PDDL3 constraints, for short), as well as time metric constraints. These temporal specifications are useful for modeling desirable and relevant properties that are not easily captured using Boolean state variables only (Felli, Montali, and Winkler 2022; Geatti, Gianola, and Gigante 2022). For instance, in a logistic domain, we could enforce an airplane to always preserve a certain amount of fuel, or impose that it should carry a specific number of passengers. We can also use trajectory constraints to control the movement of agents in spatial domains, using numeric coordinates to define prohibited areas and intermediate locations to explore (Kiam et al. 2020).

The problem of planning with trajectory constraints has been addressed from different perspectives. For the class of PDDL3 constraints, MIPS-XXL (Edelkamp 2006; Edelkamp, Jabbar, and Nazih 2006) exploits automata transformation to encode the trajectory constraints into the planning problem, and then invokes a custom version of Metric-FF (Hoffmann 2003) to solve the resulting problem. Optic (Benton, Coles, and Coles 2012) exploits a similar automaton representation directly in the search engine and in the relaxed planning graph heuristic of the planner. Lastly, SGPlan (Hsu et al. 2007) is based on a decomposition-based approach and handles trajectory constraints using the Metric-FF planning system as a sub-solver. In the context of PDDL3, other researchers have instead focused on soft trajectory constraints (Baier, Bacchus, and McIlraith 2009; Coles and Coles 2011; Wright, Mattmüller, and Nebel 2018; Percassi and Gerevini 2019) or on trajectory constraints over actions (Bonassi, Gerevini, and Scala 2022). Other approaches deal with the more general class of trajectory constraints represented as LTL propositional constraints over finite traces, and do so either through compilation (Baier and McIlraith 2006a,b; Torres and Baier 2015; Bonassi et al. 2023a,b) or techniques exploiting the trajectory constraints to control the search during planning (Bacchus and Kabanza 1998). However, overall, little attention has been devoted to planning problems involving numeric state variables and trajectory constraints. To fill this gap, this paper proposes a novel encoding to translate a numeric problem with PDDL3 constraints into an equivalent numeric problem without them. Our compilation takes inspiration from TCORE (Bonassi et al. 2021), which however only focuses on propositional trajectory constraints, and substantially extends it with PDDL3 constraints involving numeric conditions and metric time trajectory constraints.

The contributions of this paper are as follows. We propose a compilation schema that handles mixed propositional and numeric formulas in the trajectory constraints. We do so by providing a regression mechanism that reasons over both Boolean and numeric state variables (Rintanen 2008; Scala et al. 2020a; Bonassi et al. 2021). The regression we propose works with trajectory constraints over arbitrary propositional formulas and general arithmetic expressions (e.g., linear and non-linear expressions are supported), that can be

object of metric time conditions. We prove that our compilation is sound and complete. Then, starting from a basic schema, we put forward an algorithm that, by analyzing the regressed terms, provides an optimized variant generating provably smaller compiled problems. This variant is sound and complete, too. Finally, starting from well-known planning domains, we design a new set of benchmarks featuring numeric trajectory constraints and conduct an extensive experimental analysis evaluating both the base and the advanced compilation against existing state-of-the-art planners. The results show that the new compilation significantly improves the ability to solve problems that are challenging or impossible for existing planners.

Background

This section provides the necessary background notions on numeric planning problems with trajectory constraints expressed in PDDL3 (Gerevini et al. 2009). We start by defining numeric planning – the core fragment not including trajectory constraints and corresponding to PDDL 2.1 (Fox and Long 2003), and then present the class of constraints we aim to support.

A numeric planning problem is a tuple $\Pi = \langle F, X, A, I, G\rangle$, where F is a set of Boolean variables, X is a set of numeric variables, A is a set of actions, I is an initial state and G is a goal. A state assigns a truth value to each variable from F and a rational number to each variable from X. Numeric and Boolean variables can appear as terms in propositional formulas. More formally, a formula ϕ is recursively defined as follows:

$$\phi := \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \xi \trianglerighteq \xi \mid f, \text{with } f \in F$$
$$\xi := \xi + \xi \mid \xi - \xi \mid \xi \times \xi \mid \xi / \xi \mid v \mid c, \text{with } v \in X, c \in \mathbb{Q}$$

where $\trianglerighteq \in \{>, \geq, =, \leq, <\}$. We say that f and $\neg f$ are positive and negative *literals*, respectively. A formula is in Negation Normal Form (NNF) when the negation symbols only appear at the literals (notice that it is always possible to remove negated numeric conditions by rewriting, e.g., $\neg(x \geq 0)$ can be rewritten as $(-x > 0)$), and we say that a numeric condition is in *standard form* when written as $\xi \trianglerighteq 0$ with $\trianglerighteq \in \{>, \geq, =\}$. We assume this representation if needed. A formula ϕ is evaluated with respect to a state s, and we write $s \models \phi$ ($s \not\models \phi$) to indicate that ϕ is true (false) in s. Moreover, we write $[v]^s$ and $[\xi]^s$ to denote the value of a variable v in a state s and the valuation of an arithmetical expression ξ in s, respectively.

Formulas are used to represent the goal G and the model of the actions in A. Formally, an action $a \in A$ is a pair $\langle \mathit{Pre}(a), \mathit{Eff}(a)\rangle$, where $\mathit{Pre}(a)$ is a formula expressing the preconditions of a, and $\mathit{Eff}(a)$ is a set of conditional effects, each of which is a pair $c \triangleright e$; c is a formula, while e is an assignment $v := \eta$ of the variable v to the value of η. If v is a Boolean variable from F, then $\eta \in \{\top, \bot\}$, while for numeric variables $v \in X$, η is an arithmetical expression ξ. For an assignment e, we write $\mathsf{lhs}(e)$ and $\mathsf{rhs}(e)$ to indicate the left-hand and right-hand sides of e. Intuitively, a conditional effect $c \triangleright e$ indicates that whenever the action is executed in a state s, the variable $\mathsf{lhs}(e)$ gets updated with the value of $\mathsf{rhs}(e)$ only if s satisfies c. Moreover, we say that $c \triangleright e$ is a *simple effect* if c is $\top$. Formally, an action is applicable in s if $s \models \mathit{Pre}(a)$, and the application of an action a in s yields a new state s', indicated with $s' = s[a]$, where the value of each variable $v \in F \cup X$ is defined as follows:

$$[v]^{s'} = \begin{cases} \mathsf{rhs}(e) & \text{if } \exists\, c \triangleright e \in \mathit{Eff}(a) \text{ such that} \\ & \mathsf{lhs}(e) = v,\ v \in F, \text{ and } s \models c \\ [\mathsf{rhs}(e)]^s & \text{if } \exists\, c \triangleright e \in \mathit{Eff}(a) \text{ such that} \\ & \mathsf{lhs}(e) = v,\ v \in X, \text{ and } s \models c \\ [v]^s & \text{otherwise} \end{cases}$$

We assume that conflicting effects are only yielded by conditional effects with mutually exclusive conditions in a state.

A plan π for a problem $\Pi = \langle F, X, A, I, G\rangle$ is a sequence of actions $\langle a_0, a_1, ..., a_{n-1}\rangle$ in Π; a plan π is executable iff there exists a sequence of states (state trajectory) $\sigma = \langle s_0, s_1, ..., s_n\rangle$ such that $s_0 = I$, $\forall\ i \in [0, \ldots, n-1]$ we have that $s_i \models \mathit{Pre}(a_i)$ and $s_{i+1} = s_i[a_i]$. We say that π is valid for Π iff π is executable for Π and the last state s_n of the induced state trajectory satisfies the goal, that is, $s_n \models G$. Moreover, we say that a state s_k of Π is reachable iff there exists an executable plan $\pi = \langle a_0, a_1, \ldots a_{k-1}\rangle$ for Π that induces the state trajectory $\sigma = \langle s_0 = I, s_1, \ldots, s_k\rangle$.

PDDL3 introduced trajectory constraints as a class of temporal logic formulae over state trajectories. Specifically, PDDL3 constraints are of the following types: `at-end`(ϕ) (AE_ϕ), requiring that ϕ must be true in the last state; `always`(ϕ) (A_ϕ), requiring that every state traversed by the plan satisfies formula ϕ; `at-most-once`(ϕ) (AO_ϕ), requiring that ϕ is true in at most one continuous subsequence of traversed states; `sometime-before`(ϕ, ψ) ($\mathsf{SB}_{\phi,\psi}$), requiring that if ϕ is true in a state s_i, then also ψ is true in a previously traversed state; `sometime`(ϕ) (ST_ϕ), requiring that there is at least one state traversed by the plan where ϕ is true; `sometime-after`(ϕ, ψ) ($\mathsf{SA}_{\phi,\psi}$), requiring that if ϕ is true in a traversed state, then also ψ is true in that state or in a later traversed state; `within`$\,(t, \phi)$ (W^t_ϕ), requiring that ϕ must become true no later than the t-th state; `hold-after`$\,(t, \phi)$ (HA^t_ϕ), requiring that ϕ must become true after the t-th state, or in the last state if the state trajectory has t or less states; `hold-during`$\,(u_1, u_2, \phi)$ ($\mathsf{HD}^{u_1,u_2}_\phi$), requiring that ϕ must remain true in the interval $s_{u_1} \ldots s_{u_2-1}$, or in the last state if the state trajectory has u_1 or less states ; `always-within`$\,(t, \phi, \psi)$ ($\mathsf{AW}^t_{\phi,\psi}$), requiring that if ϕ is true in a state s_i, then ψ must be true no later than state s_{i+t}. Note that t, u_1, and u_2 are numbers from $\mathbb{N}$. Formal semantics of PDDL3 is summarized in Figure 1.

In our work, we call PDDL3 *planning problem* a tuple $\langle \Pi, C\rangle$ where Π is a numeric planning problem and C is a set of PDDL3 trajectory constraints. A plan π is valid for a PDDL3 problem $\langle \Pi, C\rangle$ if it is valid for Π and the state trajectory induced by π satisfies all constraints in C.

Our method uses an operator R for regressing a formula over the effects of an action. Our definition of R combines the notion of propositional regression (Rintanen 2008) with the notion of numeric regression (Scala et al. 2020a). Since the numeric regression has been defined only for planning problems without conditional effects, we assume that all effects $c \triangleright v := \xi$ with $v \in X$ are *simple effects*.

$\sigma \models$ `at-end`(ϕ) iff $s_n \models \phi$
$\sigma \models$ `always`(ϕ) iff $\forall i : 0 \leq i \leq n \cdot s_i \models \phi$
$\sigma \models$ `sometime`(ϕ) iff $\exists i : 0 \leq i \leq n \cdot s_i \models \phi$
$\sigma \models$ `at-most-once`(ϕ) iff $\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then
$\exists j : j \geq i \cdot \forall k : i \leq k \leq j \cdot s_k \models \phi$ and $\forall k : k > j \cdot s_k \models \neg\phi$
$\sigma \models$ `sometime-after`(ϕ, ψ)
iff $\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then $\exists j : i \leq j \leq n \cdot s_j \models \psi$
$\sigma \models$ `sometime-before`(ϕ, ψ)
iff $\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then $\exists j : 0 \leq j < i \cdot s_j \models \psi$
$\sigma \models$ `within`(t, ϕ) iff $\exists i : 0 \leq i \leq t \cdot s_i \models \phi$
$\sigma \models$ `hold-after`(t, ϕ) iff
if $n > t$ then $\exists i : t < i \leq n \cdot s_i \models \phi$
if $n \leq t$ then $s_n \models \phi$
$\sigma \models$ `hold-during`(u_1, u_2, ϕ) iff
if $n > u_1$ then $\forall i : u_1 \leq i < u_2 \cdot s_i \models \phi$
if $n \leq u_1$ then $s_n \models \phi$
$\sigma \models$ `always-within`(t, ϕ, ψ) iff
$\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then $\exists j : i \leq j \leq i + t \cdot s_j \models \psi$.

Figure 1: PDDL3 semantics with σ being a state trajectory.

Definition 1 (Numeric effect regression)**.** *The regression of a numeric condition in standard form $\nu = \xi \trianglerighteq 0$ through the effects of action a, denoted with $\nu^{r(a)}$, is $\xi[v_1/\tau(a, v_1), \ldots, v_k/\tau(a, v_k)] \trianglerighteq 0$ where $v_1 \ldots v_k$ are the numeric variables assigned by a, $\xi[v_i/\tau(a, v_i)]$ is the rewriting of ξ where v_i is substituted with $\tau(a, v_i)$, and*

$$\tau(a, v_i) = \begin{cases} \mathsf{rhs}(e) & \text{if } \exists \top \triangleright e \in \mathit{Eff}(a) \cdot \mathsf{lhs}(e) = v_i \\ v_i & \text{otherwise} \end{cases}$$

Definition 2 (Regression operator)**.** *The regression $R(\phi, a)$ of a NNF formula ϕ through the effects $\mathit{Eff}(a)$ of action a is the formula obtained from ϕ by replacing every numeric condition ν with $\nu^{r(a)}$ and by replacing every positive literal f in ϕ with $\Gamma_{f:=\top}(a) \vee (f \wedge \neg\Gamma_{f:=\bot}(a))$, where $\Gamma_e(a)$ for an assignment e involving a Boolean variable is defined as $\Gamma_e(a) = \bigvee_{c \triangleright e \in \mathit{Eff}(a)} c$.*

Note that Definition 2 implies that $s[a] \models \phi$ iff $s \models R(\phi, a)$, for every formula ϕ, state s and action a.

Solving PDDL3 Problems via Compilation

This section presents NTCORE (Numeric TCORE), the approach that compiles a PDDL3 planning problem into an equivalent problem without trajectory constraints. NTCORE starts from the TCORE compilation schema (Bonassi et al. 2021), and extends it to support PDDL3 trajectory constraints using the regression operator of Definition 2. NTCORE gets a PDDL3 problem $\langle\langle F, X, A, I, G\rangle, C\rangle$ and yields a new numeric problem $\langle F', X', A', I', G'\rangle$.

Preprocessing. To handle the compilation of metric time constraints, we exploit the observation that we can use a single numeric variable to record the current time. This way, we compile some of the constraints, namely `within`, `hold-after`, and `hold-during` using the constraints that do not involve explicit time, and `always-within` with a custom treatment (see below). For the compiled constraint we exploit the following equivalences.

Theorem 1. *Let $\sigma = \langle s_0, \ldots, s_n\rangle$ be a sequence of states, and let τ be a numeric variable such that $\forall i : 0 \leq i \leq n \cdot [\tau]^{s_i} = i$. Then the following equivalences hold:*

`within`$(t, \phi) \equiv$	$\{$`sometime`$(\phi \wedge (\tau \leq t))\}$
`hold-after`$(t, \phi) \equiv$	$\{$`sometime-after`$(\tau = t + 1), \phi)$, `at-end`$((\tau \leq t) \Rightarrow \phi)\}$
`hold-during`$(u_1, u_2, \phi) \equiv$	$\{$`always`$((u_1 \leq \tau) \wedge (u_2 > \tau)) \Rightarrow \phi)$, `at-end`$((\tau \leq u_1) \Rightarrow \phi)\}$

Proof sketch. In each case, we can show the equivalences between the constraints using the rules in Figure 1. Full proof in Bonassi, Gerevini, and Scala (2023). □

To keep τ always synchronized, NTCORE extends all actions in A with the effect $\top \triangleright \tau := \tau + 1$, and sets τ to zero in the initial state ($[\tau]^I = 0$). Then, NTCORE applies the above reformulations before the actual compilation.

Boolean variables. Similarly to other compilations that keep track of the truth of a subformula (Bonassi et al. 2021, 2023a,b), NTCORE introduces a linear number of new Boolean variables. For each AO_ϕ and $\mathsf{SB}_{\phi,\psi}$, we add the fresh $seen_\phi$ and $seen_\psi$ variables to record whether ϕ and ψ have ever held. Moreover, for each constraint c of type ST_ϕ and $\mathsf{SA}_{\phi,\psi}$, we use a fresh variable $hold_c$ to encode whether c is satisfied by the current partial state trajectory. Interestingly, also in the case of numeric constraints, we can reason about the truth of a numeric condition without looking into its structure; that is, we only need a Boolean variable to say whether the constraint is satisfied or not. The set of Boolean variables F' of the numeric problem generated by NTCORE is defined as $F' = F \cup F_{hold} \cup F_{seen}$ where:

$$F_{hold} = \{hold_c \mid c \in C \text{ and } c = \mathsf{ST}_\phi \text{ or } c = \mathsf{SA}_{\phi,\psi}\}$$
$$F_{seen} = \{seen_\psi \mid \mathsf{SB}_{\phi,\psi} \in C\} \cup \{seen_\phi \mid \mathsf{AO}_\phi \in C\}$$

Numeric variables. $\mathsf{AW}^t_{\phi,\psi}$ requires to reason about the time passing between a state s_i satisfying ϕ and a future state s_j satisfying ψ. To keep track of such a gap, we introduce a fresh numeric variable m_ϕ that we use to record the time when ϕ becomes true. In particular, we use the value -1 when there are no instances of ϕ to monitor; either ϕ never became true, or the last occurrence of ϕ has been followed by an occurrence of ψ within t states. Otherwise, we will use the current value of τ to record when ϕ becomes true in a state and signal that there is a deadline to monitor. The set of numeric variables X' of the numeric problem generated by NTCORE is defined as $X' = X \cup \{m_\phi \mid \mathsf{AW}^t_{\phi,\psi} \in C\}$.

Initial state. NTCORE modifies the initial state to reflect the initial status of the constraints; e.g., if we have ST_ϕ with ϕ already satisfied in the initial state, then we set the associated variable to $\top$ in the new initial state I'. Analogously, we set $seen_\phi$ to $\top$ in I' if there is a AO_ϕ constraint such that $I \models \phi$. For an $\mathsf{AW}^t_{\phi,\psi}$ constraint, if $I \models \phi \wedge \neg\psi$, then we have to set the respective m_ϕ to zero. Otherwise, we set m_ϕ to -1, as there is no deadline to monitor yet. Formally, for every $c \in C$:

$$\text{if } c \text{ is } \mathsf{ST}_\phi \quad [hold_c]^{I'} := \begin{cases} \top & \text{if } I \models \phi \\ \bot & \text{otherwise} \end{cases}$$
$$\text{if } c \text{ is } \mathsf{SA}_{\phi,\psi} \; [hold_c]^{I'} := \begin{cases} \top & \text{if } I \models \psi \vee \neg\phi \\ \bot & \text{otherwise} \end{cases}$$
$$\text{if } c \text{ is } \mathsf{SB}_{\phi,\psi} \; [seen_\psi]^{I'} := \begin{cases} \top & \text{if } I \models \psi \\ \bot & \text{otherwise} \end{cases}$$

$$\text{if } c \text{ is } \mathsf{AO}_\phi \quad [seen_\phi]^{I'} := \begin{cases} \top & \text{if } I \models \phi \\ \bot & \text{otherwise} \end{cases}$$

$$\text{if } c \text{ is } \mathsf{AW}^t_{\phi,\psi} \; [m_\phi]^{I'} \quad := \begin{cases} 0 & \text{if } I \models \phi \wedge \neg\psi \\ -1 & \text{otherwise} \end{cases}$$

After extending the initial state, NTCORE also checks whether there is some constraint violated in the initial state, which would make the problem unsolvable. In particular, in PDDL3 planning problems, the initial state is a dead end when either $\exists \mathsf{SB}_{\phi,\psi} \in C \cdot I \models \phi$, $\exists \mathsf{A}_\phi \in C \cdot I \models \neg\phi$, or $\exists \mathsf{AW}^0_{\phi,\psi} \in C \cdot I \models \phi \wedge \neg\psi$. In these cases, NTCORE detects unsolvability and terminates.

Actions. NTCORE encodes the dynamics of PDDL3 constraints into the compiled problem by determining a new precondition P_a and a new set of effects E_a for each action $a \in A$. Intuitively, P_a is a formula preventing a from violating any constraint, while effects in E_a are meant to capture the satisfaction of some key formula relevant to the constraints. Formally, the new set of actions A' is defined as follows: $A' = \{\langle \mathit{Pre}(a) \wedge P_a, \mathit{Eff}(a) \cup E_a \rangle \mid a \in A\}$, where

$P_a := \bigwedge_{c \in C} p_{c,a}$ such that

if c is A_ϕ	$p_{c,a} = R(\phi, a)$
if c is AO_ϕ	$p_{c,a} = \neg(R(\phi, a) \wedge seen_\phi \wedge \neg\phi)$
if c is $\mathsf{SB}_{\phi,\psi}$	$p_{c,a} = \neg(R(\phi, a) \wedge \neg seen_\psi)$
if c is $\mathsf{AW}^t_{\phi,\psi}$	$p_{c,a} = (m_\phi = -1 \vee (\tau - m_\phi < t))$
otherwise	$p_{c,a} = \top$

$E_a = \bigcup_{c \in C} e_{c,a}$ such that

if c is A_ϕ	$e_{c,a} = \{\}$
if c is AO_ϕ	$e_{c,a} = \{R(\phi, a) \triangleright seen_\phi := \top\}$
if c is $\mathsf{SB}_{\phi,\psi}$	$e_{c,a} = \{R(\psi, a) \triangleright seen_\psi := \top\}$
if c is ST_ϕ	$e_{c,a} = \{R(\phi, a) \triangleright hold_c := \top\}$
if c is $\mathsf{SA}_{\phi,\psi}$	$e_{c,a} = \{R(\phi \wedge \neg\psi, a) \triangleright hold_c := \bot, R(\psi, a) \triangleright hold_c := \top\}$
if c is $\mathsf{AW}^t_{\phi,\psi}$	$e_{c,a} = \{R(\phi \wedge \neg\psi, a) \wedge (m_\phi = -1) \triangleright m_\phi := (\tau + 1), R(\psi, a) \triangleright m_\phi := -1\}$

In the classical planning setting, as shown by Bonassi et al. (2021), the regression predicts when propositional formulas become true (or false) by simply checking which atoms are added or deleted by the action. To perform the same reasoning in the numeric setting, we must take the internal structure of the numeric conditions into consideration. We show this process through a simple example.

Example 1. *Let $c = \mathsf{A}_\phi$ be a constraint with $\phi = (x - 10 > 0)$ and a an action with $\mathit{Eff}(a) = \{\top \triangleright x := x - 2\}$. Since we need to prevent the violation of c, we have to anticipate the conditions under which a can be executed without making ϕ false. If x is greater than 12, then we can execute a; otherwise, a will make x lower than or equal to 10 and violate c. This quantitative reasoning is elegantly captured by the numeric regression, as $R(\phi, a) = R(x - 10 > 0, a) = (x - 2) - 10 > 0 = x - 12 > 0$. To prevent a from violating c, we set $x - 12 > 0$ as a precondition of the new action a'.*

Similarly, for every $\mathsf{SB}_{\phi,\psi}$, P_a contains a condition that prevents the execution of a when it would satisfy ϕ while $seen_\psi$ is false. For AO_ϕ, P_a prevents the occurrence of a when it makes ϕ true a second time. In the case of $\mathsf{SB}_{\phi,\psi}$ (AO_ϕ, resp.), we need to keep track of the truth of ψ (ϕ, resp.). The compilation does so by updating the corresponding $seen$ variable with the additional effects defined in E_a.

Example 2. *Let $\psi = (x \times y - 10 > 0)$ be a formula of a $\mathsf{SB}_{\phi,\psi}$ constraint, and a an action that only multiplies the value of y by 2. Executing a when $2 \times x \times y$ is greater than 10 will satisfy ψ, and this is captured via the conditional effect $R(\psi, a) \triangleright seen_\psi := \top = (2 \times x \times y - 10 > 0) \triangleright seen_\psi := \top$.*

In the case of a ST_ϕ, E_a sets $hold_c := \top$ when the action satisfies ϕ. Differently, $\mathsf{SA}_{\phi,\psi}$ becomes satisfied when ψ is true, but it can also be temporarily violated when ϕ is true. This dynamic is captured by introducing two conditional effects affecting the $hold_c$ variable.

Lastly, for an $\mathsf{AW}^t_{\phi,\psi}$ constraint, E_a captures when an action a_{t-1} makes ϕ (but not ψ) true in the state s_t by setting $m_\phi = \tau + 1$. Notice that we assign m_ϕ to $\tau + 1$ only when there is no deadline to monitor (i.e., m_ϕ is equal to -1). This is because we need to prevent the deadline from resetting. To do so, we use the clause $(m_\phi = -1)$ in the conditional effect. Whenever an action makes ψ true, the conditional effect $R(\psi, a) \triangleright m_\phi := -1$ resets the value of m_ϕ to -1 to signal that there is no deadline to monitor. To prevent the violation of $\mathsf{AW}^t_{\phi,\psi}$, P_a forces the planner to meet every deadline on ψ with the precondition $(m_\phi = -1 \vee (\tau - m_\phi < t))$. If m_ϕ is -1, then there is no requirement on the action. Conversely, when m_ϕ assumes a different value, we must enforce ψ to become true before the deadline.

Example 3. *Let $\texttt{always-within}(7, x > 0, y > 0)$ be a constraint. If action a_4 gets $x > 0$ true in s_5, then we must have $y > 0$ holding no later than s_{12}, and the conditional effects of a_4 set $m_{x>0} = 5$. Then, the precondition $(m_{x>0} = -1 \vee (\tau - m_{x>0} < 7))$ prevents the planner from reaching s_{12} without achieving $y > 0$; when $\tau = 12$ and $m_{x>0} = 5$, this condition will become false, blocking a_{12}'s execution.*

Goal. The compilation forces every constraint to be satisfied by the state trajectory induced by a solution plan. For constraints of type A_ϕ, AO_ϕ and $\mathsf{SB}_{\phi,\psi}$, there is no additional requirement on the goal; preventing their violation is sufficient to ensure the validity of solutions. For ST_ϕ and $\mathsf{SA}_{\phi,\psi}$, we must require every $hold_c$ to hold in the last state. Likewise, for every $\mathsf{AW}^t_{\phi,\psi}$, the compilation forces the plan to end only if there is no pending deadline on ψ. This is achieved by forcing every variable m_ϕ to be equal to -1 in the last state. Lastly, for every $\mathsf{AE}_\phi \in C$, we conjoin ϕ to the new goal. Formally, $G' = G \wedge \bigwedge_{c=\mathsf{ST}_\phi \in C} hold_c \wedge \bigwedge_{c=\mathsf{SA}_{\phi,\psi} \in C} hold_c \wedge \bigwedge_{\mathsf{AW}^t_{\phi,\psi} \in C} (m_\phi = -1) \wedge \bigwedge_{\mathsf{AE}_\phi \in C} \phi$.

Theorem 2. *Let $\Pi = \langle \langle F, X, A, I, G \rangle, C \rangle$ be a* PDDL3 *planning problem, and $\Pi' = \langle F', X', A', I', G' \rangle$ be the problem obtained by compiling Π using* NTCORE. *Then Π has a valid plan iff so does Π'.*

Proof sketch. Given a valid plan π for Π, we can obtain a plan π' for Π', and vice versa. Then, by considering each type of constraint, we show that if π' (π) is not valid for Π' (Π) then also π (π') cannot be valid for Π (Π'). Full proof in Bonassi, Gerevini, and Scala (2023). □

Towards More Efficient Compilations

In this section, we explore how we can devise more efficient compilations. The basic NTCORE algorithm may introduce preconditions and effects even when the actions are completely unrelated to the given trajectory constraints. To detect this, we investigate the interactions between actions and PDDL3 trajectory constraints through the notion of *achievers*. Intuitively, an action a is an achiever of a formula ϕ if there exists a reachable state s such that ϕ is false in s and the execution of a in s makes ϕ true in $s[a]$.

Definition 3 (Achiever)**.** *Let $\Pi = \langle F, X, A, I, G\rangle$ be a numeric planning problem, $a \in A$, and ϕ a formula. Then a is an achiever of ϕ iff there is a reachable state s of Π such that $s \not\models \phi$, $s \models \mathit{Pre}(a)$ and $s[a] \models \phi$.*

We denote with $Ach(\phi)$ the set of achievers of a formula ϕ. Our notion of achiever is inspired by previous work on heuristics for numeric planning (Scala et al. 2020a), where the concept of "possible achiever" identifies actions that could achieve a numeric condition from a specific state s. Our notion of achiever instead identifies actions that make a formula true from at least one reachable state of a numeric problem. A direct consequence of this definition is that if $a \notin Ach(\phi)$ and ϕ is false in a state s, then it is impossible to satisfy ϕ by executing a in s. We can exploit this observation to greatly minimize additional preconditions and effects.

For an A_ϕ constraint, an action a can make ϕ false only if it is an achiever of $\neg\phi$. Otherwise, it is impossible for a to falsify ϕ, and this implies that A_ϕ will never be violated by a. Thus, no additional precondition is required. Analogously, if a is not an achiever of ϕ, then it is impossible for a to make ϕ true a second time in the case of an AO_ϕ, or to satisfy ϕ while $seen_\psi$ is false in the case of a $\mathsf{SB}_{\phi,\psi}$. The same logic applies to the additional conditional effects. If $a \notin Ach(\phi)$, then there is no need to add an effect to make $seen_\phi$ true (in the case of AO_ϕ) or to make $hold_c$ true (in the case of ST_ϕ). This is analogous for conditional effects affecting the new variables $seen_\psi$, for $\mathsf{SB}_{\phi,\psi}$, and $hold_c$ for $\mathsf{SA}_{\phi,\psi}$.

We can generalize this to metric time constraints. Consider an $\mathsf{AW}^t_{\phi,\psi}$ constraint. If an action a is not an achiever of $\phi \wedge \neg\psi$, then the execution of a will never trigger the fulfillment of ψ within t steps. Likewise, $a \notin Ach(\psi)$ implies a will never fulfill the demand of ψ after an occurrence of ϕ.

Unfortunately, deciding whether a is an achiever of a formula ϕ requires determining the existence of a state s across the reachable state space such that $s \not\models \phi$, $s \models \mathit{Pre}(a)$ and $s[a] \models \phi$. This is clearly an unfeasible task, as it would correspond to solving a numeric planning problem. Therefore, we identify the achievers in a simplified setting, i.e., for a formula ϕ, we compute an approximation of $Ach(\phi)$. To preserve the soundness and completeness of the compilation, it is sufficient to over-approximate the real set of achievers; if for a formula ϕ we consider every action as an achiever, then we are just introducing some irrelevant precondition or effect. This is exactly what the compilation described in the previous section does.

We now propose a technique to over-approximate the real set of achievers. This strategy exploits the observation that it is possible to compute how the value of arithmetical expressions and propositional formulas change after executing an action, and therefore to understand the impact that such a change would have on a condition if the action were executed. When dealing with literals, this approach is straightforward. An action a is not an achiever of a positive literal f if no effect of a can set f to $\top$. Analogously, a is not an achiever of a negative literal $\neg f$ if no effect of a falsifies f.

Lemma 1. *Let a be an action and F a set of Boolean variables. Then for every $f \in F$ the following holds:*

1. *If $\nexists c \triangleright e \in \mathit{Eff}(a) \cdot e = f := \top$ then $a \notin Ach(f)$.*
2. *If $\nexists c \triangleright e \in \mathit{Eff}(a) \cdot e = f := \bot$ then $a \notin Ach(\neg f)$.*

Proof sketch. Direct from the definition of Achiever. □

For arithmetical expressions, this type of reasoning becomes more involved. For example, let a be an action that decreases the variable x of one unit, i.e. $\top \triangleright x := x - 1 \in \mathit{Eff}(a)$, and consider the constraint $\texttt{sometime}(x > 0)$. It is easy to see that, for every state s, the execution of a in s will never make x greater than 0 if it was not so already. We can capture this case, and generalize it, by predicting whether the action has a positive or negative interaction with the formula. This is only possible for actions and numeric expressions having a constant numeric relation.

Definition 4. *Let $\phi = \xi \trianglerighteq 0$ be a numeric condition in standard form, a an action, and $R(\phi, a) = \xi^r \trianglerighteq 0$ the condition obtained by regressing ϕ through the effects of a. Then a has a Constant Numeric Relation with ϕ (*CNR*) iff $\Delta = \xi^r - \xi$ can be simplified to a rational number.*

Note that, for a numeric condition $\xi \trianglerighteq 0$, Δ may be a complex expression involving many variables. Yet, when Δ can be simplified to a rational number, then it represents the rate of change of ξ after the action. In our previous example with $\texttt{sometime}(x > 0)$, we have $\Delta = \xi^r - \xi = (x - 1) - x = -1$, which indicates that $\xi = x$ decreases by 1 after a. The sign of Δ lets us detect when an action is not an achiever of a numeric condition.

Lemma 2. *Let $\phi = \xi \trianglerighteq 0$ be a numeric condition in standard form, a an action that has a CNR with ϕ, $R(\phi, a) = \xi^r \trianglerighteq 0$ and $\Delta = \xi^r - \xi$. Then the following holds:*

1. *If $\Delta = 0$ then $a \notin Ach(\phi)$.*
2. *If $\Delta < 0$ and $\trianglerighteq \in \{>, \geq\}$, then $a \notin Ach(\phi)$.*

Proof sketch. It is easy to see that a cannot satisfy ϕ when ϕ is false. Indeed, in case 1, $[\xi]^s$ does not change after a, while in case 2 $[\xi]^s$ only decreases after a ($\Delta < 0$). Full proof in Bonassi, Gerevini, and Scala (2023). □

Notice that Lemma 2 does not consider the case of $\Delta \neq 0$ when $\trianglerighteq$ is $=$. Dealing with equality conditions requires reasoning about the exact value of an expression in a specific state, which is not accessible at compilation time. Naturally, we want to extend this approach to generic formulas, and we can do so for formulas in NNF. For a formula ϕ in NNF, we define $Cond(\phi)$ as the set of all numeric conditions and literals that appear in ϕ, that is, $Cond(\phi) = \{\psi \mid \psi$ is a numeric condition or a literal that appears in $\phi\}$. The next theorem provides sufficient conditions to detect actions that do not belong to $Ach(\phi)$.

Theorem 3. *Let a be an action and ϕ a formula in NNF such that $\forall \psi \in Cond(\phi) \cdot a \notin Ach(\psi)$. Then $a \notin Ach(\phi)$.*

Proof sketch. As formulas are in NNF, if a cannot make true any numeric condition or any literal ψ in ϕ, then the execution of a cannot satisfy ϕ, i.e., for every state s with $s \not\models \phi$ we have $s[a] \not\models \phi$, and by Definition 3 $a \notin Ach(\phi)$. □

We can directly exploit the result of Theorem 3 to obtain a new approximation Ach^+ of the real set of achievers.

Definition 5. *Let A be a set of actions. The approximation $Ach^+(\phi)$ of the set of achievers of a formula ϕ is defined as $Ach^+(\phi) = A \setminus \{a \in A \mid a \notin Ach^+(\psi)$ by rules 1-2 of Lemmas 1-2, $\forall \psi \in Cond(\phi)\}$.*

We define a new compilation, i.e., NTCORE$^+$, which does not use irrelevant preconditions and effects by using Ach^+.

Definition 6 (NTCORE$^+$)**.** *We define* NTCORE$^+$ *as the compilation approach that takes as input a* PDDL3 *planning problem $\Pi = \langle \langle F, X, A, I, G \rangle, C \rangle$ and outputs the numeric planning problem $\Pi'' = \langle F'', X'', A'', I'', G'' \rangle$ defined as in the* NTCORE *compilation schema, except that for every action $a'' \in A''$:*

1. *$p_{c,a}$ is a conjunct of P_a iff either:*
   - *$c = \mathsf{A}_\phi$ and $a \in Ach^+(\neg\phi)$, or*
   - *$c = \mathsf{AO}_\phi$ or $c = \mathsf{SB}_{\phi,\psi}$ and $a \in Ach^+(\phi)$*
2. *A conditional effect $e_{c,a} \in E_a$ iff $R(\varphi, a)$ appears in the condition of $e_{c,a}$ and $a \in Ach^+(\varphi)$.*

Theorem 4. *Let Π be a* PDDL3 *planning problem, and Π'' be the numeric problem obtained by compiling Π using* NTCORE$^+$*. Then Π has a valid plan iff so does Π''.*

Proof sketch. Let Π' be the problem obtained by compiling Π with NTCORE. We can show that Π' admits a solution iff so does Π''. Full proof in Bonassi, Gerevini, and Scala (2023). □

Experiments

We experimentally compare NTCORE with the state-of-the-art systems natively supporting PDDL3 trajectory constraints: OPTIC (Benton, Coles, and Coles 2012), SGPLAN (Hsu et al. 2007), and MIPS-XXL (MIPS) (Edelkamp, Jabbar, and Nazih 2006). To our knowledge, these are the only approaches supporting the considered class of problems. NTCORE is available in Bonassi, Gerevini, and Scala (2023) and has been implemented using the unified planning library (https://github.com/aiplan4eu/unified-planning/) and SymPy (https://www.sympy.org/) to deal with Definition 4. In our experiments, we test the basic NTCORE compilation and NTCORE$^+$. An objective of our empirical analysis is to understand the behavior of these two variants. Theoretically, the differences are clear: NTCORE is dominated by NTCORE$^+$ in terms of the size of compiled problems. However, it is not clear how the optimization affects the overall planning performance. To solve problems resulting from the two compilations we employed ENHSP (https://sites.google.com/view/enhsp/), which is a state-of-the-art planner supporting conditional effects. The configuration of ENHSP we tested performs a Greedy Best-First Search guided by the numeric h_{add} heuristic (Scala et al. 2020a).

We tested all systems on benchmark domains featuring PDDL3 trajectory constraints, and we analyzed the coverage (number of solved instances), the time spent for finding a solution (compilation time plus search time for NTCORE), and the solution quality (in terms of plan length). For the two variants of NTCORE, we also measure the overhead of the compilation by counting the additional terms (variables and numeric constants) in preconditions and effects. Experiments were run on an Intel Xeon Gold 6140M 2.3 GHz, with runtime and memory limits of 1800s and 8GB, respectively.

Benchmark Domains. To our knowledge, there are no existing domains featuring PDDL3 trajectory constraints with numeric terms and time metric constraints. Therefore, we generated a new set of instances starting from ten well-known (Scala et al. 2020a; Kuroiwa et al. 2021) numeric planning domains: COUNTERS, ROVERS, DEPOTS, TPP, ZENOTRAVEL, FARMLAND, PLANTWATERING, SAILING, and BLOCKGROUPING. For each instance of such domains, we produced a new instance featuring PDDL3 constraints.

In COUNTERS, we used `sometime-before` constraints to force a counter to reach a specific value before changing other counters. In ROVERS, we used `sometime` and `at-most-once` to control the battery usage of the rovers. In DEPOTS, we controlled the load of trucks with `always` and `sometime` constraints. In TPP, we used `within` constraints to impose a deadline on the satisfaction of subgoals. For ZENOTRAVEL, we used the original instances to generate: (I) 23 new problems with `always` and `sometime` constraints that limit fuel consumption and balance the number of passengers on board each plane, and (II) 23 new instances where `always-within` is employed to limit the time passengers can spend on board and to force planes to refuel within t steps when the fuel level reaches a threshold. In FARMLAND, we used `sometime` and `always` constraints to control the power distribution between the farms. For PLANTWATERING, we designed a complex `always` constraint to challenge and understand the scalability of all systems. In SAILING, we used the original instances to generate: 40 new problems with `at-most-once` constraints specifying that a boat should never visit a certain area twice, and 40 new instances where `hold-during` is used to request that boats should remain in a certain area for a specific interval of time. Lastly, in BLOCKGROUPING, we used `at-most-once` constraints to request that a block should never be moved inside a certain area twice. Figure 2 summarizes the different types of constraints across all domains. In total, our benchmark suite features 512 instances: 11 for COUNTERS, 20 for ROVERS, 22 for DEPOTS, 40 for TPP, 46 (23 + 23) for ZENOTRAVEL, 50 for FARMLAND, 51 for PLANTWATERING, 80 (40 + 40) for SAILING, and 192 for BLOCKGROUPING.

Results. Table 1 reports coverage achieved by all systems across all domains. We observe that NTCORE$^+$ solves most of the problems, achieving about five times the coverage of OPTIC, the best-performer among native PDDL3 systems. In particular, NTCORE$^+$ performs better in all but three domains: ROVERS, DEPOTS, and TPP. In these domains,

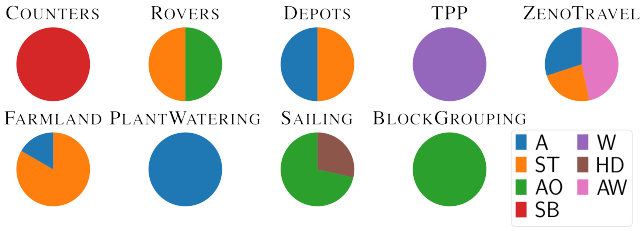


Figure 2: Types of constraints across all domains.

Domain	NTCORE	NTCORE$^+$	OPTIC	SGPLAN	MIPS
COUNTERS (11)	5	**6**	2	0	0
ROVERS (20)	8	9	**10**	0	9
DEPOTS (22)	10	12	9	0	**14**
TPP (40)	10	12	14	0	**25**
ZENOTRAVEL (46)	31	**35**	16	1	20
FARMLAND (50)	**50**	**50**	8	0	0
PLANTWATERING (51)	24	**30**	12	0	0
SAILING (80)	35	**36**	0	10	0
BLOCKGROUPING (192)	116	**161**	N.A.	24	0
TOTAL	289	**351**	71	35	68

Table 1: Coverage analysis. N.A. stands for Not Applicable.

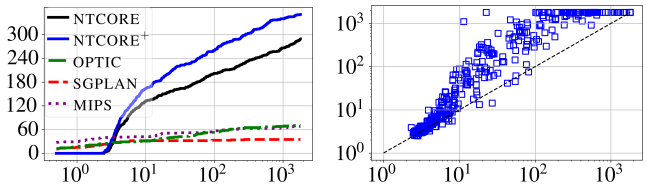
OPTIC and MIPS outperform NTCORE$^+$, especially in TPP. These domains are mostly propositional, and we attribute this to ENHSP being optimized for domains with predominant numeric structures. Differently, in *heavily-numeric* domains (Scala et al. 2020b) such as FARMLAND, SAILING and PLANTWATERING, NTCORE performs much better. To confirm our intuition, we tested METRIC-FF (Hoffmann 2003) with NTCORE$^+$ and, while obtaining worse overall results, we were able to solve 27 instances in TPP, outperforming MIPS. This result shows the key advantage of NTCORE: we can directly exploit the most effective and sophisticated heuristic search approaches to solve PDDL3 problems. For BLOCKGROUPING, we could not use OPTIC as the original instances include disjunctive goals, which are not supported by OPTIC. We also observed some instabilities in some instances of ZENOTRAVEL, TPP, and SAILING, as `always-within`, `hold-during`, and `within` do not seem well supported by OPTIC. Yet, even if we do not count these instances, NTCORE is still the best approach coverage-wise (156 solved by NTCORE$^+$ vs. 55 solved by OPTIC).

Figure 3a shows how each system increases its coverage over time. We can see that MIPS and SGPLAN reach their maximum coverage almost instantly. Yet, after about four seconds, NTCORE dominates all PDDL3 native planners. NTCORE is indeed affected by a bit of overhead at start time, given by the compilation and by the fact of using ENHSP that is written in Java, and the JVM needs some starting time.

If we compare the two versions of NTCORE, we can see that, as expected, NTCORE$^+$ dominates NTCORE coverage-wise. By adopting the proposed optimization we increase the coverage by 21.4%. We also measure a substantial improvement in terms of runtime, as shown in an instance-by-instance comparison between NTCORE and NTCORE$^+$ in Figure 3b. In many cases, NTCORE$^+$ is one order of magnitude faster than NTCORE, and in total NTCORE$^+$ achieves lower runtimes in 319 instances.

Domain	Preconditions		Effects	
	NTCORE	NTCORE$^+$	NTCORE	NTCORE$^+$
COUNTERS	3279.5	**76.7**	3279.5	**153.5**
ROVERS	7692.6	**87.0**	47260.9	**116.0**
DEPOTS	113135.2	**3137.6**	150324.0	**3922.0**
TPP	**0.0**	**0.0**	332017.5	**7702.5**
ZENOTRAVEL	164435.6	**122596.2**	345816.2	**41808.1**
FARMLAND	92.4	**46.2**	611.2	**94.4**
PLANTWATERING	1766052.7	**44359.1**	**0.0**	**0.0**
SAILING	**3037.4**	**3037.4**	4565.0	**955.6**
BLOCKGROUPING	**5321.2**	**5321.2**	138030.0	**3386.2**

Table 2: Average number of additional terms in preconditions and effects. Averages are computed only considering instances compiled by NTCORE and NTCORE$^+$.



(a) (b)

Figure 3: Coverage (y-axis) versus planning time (x-axis) (a) and pairwise runtime comparison of NTCORE (y-axis) with NTCORE$^+$ (x-axis) (b).

To better understand these results, Table 2 reports the average number of *additional* terms (variables and numeric constants) in preconditions and effects. We observe that NTCORE$^+$ adds up to two orders of magnitude fewer terms than NTCORE in both preconditions and effects. These lighter models speed up ENHSP substantially.

Regarding solution quality, we measured that NTCORE$^+$ and the PDDL3 systems return plans of similar lengths, except in two domains. In PLANTWATERING, NTCORE$^+$ performs much better than OPTIC; on average, plans returned by NTCORE$^+$ have 80 fewer actions than those returned by OPTIC. Instead, in the TPP domain, OPTIC and MIPS yield solutions that on average contain 63.8% and 72.9% fewer actions than the solutions of NTCORE$^+$, respectively.

Conclusions

We have presented a novel approach to planning with state trajectory constraints involving numeric state variables and metric time constraints. Our approach works by compilation; that is, we take a planning problem with state trajectory constraints and generate one without them with the guarantee of soundness and completeness. This way, any numeric planner can be used to solve the task. Our results show that the compilation is not only feasible and effective but also outperforms state-of-the-art planners that natively handle numeric state trajectory constraints. For future work, we intend to investigate the adoption of numeric state variables in more complex temporal constraints, such as those that can be expressed in LTL modulo theory (Geatti, Gianola, and Gigante 2022), or in temporal planning (Fox and Long 2003).

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work has been partially supported by the EU H2020 project AIPlan4EU (No. 101016442), the EU ICT-48 2020 project TAILOR (No. 952215), and the PRIN project RIPER (No. 20203FFYLK).

References

Bacchus, F.; and Kabanza, F. 1998. Planning for Temporally Extended Goals. *Ann. Math. Artif. Intell.*, 22(1-2): 5–27.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artif. Intell.*, 173(5-6): 593–618.

Baier, J. A.; and McIlraith, S. A. 2006a. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*, 788–795. AAAI Press.

Baier, J. A.; and McIlraith, S. A. 2006b. Planning with Temporally Extended Goals Using Heuristic Search. In *ICAPS*, 342–345. AAAI.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *ICAPS*. AAAI.

Bonassi, L.; De Giacomo, G.; Favorito, M.; Fuggitti, F.; Gerevini, A. E.; and Scala, E. 2023a. FOND Planning for Pure-Past Linear Temporal Logic Goals. In *ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 279–286. IOS Press.

Bonassi, L.; De Giacomo, G.; Favorito, M.; Fuggitti, F.; Gerevini, A. E.; and Scala, E. 2023b. Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic. In *ICAPS*, 61–69. AAAI Press.

Bonassi, L.; Gerevini, A. E.; Percassi, F.; and Scala, E. 2021. On Planning with Qualitative State-Trajectory Constraints in PDDL3 by Compiling them Away. In *ICAPS*, 46–50. AAAI Press.

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2022. Planning with Qualitative Action-Trajectory Constraints in PDDL. In *IJCAI*, 4606–4613. ijcai.org.

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2023. Dealing with Numeric and Metric Time Constraints in PDDL3 via Compilation to Numeric Planning: Supplementary Material. https://github.com/LBonassi95/NumericTCORE.

Coles, A. J.; and Coles, A. 2011. LPRPG-P: Relaxed Plan Heuristics for Planning with Preferences. In *ICAPS*. AAAI.

De Giacomo, G.; Masellis, R. D.; and Montali, M. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *AAAI*, 1027–1033. AAAI Press.

Edelkamp, S. 2006. On the Compilation of Plan Constraints and Preferences. In *ICAPS*, 374–377. AAAI.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-scale optimal PDDL3 planning with MIPS-XXL. *5th International Planning Competition Booklet (IPC-2006)*, 28–30.

Felli, P.; Montali, M.; and Winkler, S. 2022. Linear-Time Verification of Data-Aware Dynamic Systems with Arithmetic. In *AAAI*, 5642–5650. AAAI Press.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Geatti, L.; Gianola, A.; and Gigante, N. 2022. Linear Temporal Logic Modulo Theories over Finite Traces. In *IJCAI*, 2641–2647. ijcai.org.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6): 619–668.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Artif. Intell. Res.*, 20: 291–341.

Hsu, C.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences. In *IJCAI*, 1924–1929.

Kiam, J. J.; Scala, E.; Jávega, M. R.; and Schulte, A. 2020. An AI-Based Planning Framework for HAPS in a Time-Varying Environment. In *ICAPS*, 412–420. AAAI Press.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2021. LM-cut and Operator Counting Heuristics for Optimal Numeric Planning with Simple Conditions. In *ICAPS*, 210–218. AAAI Press.

Percassi, F.; and Gerevini, A. E. 2019. On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata. In *ICAPS*, 320–328. AAAI Press.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*, 46–57. IEEE Computer Society.

Rintanen, J. 2008. Regression for Classical and Nondeterministic Planning. In *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, 568–572. IOS Press.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020a. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.*, 68: 691–752.

Scala, E.; Saetti, A.; Serina, I.; and Gerevini, A. E. 2020b. Search-Guidance Mechanisms for Numeric Planning Through Subgoaling Relaxation. In *ICAPS*, 226–234. AAAI Press.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *IJCAI*, 1696–1703. AAAI Press.

Wright, B.; Mattmüller, R.; and Nebel, B. 2018. Compiling Away Soft Trajectory Constraints in Planning. In *KR*, 474–483. AAAI Press.