Pushing the Envelope in Numeric Pattern Planning

Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
KR in Planning and Scheduling

Pushing the Envelope in Numeric Pattern Planning

Matteo Cardellini , Enrico Giunchiglia
DIBRIS, University of Genova, Genova, Italy
{matteo.cardellini, enrico.giunchiglia}@unige.it

Abstract

In this paper, we present a symbolic search-based procedure for numeric planning based on Symbolic Pattern Planning (SPP). In SPP, a pattern is a sequence of actions used to define a logic formula whose models correspond to sequences of applicable actions and reachable states. Here, starting from the empty pattern, we iteratively extend and compress it using search techniques until a goal state is reached. We prove the correctness and completeness of the procedure and demonstrate its good performance compared to both the original SPP approach and other publicly available numeric planners on the 2023 International Planning Competition Agile track.

1 Introduction

In deterministic AI planning, the objective is to find a sequence of actions leading from an initial state I to a state satisfying a goal condition G. Two main alternative approaches exist to solve a planning problem. *Planning as Search* (Bonet and Geffner 2001), instead of exploring *all* possible sequences of actions, selects a sequence and extends it with the action likely to lead to a goal state. *Planning as Satisfiability* (Kautz and Selman 1992) instead

1. fixes a bound n initially set to 0,
2. computes a logic encoding of a set S of *all* the sequences of actions executable from a state and of length at most n,
3. checks whether S contains a valid plan by imposing I as the starting state of each sequence, and G as a condition on the state resulting from executing each sequence, and
4. increases n upon failure of the check at the previous step.

Symbolic Pattern Planning (SPP) (Cardellini, Giunchiglia, and Maratea 2024a) is a recent logic encoding, which, given an arbitrary finite sequence of actions $\prec$ called *pattern*, models the sequences of actions which are executable from a state and are also a subsequence of $\prec$. Planning as Satisfiability with SPP is very effective in solving numeric (Cardellini, Giunchiglia, and Maratea 2024b) and temporal (Cardellini and Giunchiglia 2025) planning problems.

In this paper, we push the envelope of planning with patterns, and show how to symbolically search for a valid plan by iteratively extending (adding actions to) and compressing (removing actions from) an initially computed pattern. Specifically, the idea is to concatenate

1. an initial pattern that allows reaching a state s satisfying a subset of the goals, and
2. another pattern computed from s, which is extended until a state satisfying new additional goals can be reached.

At the beginning, the first pattern is computed from the initial state, and the two steps are iterated until all the goals have been satisfied, at which point a valid plan is returned.

On the theoretical side, we prove the correctness (any returned plan is valid) and completeness (if there exists a valid plan, one will be returned) of the proposed procedure. On the experimental side, the analysis shows that the proposed procedure has good performance compared to both the previous SPP approach and all the other publicly available (both symbolic and search-based) planners with the settings, domains and problems of the 2023 International Planning Competition (IPC), Agile track. Overall, our procedure solves 325 problems out of the 420 considered, compared to the 284 solved by the previous state of the art. Ablation studies reveal that compressing the initial pattern – i.e., removing actions that are not necessary to reach the state where the pattern is recomputed – has the most significant impact on the performance of our procedure.

These are the main contributions of the paper:

1. We present a SPP search-based symbolic procedure for numeric planning, the first exploiting Planning as Search techniques in a Planning as Satisfiability setting.
2. We prove its correctness and completeness.
3. We show that it outperforms the publicly available planners on the benchmarks of the 2023 IPC Agile track.
4. We conduct ablation studies to highlight which technique is most effective.

This work builds significantly on the work of (Cardellini, Giunchiglia, and Maratea 2024a), which itself builds on top of (i) the $R^2\exists$-encoding for classical planning of (Balyo 2013) adapted for numeric planning in (Bofill, Espasa, and Villaret 2016) and (ii) the rolling encoding presented in (Scala et al. 2016b). In search-based planning, several planners exploit actions sequences, first searching for a goal state employing the full sequence, and then resorting back to single actions if unsuccessful. The classical planner YAHSP (Vidal 2004), employs "look-ahead plans" (i.e., sequences) in the forward search trying to jump to intermediate states

closer to the goal. This is similar to the concept of macro-actions (Alarnaouti, Percassi, and Vallati 2024). Patterns, however, allow capturing a larger superset of sequences than macro-actions and "look-ahead plans", since actions in any position of the pattern may not be selected in the plan, while the latter approaches only employ the full sequence. Moreover, in numeric planning, the planner needs also to consider how many times a single action in the sequence has to be consecutively applied (i.e. rolled) which is standard in the SPP approach. Conclusively, all the abovementioned approaches are implemented only for search-based approaches, while our work moves forward the state-of-the-art for satisfiability-based planning.

After the background on numeric Planning as Satisfiability with SPP (Section 2), a simple example illustrates the drawbacks of the original SPP approach (Section 3), followed by the novel procedure presented in this paper (Section 4), its behaviour on the motivating example (Section 5), and the experimental analysis and the ablation study (Section 6). We end the paper with the conclusions and prospective for future work.

2 Numeric Planning with SPP

In this section, we first introduce the syntax and semantics of PDDL2.1 level 2 (Fox and Long 2003), the de-facto standard in numeric planning. Then, we show how PDDL2.1 problems have been encoded in Satisfiability Modulo Theories (SMT) (Barrett, Fontaine, and Tinelli 2016) exploiting the Planning as Satisfiability approach and the SPP encoding.

2.1 Numeric Planning

In PDDL2.1, a *numeric planning problem* Π is a tuple $\Pi = \langle V_B, V_N, A, I, G\rangle$ where V_B and V_N are sets of propositional and numeric variables with domain in $\{\top, \bot\}$ and $\mathbb{Q}$, respectively, where $\top$ and $\bot$ are the symbols for truth and falsity. A *propositional condition* is an expression of the form $v = \top$ or $v = \bot$, with $v \in V_B$. A *numeric condition* is an expression of the form $\psi \trianglerighteq 0$, with $\trianglerighteq \in \{<, \leq, =, \geq, >\}$ and ψ a linear combination of the variables in V_N, i.e., $\sum_{x \in V_N} k_x x + k$, with $k_x, k \in \mathbb{Q}$. A *state* is a function assigning each variable in $V_B \cup V_N$ to an element in its domain, and is naturally extended to conditions and formulas, the latter defined as propositional combination of conditions. In Π, I is the *initial state* and G is a set of *goal formulas* whose models are the *goal states*. An *action* $a \in A$ is a pair $\langle \mathrm{pre}(a), \mathrm{eff}(a)\rangle$ where $\mathrm{pre}(a)$ is the set of *preconditions* of a, i.e., a set of propositional and numeric conditions, and $\mathrm{eff}(a)$ is the set of *effects* of a, i.e., expressions of the form $v := \top$ or $w := \bot$ or $x := \psi$ with $v, w \in V_B$, $x \in V_N$ and ψ a linear expression. For each action a, the variables occurring in $\mathrm{eff}(a)$ to the left of the ":=" symbol are said to be *assigned by* a. A numeric effect $x := \psi$ is said to be a *linear increment* if $\psi = x + \psi'$ with ψ' a linear expression not containing x. As standard, we write $(\psi \trianglerighteq \psi')$ for $(\psi - \psi' \trianglerighteq 0)$ and we abbreviate $x := x + \psi$ and $x := x - \psi$ with $x \mathrel{+}= \psi$ and $x \mathrel{-}= \psi$ respectively, $v = \top$ and $v = \bot$ with v and $\neg v$ respectively. From here on, v, w, x represent variables and ψ a numeric expression, each symbol possibly decorated with subscripts or superscripts.

An action a is *executable* in a state s if for each $v = \top$, $w = \bot$ and $\psi \trianglerighteq 0$ in $\mathrm{pre}(a)$ we have $s(v) = \top$, $s(w) = \bot$ and $s(\psi) \trianglerighteq 0$. The *result of executing* an action a in a state s is the state s' such that, for each $v \in V_B \cup V_N$, $s'(v) = \top$ if $v := \top \in \mathrm{eff}(a)$, $s'(v) = \bot$ if $v := \bot \in \mathrm{eff}(a)$, $s'(v) = s(\psi)$ if $v := \psi \in \mathrm{eff}(a)$, and $s'(v) = s(v)$ otherwise.

Consider a *plan* π, defined as a finite sequence of actions $a_0; \ldots; a_{n-1}$ of *length* $n \geq 0$. The *state sequence* $s_0; \ldots; s_n$ *induced by* π *in* s_0 is such that for $i \in \{0, \ldots, n-1\}$, s_{i+1} is (i) undefined if either a_i is not executable in s_i or s_i is undefined, and (ii) the result of executing a_i in s_i otherwise.

We say that π is *executable in a state* s_0 if each state in the sequence induced by π in s_0 is defined. If π is executable in the initial state I and the last state induced by π in I is a goal state, we say that π is a *valid plan*.

2.2 Planning as Satisfiability with SPP

Let $\Pi = \langle V_B, V_N, A, I, G\rangle$ be a numeric planning problem. In Planning as Satisfiability (Kautz and Selman 1992; Kautz and Selman 1996), an *encoding* E *of* Π is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{I}(\mathcal{X}), \mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}'), \mathcal{G}(\mathcal{X})\rangle$ where $\mathcal{X}$ is a finite set of propositional and numeric *state variables* including $V_B \cup V_N$; $\mathcal{A}$ is a finite set of *action variables*, each one with the set of values it can take; $\mathcal{I}(\mathcal{X})$ is the *initial state formula* in the set $\mathcal{X}$ of variables, defined as

$$\bigwedge_{v:I(v)=\top} v \wedge \bigwedge_{w:I(w)=\bot} \neg w \wedge \bigwedge_{x,k:I(x)=k} x = k;$$

while $\mathcal{G}(\mathcal{X})$ is the *goal formula* in the set $\mathcal{X}$ of variables, obtained by making the conjunction of the formulas in G.

The valid transitions between states correspond to the models of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$, the *symbolic transition relation*, a formula in the variables $\mathcal{X} \cup \mathcal{A} \cup \mathcal{X}'$, where $\mathcal{X}'$ is the set of *next state variables* consisting of a new variable v' for each variable $v \in \mathcal{X}$. In SPP, the definition of the symbolic transition relation starts by fixing a *pattern*, defined as a finite sequence $a_1; \ldots; a_k$ of actions in A, with $k \geq 0$. The empty pattern, obtained for $k = 0$, is denoted with ϵ.

Consider a pattern $\prec = a_1; \ldots; a_k$, $k \geq 0$. The basic idea of SPP is to define the value of each state variable in the state resulting from the execution of each action in $\prec$ for 0 or more times, as a function of both the state in which the execution is started and of the pattern $\prec$. Notice that, by definition, the pattern can contain multiple, even consecutive, occurrences of the same action a. However, each repeated occurrence is treated as a different copy of the action. With such assumption, with a_i we denote both the i-th action in the pattern and the corresponding action variable in the encoding. Then, in the *pattern* $\prec$*-encoding of* Π,

1. the set of state variables $\mathcal{X} = V_B \cup V_N$ is the union of the propositional and numeric variables of Π,
2. The set of action variables is defined as $\{a_1, \ldots, a_k\}$, with one variable per action occurrence in the pattern $\prec$. Each variable a_i ranges over the non-negative integers $\mathbb{N}^{\geq 0}$. Intuitively, the value of a_i represents the number of times that the action is executed consecutively, following the sequential execution of each action in $\{a_1, \ldots, a_{i-1}\}$,

each also executed zero or more times. Thus, in SPP, depending on the fixed pattern $\prec$, we have a different set of action variables, denoted with $\mathcal{A}^{\prec}$.

The value taken by $v \in V_B \cup V_N$ after the sequential execution of each action occurrence a_i in $\prec$ for a number ≥ 0 of consecutive times, is given by $\sigma_i(v)$, inductively defined as $\sigma_0(v) = v$, and for each $i \in [1, k]$

1. $\sigma_i(v) = \sigma_{i-1}(v)$ if v is not assigned by a_i,
2. $\sigma_i(v) = (\sigma_{i-1}(v) \vee a_i > 0)$ if $v := \top \in \text{eff}(a_i)$,
3. $\sigma_i(v) = (\sigma_{i-1}(v) \wedge a_i = 0)$ if $v := \bot \in \text{eff}(a_i)$,
4. $\sigma_i(v) = (\sigma_{i-1}(v) + a_i \times \sigma_{i-1}(\psi))$ if $v \mathrel{+}= \psi \in \text{eff}(a_i)$ is a linear increment, where $\sigma_{i-1}(\psi)$ is the expression obtained from ψ after each variable $x \in V_N$ has been replaced with $\sigma_{i-1}(x)$,
5. $\sigma_i(v) = \text{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v))$ if $v := \psi \in \text{eff}(a_i)$ is not a linear increment.

The term $\text{ITE}(c, t, e)$ for "*If (c) Then t Else e*" returns t or e depending on whether the condition c is true or not, and is part of the standard language supported by SMT solvers.

The symbolic transition relation of the $\prec$-encoding – which defines the value of the variables in $\mathcal{X}'$ on the basis of the values of the variables in $\mathcal{X}$ and in $\mathcal{A}^{\prec}$– is denoted with $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$, and is defined as the conjunction of the formulas in the sets:

1. $\text{pre}^{\prec}(A)$, which contains, for each action a_i in $\prec$ and for each $v = \top$ and $w = \bot$ in $\text{pre}(a_i)$,

$$a_i > 0 \rightarrow v, \qquad a_i > 0 \rightarrow \neg w,$$

and for each $\psi \trianglerighteq 0$ in $\text{pre}(a_i)$,

$$a_i > 0 \rightarrow \sigma_{i-1}(\psi) \trianglerighteq 0, \qquad a_i > 1 \rightarrow \sigma_{i-1}(\psi[a_i]) \trianglerighteq 0,$$

where $\psi[a_i]$ is the linear expression obtained from ψ by substituting each variable $x \in V_N$ with

   (a) $x + (a_i - 1) \times \psi'$, whenever $x \mathrel{+}= \psi' \in \text{eff}(a_i)$ is a linear increment,
   (b) ψ', if $x := \psi' \in \text{eff}(a_i)$ is not a linear increment,
   (c) x, if x is not assigned by a_i.

The above formulas ensure that the numeric precondition $\psi \trianglerighteq 0 \in \text{pre}(a_i)$ holds both in the first and in the last state in which a_i is executed, and thus that $\psi \trianglerighteq 0$ holds also in all the intermediate states in which a_i is consecutively executed, see (Scala et al. 2016b).

2. $\text{amo}^{\prec}(A)$, which contains, for each action a_i in $\prec$ which is not eligible for rolling (Scala et al. 2016b)

$$a_i = 0 \vee a_i = 1.$$

An action a_i is *eligible for rolling* if

   (a) $v = \bot \in \text{pre}(a_i)$ (resp. $v = \top \in \text{pre}(a_i)$) implies $v := \top \notin \text{eff}(a_i)$ (resp. $v := \bot \notin \text{eff}(a_i)$), and
   (b) all the numeric variables assigned by a_i with a linear increment do not occur elsewhere in $\text{eff}(a_i)$, and
   (c) a_i contains a linear increment.

3. $\text{frame}^{\prec}(V_B \cup V_N)$, consisting of, for each variable $v \in V_B$ and $x \in V_N$,

$$v' \leftrightarrow \sigma_k(v), \qquad x' = \sigma_k(x).$$

Given $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$, following the Planning as Satisfiability approach, an integer $n \geq 0$ called *bound* or *number of steps* is fixed, $n+1$ disjoint copies $\mathcal{X}_0, \ldots, \mathcal{X}_n$ of the set $\mathcal{X}$ of state variables, and n disjoint copies $\mathcal{A}_0^{\prec}, \ldots, \mathcal{A}_{n-1}^{\prec}$ of the set $\mathcal{A}^{\prec}$ of action variables are made, and the *$\prec$-encoding of Π with bound n* is defined to be the formula

$$\Pi_n^{\prec} = \mathcal{I}(\mathcal{X}_0) \wedge \bigwedge_{i=0}^{n-1} \mathcal{T}^{\prec}(\mathcal{X}_i, \mathcal{A}_i^{\prec}, \mathcal{X}_{i+1}) \wedge \mathcal{G}(\mathcal{X}_n), \qquad (1)$$

in which $\mathcal{I}(\mathcal{X}_0)$ is the formula in the variables $\mathcal{X}_0$ obtained by substituting each variable $x \in \mathcal{X}$ with $x_0 \in \mathcal{X}_0$ in $\mathcal{I}(\mathcal{X})$, and similarly for $\mathcal{T}^{\prec}(\mathcal{X}_i, \mathcal{A}_i^{\prec}, \mathcal{X}_{i+1})$ and $\mathcal{G}(\mathcal{X}_n)$. Then, the satisfiability of $\Pi_n^{\prec}$ is checked calling an SMT solver starting from $n = 0$ and then incrementing n until a model is found.

In (Cardellini, Giunchiglia, and Maratea 2024a) it was shown that for any pattern $\prec$ and bound n, any model of $\Pi_n^{\prec}$ corresponds to a valid plan of Π (*correctness*), and that if Π has a valid plan then for any complete pattern $\prec$ there exists a bound n for which $\Pi_n^{\prec}$ is satisfiable (*completeness*). A pattern is *complete* (resp. *simple*) if it contains at least (resp. at most) one occurrence of each action in A.

For selecting the pattern, Cardellini, Giunchiglia, and Maratea exploited the Asymptotic Relaxed Planning Graph (ARPG) construction from (Scala et al. 2016a). In an ARPG, actions are divided in layers: at layer $l = 0$ there are the actions which are executable in the initial state, and at the higher layers there are the actions whose executability requires the execution of some actions in the lower layers (see (Scala et al. 2016a) for more details). This division in layers defines a partial order on actions, which is then extended to a total order, i.e., to a simple and complete pattern. In the following, to uniquely characterize the pattern produced with the ARPG, actions in the same layer are lexicographically ordered.

3 Motivating Example

In a relay race, there are $N+1$ runners $r_0, r_1, \ldots, r_N$ running on a linear track (an x axis) of length $(N+1) \times L$ with $N > 0$ and $L \geq 1$, passing a baton to each other. The position x_i of runner r_i ranges in $[L \times i, L \times (i+1)]$, with $i \in [0, N]$. Each runner can run forward or backward, increasing or decreasing its position by 1, only if it is holding the baton. To exchange the baton, two runners r_i and r_{i+1} must be in the same position. We assume that $b_i = 1$ if r_i has the baton and $b_i = 0$ otherwise,[1] while btd_i is a Boolean variable denoting if r_i has touched the baton. In all the problems in this domain, we assume that initially the runner r_0 has the baton, that he is the only one who has touched the baton and that each runner r_i is in position $L \times i$.

[1] We use $b_i \in \{0, 1\}$ instead of a Boolean variable to allow for a concise modelling of the baton exchange action.

This scenario can be modelled as a planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$ having

$$\begin{aligned} V_B &= \{btd_i \mid i \in [0,N]\}, \quad V_N = \{x_i, b_i \mid i \in [0,N]\},\\ A &= \{fw_i, bw_i \mid i \in [0,N]\} \cup \{xc_j \mid j \in [1,N]\},\\ I &= \{x_i = L \times i \mid i \in [0,N]\}\\ &\quad \cup \{b_0 = 1, btd_0 = \top\}\\ &\quad \cup \{b_j = 0, btd_j = \bot \mid j \in [1,N]\}. \end{aligned}$$

Then, for each $i \in [0,N]$ and $j \in [1,N]$, we have the actions fw_i, bw_i and xc_j modelling respectively the running forward and backward of r_i, and the baton exchange between r_{j-1} and r_j, where

$$\begin{aligned} fw_i &: \langle\{x_i < L \times (i+1), b_i > 0\}, \{x_i \mathrel{+}= 1\}\rangle,\\ bw_i &: \langle\{x_i > L \times i, b_i > 0\}, \{x_i \mathrel{-}= 1\}\rangle,\\ xc_j &: \langle\{x_j = x_{j-1}, b_j + b_{j-1} > 0\},\\ &\qquad \{b_j := b_{j-1}, b_{j-1} := b_j, btd_j := \top\}\rangle. \end{aligned}$$

Assume the pattern $\prec$, using the ARPG construction from the initial state, is

$$\prec = fw_0; bw_0; xc_1; fw_1; bw_1; \ldots; xc_N; fw_N; bw_N. \quad (2)$$

Then, in the $\prec$-encoding of Π, $\mathrm{pre}^{\prec}(A)$ contains, for the actions fw_0, bw_0 and xc_1, formulas entailing

$$\begin{aligned} fw_0 > 0 \rightarrow (x_0 < L) \wedge (b_0 > 0),\\ fw_0 > 1 \rightarrow (x_0 + (fw_0 - 1) < L),\\ bw_0 > 0 \rightarrow (x_0 + fw_0 > 0) \wedge (b_0 > 0),\\ bw_0 > 1 \rightarrow (x_0 + fw_0 - (bw_0 - 1) > 0),\\ xc_1 > 0 \rightarrow (x_0 + rt_0 - lf_0 = x_1) \wedge (b_1 + b_0 > 0), \end{aligned}$$

and likewise for all other actions in $\prec$. For each $j \in [1,N]$, the action xc_j is not eligible for rolling, and thus

$$xc_j = 0 \vee xc_j = 1,$$

belongs to $\mathrm{amo}^{\prec}(A)$. Finally, the frame axioms in $\mathrm{frame}^{\prec}(V_B \cup V_N)$, for each $i \in [0,N]$ and $j \in [1,N]$ are:

$$\begin{aligned} x'_i = x_i + fw_i - bw_i,\ \ btd'_0 = btd_0,\ \ btd'_j = btd_j \vee xc_j,\\ b'_j = \textsc{ite}(xc_j > 0, \textsc{ite}(xc_{j-1} > 0, \textsc{ite}(\ldots), b_{j-1}), b_j). \end{aligned}$$

As the frame axioms make clear, the $\prec$-encoding allows in a single state transition

1. the multiple consecutive execution of the same action, as in the rolled-up encoding (Scala et al. 2016b), and
2. the combination of multiple even contradictory effects on a same variable by different actions, as in the $R^2\exists$ encoding (Balyo 2013; Bofill, Espasa, and Villaret 2016).

Assuming the goal is that all the runners have to touch the baton, i.e.,

$$G = \bigcup_{i=0}^{N} \{btd_i = \top\}, \quad (3)$$

then the $\prec$-encoding of Π with $n = 1$, is satisfied by the assignment setting all the action variables corresponding to fw_i and xc_j to L and 1 respectively, and all the action variables corresponding to bw_i to 0, $i \in [0,N]$, $j \in [1,N]$. However, if the goal also includes returning the baton to the initial runner r_0, i.e., if

$$G = \{b_0 = 1\} \cup \bigcup_{i=0}^{N} \{btd_i = \top\}, \quad (4)$$

the $\prec$-encoding of Π requires a bound $n = N + 1$. This is because returning the baton from r_N to r_0 necessitates a plan where xc_j is executed before xc_{j-1} for $j \in [1,N]$, while their order in the pattern $\prec$ of Eq. 2 is the opposite.

4 Pushing the Envelope

Consider a numeric planning problem Π. The issue highlighted by the motivating example arises because a simple and complete pattern $\prec$ is computed only once starting from the initial state, and then exploited at every step $i \in [0, n-1]$ in the $\prec$-encoding of Π with bound n (the formula $\Pi_n^{\prec}$), without considering that:

1. A non-empty subset P of the set G of goals may have been already satisfied at a step $i < n$.
2. To satisfy the remaining goals in $G \setminus P$, it may be (far) better to use a pattern entirely different from the one used to satisfy the goals in P.
3. To facilitate the solution of the SMT formula, it may be better to discard the actions in the steps $< i$ that are useless for satisfying the goals in P.

Indeed, assuming π is a valid plan, the objective is to find the smallest possible pattern $\prec$ covering π. Ideally, $\prec$ should be the pattern of π. A pattern $\prec$ *covers a plan* π if $\prec$ is a supersequence of the pattern of π. A sequence of actions $\prec$ is the *pattern of a plan* π if $\prec$ is obtained from π by replacing consecutive occurrences of each action a eligible for rolling with a single instance of a. If a pattern $\prec$ covers a valid plan, the formula $\Pi_1^{\prec}$, representing the pattern $\prec$-encoding of Π with bound $n = 1$, is satisfiable.

Theorem 1. *Let Π be a numeric planning problem. Let $\prec$ be a pattern covering a valid plan. $\Pi_1^{\prec}$ is satisfiable.*

Proof. We first prove that considering π as a pattern, the pattern π-encoding Π_1^{π} of Π with bound $n = 1$ is satisfiable. Then we prove that if $\prec$ is the pattern of π, then $\Pi_1^{\prec}$ is satisfiable. Finally, we prove that for any supersequence $\prec$ of the pattern of π, $\Pi_1^{\prec}$ is satisfiable.

First statement. Let $\pi = a_1; \ldots; a_k$. The assignment μ extending the initial state, assigning 1 to all the action variables in $\mathcal{A}^{\pi}$ and assigning the next variables according to the frame axioms in Π_1^{π}, is a model of Π_1^{π}, i.e., it also satisfies $\mathrm{pre}^{\pi}(A)$ and $\mathcal{G}(\mathcal{X}')$. This follows from the fact that for each variable $v \in V_B \cup V_N$, if s_i, $i \in [1,k]$, is the i-th state induced by π, then $s_i(v) = \sigma_i(v)$ once a_i is substituted with 1 in $\sigma_i(v)$, which can be proven by induction on i. If $i = 0$, it is trivial. For $i > 0$, the thesis easily follows from the induction hypothesis and the definitions of s_i and σ_i.

Second statement. Let $\prec$ be a pattern and $\prec'$ be a pattern obtained replacing p consecutive occurrences $a_i; \ldots a_{i+p}$ in $\prec$ of the same action eligible for rolling, with just a_i. Then, if μ is a model of $\Pi_1^{\prec}$ then the assignment μ' which differs from μ only in $\mu'(a_i) = \sum_{j=i}^{i+1} \mu(a_j)$ is a model of

$\Pi_1^{\prec'}$. If σ and σ' are associated to $\prec$ and $\prec'$ respectively, we prove it by showing that for any variable $v \in V_B \cup V_N$, $\sigma_{i+p}(v) = \sigma'_i(v)$ once each $a_i, \ldots, a_{i+p}$ is substituted with $\mu(a_i), \ldots, \mu(a_{i+p})$ in $\sigma_{i+p}(v)$, and a_i is substituted with $\mu'(a_i)$ in $\sigma'_i(v)$. Again, the proof is by induction on p: If $p = 0$, it is trivial. For $p > 0$, the thesis easily follows from the induction hypothesis and the definitions of s_i and σ_i. The thesis then follows, since we can remove all the repeated occurrences of a same action in π obtaining the pattern of π.

Third statement. Let $\prec'$ be the pattern of π and $\prec$ be a supersequence of $\prec'$. Any model of $\Pi_1^{\prec'}$ can be extended to a model of $\Pi_1^{\prec}$ by assigning all the action occurrences in $\prec$ and not in $\prec'$ to 0. □

Naturally, the challenge of finding a pattern that covers a valid plan is as complex as finding the plan itself. In practice, employing an ARPG or, more in general, with any ordering on the set A of actions, we just have a simple and complete pattern computed from a given initial state. Such pattern can be arbitrarily extended, but, for effectiveness, this needs some care, since

1. different patterns, even when one is a permutation of the other, cover different plans, and
2. each newly introduced action in the pattern adds another variable to the encoding, increasing the solution space.

We now show how to symbolically search for a valid plan by iteratively extending and compressing an initial pattern. The final procedure, called PATTY$_{\text{DC}}$, incorporates three ideas:

1. With patterns, in Eq. 1 it is not necessary to duplicate n-times the symbolic transition relation: given the initially computed pattern $\prec_h$ and a pattern $\prec_g$, initially empty, we can iterate the procedure of concatenating $\prec_h$ to $\prec_g$ till $\prec_g$ covers a valid plan.
2. In the above outlined procedure, it is not necessary to keep the same $\prec_h$ at each iteration: given a pattern $\prec_g$ allowing us to reach a state s satisfying a strict subset P of the set G of goals, we can (i) *dynamically* recompute the pattern $\prec_h$ whenever we reach a state s satisfying a strict superset of P, and then (ii) iterate the procedure, terminating when all the goals in G are satisfied.
3. In the above outlined procedure, we can compute a plan π leading from I to the intermediate state s from which $\prec_h$ is recomputed: we can exploit the plan π and use the pattern of π instead of $\prec_g$, thereby eliminating the actions in $\prec_g$ which are not necessary to reach the state s.

In the following, we present procedures that exploit the three aforementioned ideas. This allows us to formally state their correctness and completeness. Additionally, the first two procedures correspond to ablation studies of the third, which incorporates all three ideas. The correctness and completeness of the procedures rely on the following theorem. A pattern $\prec$ is *n-complete* if $\prec$ is a supersequence of a pattern obtained by concatenating n simple and complete patterns.

Theorem 2. *Let Π be a numeric planning problem having a valid plan of length n. Let $\prec$ be a n-complete pattern. $\Pi_1^{\prec}$ is satisfiable.*

Algorithm 1 PATTY$_{\text{S}}$ algorithm. Input: a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$. Output: a valid plan for Π.

```
1: function PATTY_S(Π)
2:     ≺_g ← ≺_h ← COMPUTEPATTERN(I, A, G)
3:     while TRUE do
4:         Π^{≺_g} ← I(X) ∧ T^{≺_g}(X, A^{≺_g}, X') ∧ G(X')
5:         μ ← SOLVE(Π^{≺_g})
6:         if μ ≠ 0 then
7:             return GETPLAN(μ, ≺_g)
8:         ≺_g ← ≺_g; ≺_h
```

Proof. Let π be a plan of length n and $\prec$ be an n-complete pattern. By definition, π is a supersequence of the pattern of π. Also, π is a subsequence of $\prec$ since for the i-th action in π we can take its occurrence in the i-th complete pattern in $\prec$. Thus, $\prec$ covers π and the thesis follows from Theorem 1. □

Since we are going to present procedures that extend the pattern $\prec$ while keeping $n = 1$ in Eq. 1, we will simply refer to $\Pi_1^{\prec}$, $\mathcal{X}_0$, $\mathcal{A}_0^{\prec}$ and $\mathcal{X}_1$ as $\Pi^{\prec}$, $\mathcal{X}$, $\mathcal{A}^{\prec}$ and $\mathcal{X}'$, respectively.

4.1 Concatenating Static Patterns

Algorithm 1 shows the pseudocode for PATTY$_{\text{S}}$, the version of PATTY that incorporates the idea of iteratively concatenating a *statically* computed initial pattern. In PATTY$_{\text{S}}$:

1. COMPUTEPATTERN(s, A, G) returns a simple and complete pattern, that in practice we compute using the ARPG construction starting from the state $s = I$. The goal G is passed as a parameter because the ARPG construction can sometimes immediately reveal that the problem Π is not solvable, i.e., that G is not reachable from state s (see (Scala et al. 2016a)). To simplify the algorithm, we omit the check for this case.
2. SOLVE$(\Pi^{\prec_g})$ calls an SMT solver which returns a model of the given formula if it is satisfiable, and 0 otherwise.
3. GETPLAN$(\mu, \prec_g)$ returns the sequence of actions ordered as in $\prec_g$, each action a_i repeated $\mu(a_i)$ times.

In PATTY$_{\text{S}}$, the pattern $\prec_h$ is statically computed only once in the initial state I and we start considering the pattern $\prec_g$ equal to $\prec_h$. At Line 5 we check for satisfiability of the formula $\Pi^{\prec_g}$, i.e., we check if G can be satisfied considering $\prec_g$. If no model is returned, i.e., if $\Pi^{\prec_g}$ is not satisfiable, we skip to Line 8, and we concatenate $\prec_h$ to $\prec_g$ and we start again from Line 4. The pattern $\prec_g$ is thus continuously extended, each time concatenating $\prec_h$ to it, till $\Pi^{\prec_g}$ becomes satisfiable. By Theorem 2, if a valid plan of length n exists, we are guaranteed to find it after at most n iterations and calls to SOLVE$(\Pi^{\prec_g})$. Once a satisfiable model μ is found, a valid plan is returned at Line 7.

PATTY$_{\text{S}}(\Pi)$ is *correct* (any returned plan is valid) and *complete* (if a valid plan exists, PATTY$_{\text{S}}(\Pi)$ will return one).

Theorem 3. *Let Π be a numeric planning problem.* PATTY$_{\text{S}}(\Pi)$ *is correct and complete.*

Proof. Correctness follows from the correctness of the encoding (Theorem 2 in (Cardellini, Giunchiglia, and Maratea

Algorithm 2 PATTY$_{\text{D}}$ algorithm. Input: a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$. Output: a valid plan for Π.

```
1: function PATTY_D(Π)
2:     ≺_g ← ϵ, P ← ∅
3:     ≺_h ← COMPUTEPATTERN(I, A, G)
4:     while TRUE do
5:         ≺_f ← ≺_g; ≺_h
6:         μ ← MAXSOLVE(I(X) ∧ T^{≺_f}(X, A^{≺_f}, X'), G, P)
7:         if |SATG(μ, G)| = |G| then
8:             return GETPLAN(μ, ≺_f)
9:         else if |SATG(μ, G)| > |P| then
10:            ≺_g ← ≺_f
11:            P ← SATG(μ, G)
12:            s ← GETSTATE(I, GETPLAN(μ, ≺_f))
13:            ≺_h ← COMPUTEPATTERN(s, A, G)
14:        else
15:            ≺_g ← ≺_f
```

2024a)). For completeness, at each iteration the complete $\prec_h$ is concatenated to $\prec_g$. Thus, a valid plan of length n is found at most at the n-th iteration, since at that point $\prec_g$ is n-complete, and the thesis follow from Theorem 2. □

4.2 Concatenating Dynamic Patterns

As the motivating example makes clear, having a single pattern may have a dramatic impact on the number of iterations and calls to SOLVE in Algorithm 1. In our example, the pattern computed from the initial state allows finding a plan satisfying $|G-1|$ out of the $|G|$ goals in the first iteration, but we struggle to satisfy the last goal: once the baton is being held by r_N, any alternative pattern, including a random one, enables the computation of a plan to return the baton back to r_0 in no more, and likely fewer, iterations.

Algorithm 2 shows the pseudocode for PATTY$_{\text{D}}$, the version of PATTY that incorporates the idea to *dynamically* update the pattern by recomputing it whenever new goals are achieved. In PATTY$_{\text{D}}$,

1. MAXSOLVE$(\mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec_f}(\mathcal{X},\mathcal{A}^{\prec_f},\mathcal{X}'), G, P)$ calls a MAX-SMT solver returning an assignment satisfying $\mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec_f}(\mathcal{X},\mathcal{A}^{\prec_f},\mathcal{X}')$, all the goals in P, and a maximal subset of the goals in $G \setminus P$.
2. GETSTATE(I, π) returns the state resulting from the execution of the sequence π of actions from I.
3. SATG(μ, G) returns the set of goals in G satisfied by the assignment μ.

In PATTY$_{\text{D}}$, before the search starts, we assign $\prec_g$ to the empty pattern (ϵ), the set P (meant to contain the subset of goals that can be satisfied with $\prec_g$) to the empty set, and we compute an initial pattern $\prec_h$ from the initial state. Then,

1. we set the pattern $\prec_f$ used for the search to $\prec_g; \prec_h$ (Line 5) and then check whether all the goals in G are satisfied (Line 7) and,
2. if not, we check whether $\prec_f$ allows satisfying at least one more goal (Line 9), in which case we
   (a) set $\prec_g$ to $\prec_f$,
   (b) update the set P to the new subset of satisfied goals,
   (c) update the intermediate state,
   (d) recompute $\prec_h$ from s, and
   (e) restart the loop thereby concatenating the newly computed $\prec_h$ at the next iteration,
3. otherwise, (Line 15) we set $\prec_g$ to $\prec_f$, thereby concatenating $\prec_h$ once more at the next iteration.

Algorithm 3 PATTY$_{\text{DC}}$ algorithm. Input: a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$. Output: a valid plan for Π.

```
1: function PATTY_DC(Π)
2:     ≺_g ← ϵ, P ← ∅
3:     ≺_h ← COMPUTEPATTERN(I, A, G)
4:     while TRUE do
5:         ≺_f ← ≺_g; ≺_h
6:         μ ← MAXSOLVE(I(X) ∧ T^{≺_f}(X, A^{≺_f}, X'), G, P)
7:         if |SATG(μ, G)| = |G| then
8:             return GETPLAN(μ, ≺_f)
9:         else if |SATG(μ, G)| > |P| then
10:            ≺_g ← COMPRESS(GETPLAN(μ, ≺_f))
11:            P ← SATG(μ, G)
12:            s ← GETSTATE(I, GETPLAN(μ, ≺_f))
13:            ≺_h ← COMPUTEPATTERN(s, A, G)
14:        else
15:            ≺_g ← ≺_f
```

We prove that PATTY$_{\text{D}}(\Pi)$ is correct and complete.

Theorem 4. *Let* Π *be a numeric planning problem.* PATTY$_{\text{D}}(\Pi)$ *is correct and complete.*

Proof. Correctness follows from the correctness of the encoding (Theorem 2 in (Cardellini, Giunchiglia, and Maratea 2024a)). For completeness, assume there exists a valid plan of length n. If $|\text{SATG}(\mu, G)| < |G|$ at each step, we keep concatenating a complete pattern to $\prec_f$. Thus, at the n-iteration, $\prec_f$ is n-complete. The thesis follows from the fact that MAXSOLVE at Line 6 in PATTY$_{\text{D}}$ returns a model with the highest number of goals in G satisfied. □

4.3 Compression of the Pattern

In our example, though PATTY$_{\text{D}}$ can find a valid plan with just two iterations, it is still possible to further optimize it by compressing the pattern $\prec_f$ that led to the state s in which a new goal has been satisfied. Indeed, we can exploit the plan π that led to s and use the *pattern of* π. The pattern of π is the sequence π with all continuous repetitions of an action removed – since they can be handled by rolling – (e.g., $a; a; b; c; c$ becomes $a; b; c$). Being a subsequence of $\prec_f$ it may not allow reaching all the states that are reachable with $\prec_f$, on the other hand, by reducing the number of variables in the encoding, the task of the SMT solver becomes easier.

Algorithm 3 shows the pseudocode for PATTY$_{\text{DC}}$, which is the same as PATTY$_{\text{D}}$, except that it incorporates the idea of *compressing* the dynamically computed pattern. In this case, we have at Line 10 that $\prec_g$ assumes the value of COMPRESS(π), with π = GETPLAN$(\mu, \prec_f)$. COMPRESS(π) returns the pattern of π. PATTY$_{\text{DC}}(\Pi)$ is correct and complete.

Theorem 5. *Let* Π *be a numeric planning problem.* PATTY$_{\text{DC}}(\Pi)$ *is correct and complete.*

	Solved (out of 20)				Time (s)				SMT calls				Variables				Assertions			
Domain	P_{DC}	P_D	P_S	P_O	P_{DC}	P_D	P_S	P_O	P_{DC}	P_D	P_S	P_O	P_{DC}	P_D	P_S	P_O	P_{DC}	P_D	P_S	P_O
BLGRP (S)	**20**	**20**	**20**	**20**	**1.8**	**1.8**	**1.8**	**1.8**	**1.0**	**1.0**	**1.0**	**1.0**	**174**	**174**	**174**	**174**	**436**	**436**	**436**	**436**
CNT (S)	**20**	**20**	**20**	**20**	**0.9**	**0.9**	**0.9**	**0.9**	**1.0**	**1.0**	**1.0**	**1.0**	**90**	**90**	**90**	**90**	**225**	**225**	**225**	**225**
CNT (L)	**20**	**20**	**20**	**20**	1.0	1.0	**0.9**	**0.9**	**1.0**	**1.0**	**1.0**	**1.0**	**94**	**94**	**94**	**94**	**211**	**211**	**211**	**211**
DEL (S)	**8**	5	5	5	151.2	179.4	**139.3**	208.5	**2.2**	**2.2**	**2.2**	**2.2**	**487**	887	887	1.0k	**1.4k**	3.0k	3.0k	3.1k
DRN (S)	**16**	3	3	3	**83.5**	255.3	255.3	255.3	8.3	**5.7**	**5.7**	**5.7**	**54**	95	95	142	**137**	260	260	344
EXP (S)	**4**	**4**	**4**	2	248.9	**244.0**	244.1	270.2	4.0	**3.0**	**3.0**	**3.0**	200	**193**	**193**	254	554	**529**	**529**	612
FARM (S)	**20**	**20**	**20**	**20**	**0.8**	**0.8**	2.9	2.1	**1.0**	**1.0**	**1.0**	**1.0**	**63**	**63**	**63**	**63**	**120**	**120**	**120**	**120**
FARM (L)	**20**	**20**	**20**	**20**	2.8	2.7	**2.1**	2.3	**1.0**	**1.0**	**1.0**	**1.0**	**81**	**81**	**81**	**81**	**146**	**146**	**146**	**146**
HPWR (S)	**20**	**20**	**20**	**20**	19.9	**18.9**	26.5	19.1	**1.0**	**1.0**	**1.0**	**1.0**	**444**	**444**	**444**	**444**	**788**	**788**	**788**	**788**
MRKT (L)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MPRIME (S)	**15**	13	12	12	130.5	**121.8**	130.1	142.9	1.2	**1.1**	**1.1**	**1.1**	1.5k	**1.5k**	**1.5k**	1.5k	4.6k	**4.6k**	**4.6k**	4.6k
PATHM (S)	**20**	**20**	17	**20**	5.1	**4.8**	51.2	8.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.3k**	**1.3k**	**1.3k**	**1.3k**	**2.2k**	**2.2k**	**2.2k**	**2.2k**
PLWAT (S)	**20**	7	6	6	**11.0**	206.4	226.4	215.5	10.2	8.3	**8.2**	**8.2**	**184**	304	299	416	**511**	872	856	1.1k
RVR (S)	14	10	13	**16**	106.5	166.2	**93.3**	100.7	**1.4**	**1.4**	**1.4**	**1.4**	**485**	570	570	638	**1.1k**	1.4k	1.4k	1.5k
SAIL (S)	**20**	**20**	**20**	**20**	**1.0**	1.6	1.3	6.4	**3.3**	**3.3**	**3.3**	**3.3**	**66**	110	110	135	**122**	241	241	266
SAIL (L)	**20**	**20**	**20**	**20**	**2.5**	5.7	5.7	6.1	**1.4**	**1.4**	**1.4**	**1.4**	**63**	68	68	72	**154**	170	170	175
STLRS (S)	**15**	**15**	8	7	84.4	**83.7**	215.1	219.5	**1.0**	**1.0**	**1.0**	**1.0**	**1.5k**	**1.5k**	**1.5k**	**1.5k**	**3.1k**	**3.1k**	**3.1k**	**3.1k**
SGR (S)	**20**	**20**	**20**	**20**	**5.0**	7.7	9.3	11.1	3.4	2.9	**2.5**	**2.5**	**720**	913	844	1.0k	**1.9k**	2.6k	2.4k	2.6k
TPP (L)	**2**	**2**	**2**	**2**	270.4	273.8	270.4	**270.1**	**2.5**	**2.5**	**2.5**	**2.5**	**149**	207	207	237	**419**	621	621	651
ZENO (S)	**11**	**11**	**11**	**11**	**136.4**	137.1	136.6	136.5	**1.6**	**1.6**	**1.6**	**1.6**	**363**	505	505	542	**1.1k**	1.7k	1.7k	1.7k
LINEEX (L)	**20**	**20**	**20**	**20**	1.2	**1.0**	**1.0**	1.1	5.2	**2.9**	**2.9**	**2.9**	148	**127**	**127**	167	393	**329**	**329**	395
All domains	**325**	290	281	284	**73.5**	94.2	99.6	102.8	2.1	**1.8**	**1.8**	**1.8**	**361.6**	397.3	392.1	422.2	**847.7**	977.8	959.2	1.0k

Table 1: Comparative analysis between PATTY_{DC} (P_{DC}), PATTY_D (P_D), PATTY_S (P_S) and PATTY_O (P_O). The labels (S) and (L) indicate if the planning problem is Simple or Linear, according to the IPC definition. A "-" means that no problem in the domain was solved by the planner.

Proof. Correctness follows from the correctness of the encoding (Theorem 2 in (Cardellini, Giunchiglia, and Maratea 2024a)). For completeness, assume there exists a valid plan of length n. Each time Line 15 is executed, the last computed complete pattern $\prec_h$ is concatenated to $\prec_f$ at Line 5, while each time Line 10 is executed $\prec_f$ is compressed. However, Line 10 is executed only when the number of satisfied goals in G increases. Thus, Line 10 can be executed at most $(|G|-1)$ times, each time after at most after $(n-1)$ iterations between two consecutive executions (since after n execution of Line 5, $\prec_f$ would be n-complete and all the goals in G would be satisfiable). Thus, a valid plan will be found at most at the $p+n$-th iteration, where in the worst case $p = (|G|-1) \times (n-1)$. □

5 PATTY_{DC} on the Motivating Example

Consider the problem Π in the motivating example in which the goal is that all the runners have to touch the baton, i.e., with goal G as in Eq. 3. Assume $\text{COMPUTEPATTERN}(I, A, G)$ returns the complete pattern:

$$\prec_h = fw_0; bw_0; xc_1; fw_1; bw_1; \ldots; xc_N; fw_N; bw_N.$$

In such hypotheses, $\Pi^{\prec}$ is satisfiable and PATTY_S/PATTY_D/PATTY_{DC} will return a valid plan at the first iteration of the main loop.

Now consider the problem Π in the motivating example in which the baton has to be touched by all the runners and to be returned to the hands of the first runner r_0, i.e., with goal G as in Eq. 4. We assume $\text{COMPUTEPATTERN}(I, A, G)$ returns the same pattern as before. Then,

1. PATTY_S at each iteration $i \in [1, N]$ extends the initial pattern $\prec_g$ by considering $\prec_g = \prec_h^i$ in which $\prec_h^i$ denotes the pattern $\prec_h$ concatenated i times. When $i = N$, $\prec_h^i$ is N-complete, $\prec_h^N$ covers the plan for returning the baton from r_{N+1} back to r_0, and $\Pi^{\prec_g}$ is satisfiable. The pattern computed by $\text{COMPUTEPATTERN}(I, A, G)$ contains $3N+2$ actions, and thus PATTY_S can find a valid plan when considering a pattern with $N \times (3N+2)$ actions.
2. PATTY_D starts with $\prec_f = \prec_h$ and at the first iteration it will satisfy the goals of having the baton touched by all runners by reaching a state s satisfying, for each $i \in [0, N)$, $x_i = L \times (i+1)$, $x_N \in [L \times N, L \times (N+1)]$, $b_0 = \ldots = b_{N-1} = 0$, $b_N = 1$ and $btd_0 = \ldots = btd_N = 1$. Assuming $s(x_N) = L \times (N+1)$, the new pattern $\prec_h'$ computed in s is:
$$\prec_h' = bw_N; xc_N; fw_N; \ldots; bw_1; xc_1; fw_1; bw_0; fw_0,$$
and PATTY_D will find a valid plan with $\prec_f = \prec_h; \prec_h'$. Every pattern computed by $\text{COMPUTEPATTERN}(s, A, G)$ contains $3N+2$ actions, and thus PATTY_D is able to find a valid plan when considering a pattern with $2 \times (3N+2) = 6N+4$ actions.
3. PATTY_{DC} starts with $\prec_f = \prec_h$ as PATTY_D. Differently from PATTY_D, it will compute the pattern $\prec_\pi$ of the plan π that led to the state s in which $\prec_h'$ was computed, i.e.,
$$\prec_\pi = fw_0; xc_1; fw_1; \ldots; xc_N; fw_N$$
and PATTY_{DC} will find a valid plan with $\prec_f = \prec_\pi; \prec_h'$. The pattern $\prec_\pi$ contains $(2N+1)$ actions, and thus PATTY_{DC} is able to find a valid plan when considering a pattern with $(2N+1)+(3N+2) = 5N+3$ actions.

6 Experimental Results

For the experiments, we considered the settings and 20 domains and 20 problems per domain used in the Agile

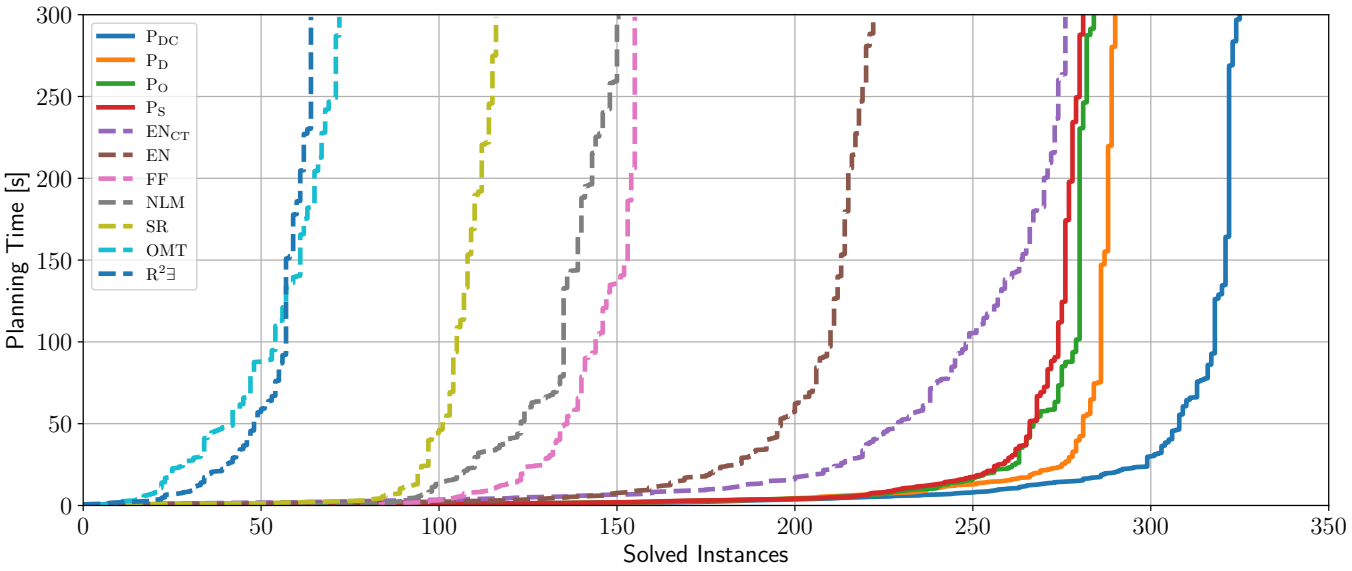

Figure 1: Number of problems solved (x-axis) within a given time (y-axis). Planner names are abbreviated. SPP planners are represented with solid lines. The legend is sorted by total instances solved.

	Solved (out of 20)					Time (s)				
Domain	P_{DC}	EN_{CT}	EN	FF	NLM	P_{DC}	EN_{CT}	EN	FF	NLM
BLGRP (S)	**20**	14	16	2	-	**1.8**	117.2	81.5	270.2	-
CNT (S)	**20**	10	12	15	11	**0.9**	163.8	133.8	95.7	149.8
CNT (L)	**20**	12	10	8	6	**1.0**	142.3	170.9	180.0	214.0
DEL (S)	8	14	13	**18**	9	151.2	117.1	121.7	**41.2**	165.2
DRN (S)	16	**18**	16	2	16	83.5	**55.4**	62.9	268.4	66.0
EXP (S)	4	**6**	**6**	-	3	248.9	224.0	**212.3**	-	253.7
FARM (S)	**20**	**20**	**20**	9	15	**0.8**	1.8	0.9	188.1	85.3
FARM (L)	**20**	**20**	18	15	11	2.8	**2.5**	48.6	80.5	151.2
HPWR (S)	**20**	**20**	2	1	1	19.9	**4.6**	270.3	285.0	285.1
MRKT (L)	-	**20**	4	-	-	-	**35.0**	259.3	-	-
MPRIME (S)	15	**17**	**17**	**17**	14	130.5	74.6	68.1	**45.1**	127.2
PATHM (S)	**20**	3	2	10	1	**5.1**	262.8	272.2	154.9	284.2
PLWAT (S)	**20**	**20**	16	3	14	**11.0**	41.1	101.3	268.3	167.2
RVR (S)	**14**	12	8	10	4	**106.5**	143.7	197.4	133.3	240.8
SAIL (S)	**20**	**20**	**20**	1	10	**1.0**	5.0	2.0	285.0	150.3
SAIL (L)	**20**	2	2	8	15	**2.5**	270.8	270.6	182.8	96.8
STLRS (S)	**15**	2	1	4	-	**84.4**	279.0	288.6	243.8	-
SGR (S)	**20**	11	8	13	4	**5.0**	144.5	182.5	122.5	245.7
TPP (L)	2	**7**	3	2	2	270.4	**212.3**	255.2	266.7	270.0
ZENO (S)	11	17	**19**	11	9	136.4	89.5	**28.1**	135.0	172.5
LINEEX (L)	**20**	11	9	6	6	**1.2**	149.5	175.4	211.6	235.0
All domains	**325**	276	222	155	151	**73.5**	120.8	152.5	193.4	202.7

Table 2: Comparative analysis of PATTY_{DC} (P_{DC}) and the search based planners ENHSP_{CT} (EN_{CT}), ENHSP (EN), FF and NLM. A "-" means that no problem in the domain was solved by the planner in the cut-off time.

track of the last 2023 Numeric IPC, and also the 20 problems in the line exchange domain introduced in (Cardellini, Giunchiglia, and Maratea 2024a). Thus, each system had a time limit of 5 minutes on an Intel Xeon Platinum 8000 3.1GHz with 8 GB of RAM.

About the systems, we first considered PATTY_S, PATTY_D and PATTY_{DC} described in the previous section, and the Planning as Satisfiability planners

1. OMTPLAN (Leofante et al. 2020),
2. the original SPP procedure PATTY_O from (Cardellini, Giunchiglia, and Maratea 2024a),
3. the system $R^2\exists$ by (Cardellini, Giunchiglia, and Maratea 2024a) implementing the relaxed-relaxed-$\exists$ encoding proposed in (Bofill, Espasa, and Villaret 2017), and
4. SPRINGROLL (Scala et al. 2016b).

For these 7 planners, we used `Z3` v4.8.7 (De Moura and Bjørner 2008) for computing the model (if any) satisfying the given set of SMT assertions (representing the hard constraints of the encodings) and also the maximum number of soft assertions (representing the goals in PATTY_D and PATTY_{DC}). Table 1 presents results only for PATTY_O, PATTY_S, PATTY_D, and PATTY_{DC}, given that OMTPLAN, $R^2\exists$, and SPRINGROLL underperformed compared to PATTY_O on every single domain. In the first three subtables of Table 1, we show: the name of the domain (subtable Domain); the number of solved problems (subtable Solved); the average time to find a solution, counting the time limit when the solution could not be found (subtable Time). In the last three subtables, we considered only the problems solved by all the planners and show the average number of calls to the SMT solver (subtable SMT calls); the number of variables (subtable Variables) and assertions (subtable Assertions) of the encoding when a solution is found.

Looking at the performance results, the first observation is that PATTY_{DC} has the best results: PATTY_{DC} can solve the highest number of problems in each domain but one, improving the number of solved problems in 6 domains (out of the 10 which PATTY_O did not completely solve). Then, we observe that

1. comparing PATTY_S vs PATTY_O, the elimination of the intermediate state variables obtained by the concatenation of the pattern, may affect performance, overall leading to a modest degradation,
2. comparing PATTY_{DC} vs PATTY_D/PATTY_S/PATTY_O, the compression of the pattern plays a major role in PATTY_{DC}

positive performance, and

3. none of PATTY$_{DC}$/PATTY$_{D}$/PATTY$_{S}$/PATTY$_{O}$ can solve a problem before the cut-off in the MRKT(L) domain – where a trader has to go back and forth from different locations buying and selling articles, similarly to our motivating example – since all the problems have a single goal (i.e., $|G| = 1$) and thus the pattern is never recomputed during the search by PATTY$_{DC}$/PATTY$_{D}$.

Examining the subtable with the number of calls made to the SMT solver, we observe that PATTY$_{O}$ and PATTY$_{S}$ yield identical values, while the count for PATTY$_{D}$ is never higher than that of PATTY$_{DC}$. No strict relationship emerges when comparing PATTY$_{O}$/PATTY$_{S}$ with PATTY$_{D}$ and PATTY$_{DC}$. These results align with the theoretical findings. As expected, the total number of variables and the total number of assertions utilized in the final call to the SMT solver are typically lower for PATTY$_{DC}$ than for PATTY$_{O}$/PATTY$_{S}$/PATTY$_{D}$, even on domains in which PATTY$_{DC}$ needs more iterations/calls to the SMT solver. The number of variables and assertions used by PATTY$_{O}$ are always higher than those used by PATTY$_{S}$, as PATTY$_{O}$ uses state variables representing intermediate states, along with the corresponding assertions. Conversely, the average number of variables per assertion is higher for PATTY$_{S}$ than for PATTY$_{O}$, being 16.9 and 7.3 respectively.

We compared PATTY$_{DC}$ with the search-based planners

1. ENHSP with the configurations `sat-hadd` (Scala, Haslum, and Thiébaux 2016), `sat-aibr` (Scala et al. 2016a) and `sat-hmrphj` (Scala et al. 2020) – the last one inspired on the helpful actions of the METRICFF planner (Hoffmann 2003) – considering for each problem its best resulting configuration,
2. the very recent extension of the ENHSP planner introduced in (Chen and Thiébaux 2024) (that we call ENHSP$_{CT}$),
3. METRICFF (FF) (Hoffmann 2003), and
4. NLM-CUTPLAN (NLM) in the `sat` configuration which won the Agile track of that last IPC (Kuroiwa, Shleyfman, and Beck 2022).[2]

Table 2 shows for each planner the number of problem it solves and the average time it takes, the latter computed as before. As it can be seen, PATTY$_{DC}$ solves more problems on 14 domains, compared to the 10 by ENHSP$_{CT}$, 5 by ENHSP, and 2 by FF. Overall, PATTY$_{DC}$/ENHSP/ENHSP$_{CT}$/FF/NLM can solve 325/276/222/155/151 of the 420 problems we considered, respectively. Interestingly, every planner can solve more problems than the other search-based planners in at least one domain, pointing out the importance of the specific heuristics used. In particular, ENHSP$_{CT}$ heuristics are particularly effective on the MRKT (L) domain, whose problems are unsolvable by PATTY$_{DC}$. On the other hand, the SPP approach has a significant advantage over search-based planners on other domains, like SAIL (L).

[2]For ENHSP, ENHSP$_{CT}$ and NLM, we got in touch with the authors about which configuration to use for their planners. See https://ipc2023-numeric.github.io/results/presentation.pdf for IPC results.

The cactus plot in Figure 1 summarizes the performance of all the systems considered, showing how many problems each solves within a given time. As it can be seen, (i) symbolic planners not based on patterns (i.e., $R^2\exists$, OMT, SPRINGROLL) solve the fewest problems, (ii) search-based approaches (i.e., NLM, FF, ENHSP, ENHSP$_{CT}$) are in the middle sections, and (iii) pattern-based approach outperform all other planners with PATTY$_{DC}$ being the leader. Clearly, different figures can be obtained by considering different domains/problems, especially if comparing symbolic vs search-based planners. Indeed, as also the above results point out, depending on the domain, search-based planners may perform far better/worse than symbolic planners.

We also experimented with a 1800s time-limit, obtaining the same overall picture.

7 Conclusions and Future Work

In this paper, we introduced a novel symbolic procedure for numeric planning, the first exploiting Planning as Search techniques in a Planning as Satisfiability setting. We proved its correctness and completeness. We showed that it performs comparatively well with respect to all the available numeric planners on the benchmarks of the 2023 IPC Agile Numeric Track, and we performed ablation studies. This work opens new avenues for future research on leveraging search techniques in symbolic planning. Indeed, this is the first work where the choice of how to encode the planning problem as a logic formula is guided by search. As the poor performance of PATTY$_{DC}$/PATTY$_{D}$/PATTY$_{S}$ on MRKT (L) show, more research is needed to deal with such problems where there is only one subgoal, for which the strategies we propose here are ineffective. Despite this, we have shown that PATTY$_{DC}$ achieves remarkably good performance, even when compared to state-of-the-art search-based planners that benefit from decades of research into the design and implementation of effective heuristics. We are currently working on how to further exploit such research in our setting, checking whether it produces corresponding benefits. Moreover, it could be very interesting to explore how to modify the SMT solver to exploit planning-specific branching heuristics, as proposed in (Giunchiglia, Massarotto, and Sebastiani 1998; Rintanen 2012).

In this paper, we have mainly concentrated on the numeric planning fragment, which is a superset of the classical planning fragment (Haslum et al. 2019) where variables can only be propositional. It is clear that our approach can be used, as-is also for classical planning. However, preliminaries results run on the 2023 Classical IPC (Taitler et al. 2024) show that even PATTY$_{DC}$ is not yet competitive in the contemporary classical setting. In this fragment, the pattern-based approach must be extended to leverage the specific characteristics of the language more effectively (e.g., conditional effects) and to be able to deal with problems with thousands of actions, thus challenging to plan with ground actions in a satisfiability-based approach where the variables become too many to handle. We plan in the future to explore a lifted version (Höller and Behnke 2022) of our solver to be able to deal with these huge problems.

Acknowledgments

Matteo Cardellini and Enrico Giunchiglia were each partially supported by the projects FAIR (PE00000013) and SERICS (PE00000014), respectively, under the NRRP MUR program funded by the EU - NGEU.

References

Alarnaouti, D.; Percassi, F.; and Vallati, M. 2024. An extensive empirical analysis of macro-actions for numeric planning. In *International Conference of the Italian Association for Artificial Intelligence*, 23–36. Springer.

Balyo, T. 2013. Relaxing the Relaxed Exist-Step Parallel Planning Semantics. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, 865–871. Herndon, VA, USA: IEEE.

Barrett, C.; Fontaine, P.; and Tinelli, C. 2016. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`.

Bofill, M.; Espasa, J.; and Villaret, M. 2016. The rantanplan planner: system description. *The Knowledge Engineering Review* 31(5):452–464.

Bofill, M.; Espasa, J.; and Villaret, M. 2017. Relaxed exists-step plans in planning as SMT. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 563–570. ijcai.org.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Cardellini, M., and Giunchiglia, E. 2025. Temporal numeric planning with patterns. *Proceedings of the AAAI Conference on Artificial Intelligence* 39(25):26481–26489.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2024a. Symbolic numeric planning with patterns. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, 20070–20077. AAAI Press.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2024b. Symbolic numeric planning with patterns. In *AAAI'24*.

Chen, D. Z., and Thiébaux, S. 2024. Novelty heuristics, multi-queue search, and portfolios for numeric planning. In Felner, A., and Li, J., eds., *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, 203–207. AAAI Press.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

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.

Giunchiglia, E.; Massarotto, A.; and Sebastiani, R. 1998. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In Mostow, J., and Rich, C., eds., *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*, 948–953. AAAI Press / The MIT Press.

Haslum, P.; Nir, L.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*.

Hoffmann, J. 2003. The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of artificial intelligence research* 20:291–341.

Höller, D., and Behnke, G. 2022. Encoding Lifted Classical Planning in Propositional Logic. *Proceedings of the International Conference on Automated Planning and Scheduling* 32:134–144.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In Neumann, B., ed., *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, 359–363. John Wiley and Sons.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In Clancey, W. J., and Weld, D. S., eds., *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, 1194–1201. AAAI Press / The MIT Press.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-cut heuristics for optimal linear numeric planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 203–212.

Leofante, F.; Giunchiglia, E.; Ábráham, E.; and Tacchella, A. 2020. Optimal planning modulo theories. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 4128–4134. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artif. Intell.* 193:45–86.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016a. Interval-based relaxation for general numeric planning. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 655–663.

Scala, E.; Ramirez, M.; Haslum, P.; and Thiebaux, S. 2016b. Numeric planning with disjunctive global constraints via SMT. *Proceedings of the International Conference on Automated Planning and Scheduling* 26:276–284.

Scala, E.; Saetti, A.; Serina, I.; and Gerevini, A. E. 2020. Search-guidance mechanisms for numeric planning through subgoaling relaxation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 226–234.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for numeric planning via subgoaling. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Confer-*

ence on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, 3228–3234. IJCAI/AAAI Press.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; et al. 2024. The 2023 international planning competition.

Vidal, V. 2004. The yahsp planning system: Forward heuristic search with lookahead plans analysis. In *International Planning Competition*, 56.