PDBs Go Numeric: Pattern-Database Heuristics for Simple Numeric Planning

Daniel Gnad[1], **Lee-or Alon**[2], **Eyal Weiss**[2,3], **Alexander Shleyfman**[2]

[1]Linköping University, Linköping, Sweden
[2]Bar-Ilan University, Ramat Gan, Israel
[3]Technion–Israel Institute of Technology, Haifa, Israel
daniel.gnad@liu.se, {alonlee1, eyal.weiss, alexash}@biu.ac.il

Abstract

Despite the widespread success of pattern database (PDB) heuristics in classical planning, to date there has been no application of PDBs to planning with numeric variables. In this paper we attempt to close this gap. We address optimal numeric planning involving conditions characterized by linear expressions and actions that modify numeric variables by constant quantities. Building upon prior research, we present an adaptation of PDB heuristics to numeric planning, introducing several approaches to deal with the unbounded nature of numeric variable projections. These approaches aim to restrict the initially infinite projections, thereby bounding the number of states and ultimately constraining the resulting PDBs. We show that the PDB heuristics obtained with our approach can provide strong guidance for the search.

Code — https://github.com/dgnad/numeric-fast-downward
Datasets — https://doi.org/10.5281/zenodo.14502394

Introduction

In this work, we concentrate on simple numeric planning (SNP) with instantaneous actions. Numeric state fluents introduce a degree of complexity over classical planning, making the plan existence problem undecidable in general (Helmert 2002). Nevertheless, since the introduction of numeric variables in PDDL2.1 (Fox and Long 2003), several methods have been proposed to solve satisficing and optimal variants of numeric planning problems (Hoffmann 2003b; Shin and Davis 2005; Gerevini, Saetti, and Serina 2008; Eyerich, Mattmüller, and Röger 2009; Coles et al. 2013; Scala et al. 2016; Illanes and McIlraith 2017; Li et al. 2018; Scala, Haslum, and Thiébaux 2016; Scala et al. 2017; Aldinger and Nebel 2017; Piacentini et al. 2018a,b; Kuroiwa et al. 2022; Kuroiwa, Shleyfman, and Beck 2022). We consider here optimal planning in *simple* numeric planning, where numeric variables can be increased or decreased by constant quantities and preconditions and goals are linear inequalities.

Previous works have utilized heuristics based on pattern databases (PDBs) (Culberson and Schaeffer 1996, 1998) in classical planning (Edelkamp 2002; Holte et al. 2004; Felner, Korf, and Hanan 2004; Haslum, Bonet, and Geffner 2005; Holte et al. 2006; Anderson, Holte, and Schaeffer 2007; Haslum et al. 2007; Katz and Domshlak 2009). A PDB heuristic considers only a subset of the state variables, called the *pattern*, and captures this sub-task exactly. Other variables are projected away in the heuristic calculation.

While showing state-of-the-art performance in classical planning, to the best of our knowledge, PDB heuristics have never been used in the numeric planning setting. Addressing this gap may be desirable for two reasons: (1) multiple other heuristics known from classical planning, such as $h^{\max}$ (Bonet and Geffner 2001), or LM-cut (Helmert and Domshlak 2009), have been adopted successfully in numeric planning, and (2) as PDB heuristics perform very well in classical planning, it is reasonable to assume that their success may be brought to the numeric realm.

Unlike in classical planning, the transition systems induced by a projection of a task onto a set of variables that include numeric fluents have infinitely many states. Thus, to adopt approaches used in classical planning to generate informative PDBs, we need to bound the numeric transition system. We introduce several strategies that can be employed to that end, bounding numeric transition systems effectively. One approach is to restrict the numeric fluents within the transition system, reducing the infinite state space to an incomplete finite one. Another approach defines structure-specific constraints or rules that limit the possible values of numeric fluents during the transition, thereby constraining the state space. Additionally, abstraction techniques can be utilized to represent the numeric transition system at a higher level of granularity, focusing only on relevant aspects while abstracting away unnecessary details. By implementing these strategies, it becomes feasible to generate informative PDBs for tasks involving numeric fluents in planning domains. We discuss these approaches and provide them with a theoretical background.

We evaluate our numeric PDBs by adapting established pattern generation methods and using the canonical heuristic to combine multiple PDBs admissibly. We compare against state-of-the-art optimal planners for simple numeric tasks.

Preliminaries

We work over a fragment of numeric planning based on the finite-domain representation (FDR) formalism (Bäckström and Nebel 1995; Helmert 2009) extended with numeric flu-

ents called *integer-restricted task* (IRT). This is a variant of the *restricted numeric task* (RT) formalism (Hoffmann 2003a), where all values of the numeric state variables are integers. Hoffmann (2003a) shows that RTs are equivalent to simple numeric tasks, and Helmert (2002) demonstrates that any RT can be transformed into an IRT.

An IRT is a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$, where $\mathcal{V} = \mathcal{V}_n \cup \mathcal{V}_p$ is a set of *numeric* and *finite-domain variables*, respectively. $\mathcal{A}$ is the set of the actions of Π, s_0 is the *initial state*, and G is the *goal description*. All these sets are finite. The task with no numeric variables, $\mathcal{V}_n = \emptyset$, is called an FDR task.

Let $v \in \mathcal{V}$, and let $\mathcal{D}(v)$ be its domain. Then, $|\mathcal{D}(v)| < \infty$ if $v \in \mathcal{V}_p$ and $\mathcal{D}(v) = \mathbb{Z}$ otherwise. A *state* of Π is a full assignment over the variables in $\mathcal{V}$. The set of all states is denoted $\mathcal{S}$. Each state $s \in \mathcal{S}$ can be represented as a tuple $\langle s_p, s_n \rangle$, where $s_p \in \times_{v \in \mathcal{V}_p} \mathcal{D}(v)$ and $s_n \in \times_{v \in \mathcal{V}_n} \mathbb{Z}$. $\langle v, d \rangle$ denotes a *fact*, where $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$. We say that $s \models \langle v, d \rangle$ or $\langle v, d \rangle \in s$ iff $s[v] = d$, where $s[v]$ indicates the value of v in s. A state s can be seen as a set of facts, thus as a minor abuse of notation we write $s = s_p \cup s_n$. A set of facts s^{pt} is a *partial state* if $s^{pt} \subseteq s$ and $s \in \mathcal{S}$.

Conditions can be either propositional or numeric. Propositional conditions are formed out of facts $\langle v, d \rangle$, a non contradicting set of such propositions forms a partial state s^{pt}. s^{pt} is satisfied by the state s if $s^{pt} \subseteq s$. Numeric conditions have the form $v \bowtie w$, with $\bowtie \in \{>, \geq, =, \leq, <\}$, $v \in \mathcal{V}_n$, and $w \in \mathbb{Z}$. $s \models v \bowtie w$ if $s[v] \bowtie w$. For a set of conditions Ψ we say $s \models \Psi$ if $s \models \psi$ for each $\psi \in \Psi$.

A tuple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a) \rangle$ forms an action $a \in \mathcal{A}$, where $\mathsf{pre}(a)$ is the *precondition*, $\mathsf{eff}(a)$ is the *effect*, and $\mathsf{cost}(a)$ is the *cost* of a. A precondition $\mathsf{pre}(a) := \mathsf{pre}_p(a) \cup \mathsf{pre}_n(a)$ consists of propositional and numeric conditions, respectively. Similarly, an effect $\mathsf{eff}(a) := \mathsf{eff}_p(a) \cup \mathsf{eff}_n(a)$ is a set of propositional and numeric effects. Numeric effects in an IRT have the form $(v \mathrel{+}= m)$, where $v \in \mathcal{V}_n$ and $m \in \mathbb{Z} \setminus \{0\}$. We say that a is *applicable* in s if $s \models \mathsf{pre}(a)$, the result of this application is $s[\![a]\!] := s'_p \cup s'_n$, with $s'_p[v] = d$ if $\langle v, d \rangle \in \mathsf{eff}_p(a)$, $s'_n[v] = s_n[v] + m$ if $(v \mathrel{+}= m) \in \mathsf{eff}_n(a)$, and $s[\![a]\!][v] = s[v]$ otherwise. We assume that each action has at most one effect on each variable.

The goal description G comprises both propositional conditions, G_p, and numeric conditions, G_n. A state s_* is considered a *goal state* if it satisfies the goal description, i.e., $s_* \models G$. An *s-plan*, denoted as π, is a sequence of actions applied consecutively, starting from the state s and leading to some s_*. A plan for Π is an s_0-plan. The *cost of an s-plan* π is the sum of all its action costs and an *optimal s-plan* has minimal cost among all possible s-plans. The optimal cost of an s-plan is denoted by $h^*(s)$ and called *perfect heuristic*. If no goal state is reachable from s we define $h^*(s) := \infty$.

Let $\mathcal{T} = \langle S, L, \mathsf{cost}, T, s_0, S_* \rangle$ be a directed weighted labeled graph called a *labeled transition system* (LTS), where S denotes the set of states, L is the set of labels, $\mathsf{cost} : L \to \mathbb{R}^{0+}$ is the cost function, $T \subseteq S \times S \times L$ is the set of transitions, $s_0 \in S$ is the initial state, and S_* the set of goal states. By $(s, s', l) \in T$ we denote a transition from s to s' under label l. Note that every planning task Π defines an LTS, which we denote by $\mathcal{T}_\Pi$. The main difference between classical and numeric planning is that in the former there is finite number of states, while in the latter it often occurs that $|S| = \infty$.

A path from $s \in S$ to $s' \in S$, denoted $\mathrm{path}(s, s')$ is a sequence $\langle t_1, \ldots, t_n \rangle$ of transitions such that there exists a sequence of states $s = s_0, \ldots, s_n = s'$ with $t_i = (s_{i-1}, s_i, l_i) \in T$ for all $i \in [n]$. We allow the special case of an empty path from s to s, denoted $\langle\rangle$. The cost of a path is given by $\mathsf{cost}(s, s') := \mathsf{cost}(\mathrm{path}(s, s')) = \sum_{i=1}^{n} \mathsf{cost}(l_i)$, the cost of an empty path is zero, and if no path exists, we set the cost to be infinity. $\mathsf{cost}^*(s, s')$ is the minimal cost of a path from s to s'.

The *state space* of an IRT $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$ is the LTS $\mathcal{T}_\Pi = \langle S, \mathcal{A}, \mathsf{cost}, T, s_0, S_* \rangle$, where $S := \mathcal{S}$, the labels are the action names in $\mathcal{A}$ and the transitions are defined by $(s, s[\![a]\!], a)$ with cost $\mathsf{cost}(a)$. The initial states s_0 coincide and the goal states are $S_* = \{s \in \mathcal{S} \mid s \models G\}$. A plan π for Π corresponds to a path from s_0 to some goal state $s_* \in S_*$. An optimal plan π is given by the cheapest path from s_0 to some $s_* \in S_*$, i.e., $\mathsf{cost}(\pi) = \min_{s_* \in S_*} \mathsf{cost}^*(s, s_*)$.

PDBs for Classical Planning

One of the most employed algorithms to optimally solve planning tasks is A^* search (Hart, Nilsson, and Raphael 1968) with an admissible heuristic, which estimates the cost of reaching a goal state. A heuristic $h : \mathcal{S} \to \mathbb{R}^{0+} \cup \{\infty\}$ is called *admissible* if it assigns every state s an estimate such that $h(s) \leq h^*(s)$. A Pattern Database (PDB) heuristic, denoted as $h\big|_P$, is induced by a subset of variables $P \subseteq \mathcal{V}$ comprising the variables of Π, known as the *pattern*. All variables that are not in the pattern are ignored, thus inducing an abstraction of the state space. A projection maps each state in the original state space to a state in the abstract state space. $h\big|_P(s)$ is defined as the perfect heuristic in the projection $\Pi\big|_P$ of Π onto P. Such a projection can be computed by eliminating all occurrences of variables from $\mathcal{V} \setminus P$ in Π. PDB heuristics are usually precomputed once by determining the optimal solution costs, $h\big|_P(s\big|_P)$, for all abstract states $s\big|_P \in \mathcal{S}\big|_P$ in the abstract planning task $\Pi\big|_P$.

Regression search is commonly used for calculating distances in abstract state spaces instead of progression search. Unlike progression, which starts from the initial abstract state and requires two iterations to compute the heuristic, regression begins from the abstract goal states and works backwards, computing the heuristic in a single exploration. This avoids redundant distance calculations and can identify dead ends, which are the abstract states that are not reachable from a goal state during regression.

During the search process, concrete states s are projected onto the variables in P. A perfect hash function (Sievers, Ortlieb, and Helmert 2012) is used for efficient lookup of the heuristic value. This hash function is well-defined for all partial states s^{pt} defined over the variables in P. PDBs grow exponentially in the number of included variables, which limits the heuristic accuracy of single-PDB heuristics. Consequently, state-of-the-art planners employ various techniques to admissibly combine multiple small PDB heuristics.

Combining Multiple PDBs Let H be a set of admissible heuristics. For any state s a maximum over this set of heuris-

tics is an admissible estimate. We say that H is *additive* if $\sum_{h \in H} h(s)$ is admissible. A sufficient condition for a set of PDB heuristics H to be additive was proposed by Haslum et al. (2007) who introduced the *canonical heuristic* based on the *disjoint additivity of patterns* underlying the PDBs. Two patterns P_1 and P_2 are disjoint-additive if there exists no action a that affects at least one variable in each pattern. For a set (collection) of patterns C, the heuristic h^C is defined as the maximum over the sums of PDB heuristics induced by patterns in maximal disjoint-additive subsets, i.e., $h^C = \max_{M \in \mathcal{M}(C)} \sum_{P \in M} h\big|_P$, where $\mathcal{M}(C)$ is a set of maximal disjoint-additive subsets of C. h^C is admissible.

Causal Graphs for Numeric Planning

Planning tasks are often complex, prompting the use of various methods to study their structure. One such method is the *causal graph* (CG), commonly used in classical planning (e.g. Knoblock (1994); Bacchus and Yang (1994); Brafman and Domshlak (2003)). Shleyfman et al. (2023) applied Helmert's (2004) compact definition of CGs to RTs. Let $\mathit{vars}(\Psi)$ be the set of variables involved in a set of formulas Ψ. For each formula $\psi \in \Psi$, $\mathit{vars}(\psi)$ returns a single variable. Since, in RTs, all conditions and effects, whether numeric or propositional, involve exactly one variable per formula, the CG definition can be applied almost directly.

The CG of an IRT planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$ is a digraph $CG(\Pi) = \langle \mathcal{V}, \mathcal{E} \rangle$ with nodes $\mathcal{V}$, where $(u, v) \in \mathcal{E}$ if $u \neq v$ and there exists an $a \in \mathcal{A}$, s.t. $u \in \mathit{vars}(\mathsf{pre}(a)) \cup \mathit{vars}(\mathsf{eff}(a))$ and $v \in \mathit{vars}(\mathsf{eff}(a))$. In essence, in a causal graph, there is an edge from a variable v to a variable v' if changing the value of v' might necessitate a change in the value of v, showing that v' depends on v.

PDB Heuristics for Integer Variables

Common techniques employed to compute PDBs for classical planning involve computing the entire LTS across the specified projection. In the context of numeric planning, though, even projecting the task onto a subset of variables, with at least one being numeric, may yield an infinite LTS. In this section we introduce several approaches that are capable of dealing with potentially infinite LTSs. We define infinite LTSs and discuss the challenges involved in having PDBs with infinite state spaces.

How to Deal with Numeric Variables?

The first question is which numbers PDB heuristics can handle. We use integers, covering much of numeric planning (Helmert 2002; Hoffmann 2003a; Gnad et al. 2023). But this is not a strict requirement; our approach supports any setting where only a finite set of values needs to be distinguished for each numeric variable. A more significant limitation involves numeric effects. In IRTs, effects are restricted to constant additions. Allowing effects like $x := y$ would cause infinite branching in the abstract LTS if $x \in P$ but $y \notin P$. To avoid this, we transform the task into IRT form, where action effects are independent of other variables.

PDB heuristics compute the abstract state space for a pattern P and minimal goal distances for all abstract states. For this, usually regression search is used for efficiency. In presence of numeric non-goal variables in P or goal expressions using inequalities, however, which imply infinitely many abstract goal states, this fails. We use progression search from the abstract initial state $s_0\big|_P$ to address this, exploring up to N states or until exhaustion. We then compute goal distances using regression. If the state limit N is reached, fringe states lack distances, which we approximate, e.g. by the minimum action cost. After constructing the abstract LTS, we store goal distances in a lookup table with hashing. Details are shown in Algorithm 1. Since only finitely many numeric values are reachable during the LTS construction, numeric variables can be treated as finite-domain, and hashes are computed similar to classical planning.

A downside of partially constructing the abstract state space is that we may visit a concrete state s in the main search without an abstract counterpart $s|_P$ in the PDB. This occurs when an insufficient portion of the domain of numeric variables $x \in P \cap \mathcal{V}_n$ is generated during PDB construction. We propose to ignore this PDB using a heuristic value of 0, since large collections of PDB heuristics are typically used, another PDB heuristic is likely to cover state s.

Exploit Known Bounds of Tractable IRT Fragments

Prior work has established multiple conditions that allow to bound the state space of a numeric planning task (Helmert 2002; Shleyfman, Gnad, and Jonsson 2023; Gigante and Scala 2023; Helal and Lakemeyer 2024). Of particular interest for our work are results that impose conditions on the variables dependencies of a planning task, e. g., by restricting the structure of the CG. If we select the pattern $P \subseteq \mathcal{V}$ such that the projection $\Pi\big|_P$ can be bounded, then we have the guarantee that the abstract state space can be exhausted. For some fragments of RT Shleyfman, Gnad, and Jonsson (2023) provide a recipe to compute the bounded domain intervals, which can be used to estimate the size of a PDB.

Care is needed as ignoring dependencies outside P is an approximation. If a numeric variable $v \in P$ depends cyclically on a variable not in P, the projection's bound may be invalid in the full task. Even valid bounds might fail if states in the search exceed them. CG analyses consider in-bound values sufficient but do not ensure others are unreachable. We address this by discarding PDBs for out-of-bound states.

Homomorphisms of Infinite LTSs

Since numeric tasks typically induce infinite LTSs, we present below a standard definition for mapping one LTS to another, bearing in mind that in our case the number of states is infinite. We argue that the distance-to-goal in the image LTS is an admissible heuristic in the original one.

LTSs can be viewed as first-order relational structures used in logic and model theory. Formally, an LTS $\mathcal{T} = \langle S, L, \mathsf{cost}, T, s_0, S_* \rangle$ is a relational structure defined in $S \cup L$ endowed with unary predicates that define states and labels, a ternary relation T, a constant s_0, and a unary predicate S_*. Thus, we can use the standard definition of homomorphism from model theory. In the planning literature, LTS homomorphisms are also called abstractions (Libkin 2004).

Algorithm 1: Numeric PDB Construction

Input: IRT $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$, pattern P, max. # of states N
Output: PDB heuristic $h_{\text{PDB}}(s)$

```
open.insert(s_0|_P, 0), closed ← ∅, goals ← ∅
while open ≠ ∅ and |closed| + |open| < N do
    s ← open.pop_min()
    closed ← closed ∪ {s}
    if s ⊨ G|_P then
        goals ← goals ∪ {s}
    for each a ∈ A : s ⊨ pre(a)|_P do
        s' ← s⟦a⟧
        s'.parents ← s'.parents ∪ {(s, a)}
        g(s') ← g(s) + cost(a)
        if s' ∉ closed then
            open.insert(s', g(s'))
backward ← ∅
for each s ∈ goals do
    backward.insert(s, 0)
for each s ∈ open do
    h_PDB(s) ← Approximate distance to goal
    backward.insert(s, h_PDB(s))
while backward ≠ ∅ do
    s' ← backward.pop_min()
    for each (s, a) ∈ s'.parents do
        h_PDB(s) ← min(h_PDB(s), h_PDB(s') + cost(a))
        backward.insert(s, h_PDB(s))
```

Definition 1. *Let $\mathcal{T} = \langle S, L, \mathsf{cost}, T, s_0, S_* \rangle$ and $\mathcal{T}' = \langle S', L', \mathsf{cost}', T', s_0', S_*' \rangle$ be two LTSs. We say that the map $\alpha : S \cup L \to S' \cup L'$ is an **LTS homomorphism** if:*

1. *$\alpha(S) \subseteq S'$ and $\alpha(L) \subseteq L'$,*
2. *$\alpha(T) \subseteq T'$, where $\alpha((s, s', l)) = (\alpha(s), \alpha(s'), \alpha(l))$,*
3. *$\mathsf{cost}(\alpha(l)) \leq \mathsf{cost}(l)$ for each $l \in L$, and*
4. *$\alpha(s_0) = s_0'$ and $\alpha(S_*) \subseteq S_*'$.*

To ease notation, we extend the mapping α to sequences $\alpha(\langle x_1, \ldots, x_n \rangle) = \langle \alpha(x_1), \ldots, \alpha(x_n) \rangle$ and sets $\alpha(\{x_1, \ldots, x_n\}) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$. By $\alpha(\mathcal{T})$ we denote the image of $\mathcal{T}$ under α.

The special case of LTS homomorphisms that map the LTS to itself is called an endomorphism and was defined by Horčík and Fišer (2021). A bijective endomorphism is called an automorphism. These morphisms were studied by Shleyfman et al. (2015; 2023) in the context of numeric planning. The methods mentioned deal with state and action pruning.

Since existential-positive formulas are invariant under homomorphisms, they also preserve plans, i. e., the cost of a resulting plan is no greater than the cost of the original plan.

Theorem 1. *For a homomorphism α of an LTS $\mathcal{T}$ and plan π for $\mathcal{T}$, $\alpha(\pi)$ is a plan for $\alpha(\mathcal{T})$ with $\mathsf{cost}(\alpha(\pi)) \leq \mathsf{cost}(\pi)$.*

Proof. Let π be a sequence of transitions that starts at s_0 and ends at some s_*. Note that while $\mathcal{T}$ may be an infinite transition system, π is always finite. Then, π is a path due to Property 2., the initial and goal states go to initial and goal state, respectively, due to Property 4., and the non-increasing cost Property 3. of Definition 1. □

Let π_s^* be an optimal s-plan for $\mathcal{T}$, and let $\pi_{\alpha(s)}^*$ be an optimal $\alpha(s)$-plan for $\alpha(\mathcal{T})$. Then, by Theorem 1 we have that $\mathsf{cost}(\pi_{\alpha(s)}^*) \leq \mathsf{cost}(\pi_s^*)$, and the function $h_\alpha(s) := \mathsf{cost}(\pi_{\alpha(s)}^*)$ is an admissible and consistent heuristic for $\mathcal{T}$.

Orthogonal Projections of Infinite LTSs

We next discuss a method for combining estimates of different PDB heuristics additively. Here, we show that while orthogonal projections are additive in the IRT setting, it is impossible for a numeric variable with goal conditions to appear in two patterns that are combined additively.

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$ be an IRT and let P be a subset of $\mathcal{V}$. For $a \in \mathcal{A}$ and $s \in \mathcal{S}$, the mapping $\alpha(s) := s|_P$ and $\alpha(a) := a$ defines a homomorphism between the transition systems $\mathcal{T}_\Pi = \langle S, \mathcal{A}, \mathsf{cost}, T, s_0, S_* \rangle$ and $\mathcal{T}|_P = \langle S|_P, \mathcal{A}, \mathsf{cost}, \alpha(T), s_0|_P, S_*|_P \rangle$, where the transitions in $\alpha(T)$ are given by the action $\langle \mathsf{pre}(a)|_P, \mathsf{eff}(a)|_P \rangle$, i.e., $\alpha((s, s[\![a]\!], a)) := (s|_P, s|_P[\![a]\!], a)$. Checking that the last map is well-defined proves that α is a homomorphism.

As we want to combine multiple heuristics admissibly, we next look into additivity of multiple patterns. We recall the definition of orthogonal projections and adopt it for IRTs.

Definition 2. *Let $\mathcal{T}$ be an LTS, and let α and β be homomorphisms for $\mathcal{T}$. We say that α and β are **orthogonal** if for every transition $(s, s', l) \in T$ in every plan for $\mathcal{T}$ it holds that $\alpha(s) = \alpha(s')$ or $\beta(s) = \beta(s')$.*

In classical planning orthogonal abstractions yield an admissible additive heuristic estimate for every s that is reachable from s_0. The same result holds for infinite LTSs.

Proposition 1. *Let α and β be orthogonal LTS homomorphisms and let s be reachable from s_0. Then $h_\alpha(s) + h_\beta(s) \leq h^*(s)$.*

Proof. Since s is reachable from s_0, each s-plan can be extended into an s_0-plan. Let $\pi_s = \langle t_1, \ldots, t_n \rangle$ be an optimal s-plan. Then, by orthogonality we have that each transition for $t \in T$ in π_s is a self-loop in $\alpha(\mathcal{T})$ or $\beta(\mathcal{T})$. Thus, there are sub-sequences π_1 and π_2 of π_s s.t. $\alpha(\pi_1)$ is an $\alpha(s)$-plan for $\alpha(\mathcal{T})$, $\beta(\pi_2)$ is an $\beta(s)$-plan for $\beta(\mathcal{T})$, and π_1 and π_2 do not share any transitions. Since $\alpha(\pi_1)$ and $\beta(\pi_2)$ are s-plans, but not necessary optimal plans, we have:

$$h_\alpha(s) + h_\beta(s) \leq \mathsf{cost}(\alpha(\pi_1)) + \mathsf{cost}(\beta(\pi_2)) \leq h^*(s). \quad \square$$

Thus, as in classical planning, one can use additive heuristics with projection patterns that do not have similar non-looping transitions. Unfortunately, there are also challenges. Unlike finite domain variables, there are no loop transitions for actions that affect numeric variables in the pattern.

Proposition 2. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$ be an IRT with a numeric variable $x \in \mathcal{V}_n$ s.t. $s_0[x] \not\models G|_{\{x\}}$, and let α and β be two projections on patterns P and P', respectively, s.t. $x \in P \cap P'$. Then, α and β are not orthogonal.*

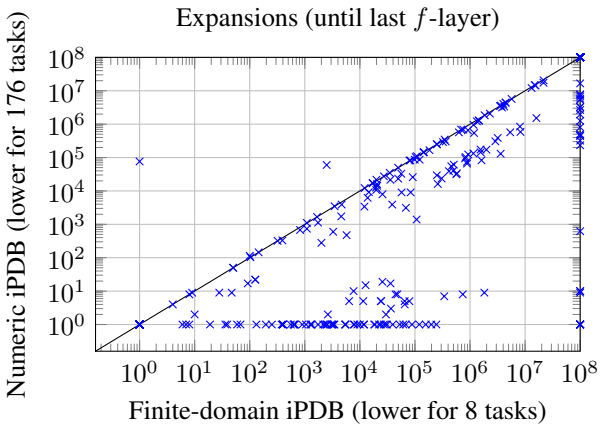

Figure 1: Per-instance comparison of the number of expanded states of finite-domain vs. numeric iPDB.

Proof. Let π be a plan for Π. Since $s_0[x] \not\models G\big|_{\{x\}}$ there must be an action a along π that changes the value of x. Assume this action corresponds to the transition (s, s', a) where $s[x] \neq s'[x]$. Thus, $s\big|_P \neq s'\big|_P$ and $s\big|_{P'} \neq s'\big|_{P'}$. Hence, both transitions $(\alpha(s), \alpha(s'), a)$ and $(\beta(s), \beta(s'), a)$ are both not loops, i.e., α and β are not orthogonal. □

Bounding Infinite LTSs

We conclude this section by presenting an algorithm to partially generate an infinite LTS in order to obtain an admissible heuristic. Note that the proposed heuristic will be informative only on a subset of the original states.

Let $\mathcal{T}$ be an LTS with the initial state s_0. Let us look at the states that are reachable from s_0. Consider a sub-transition system $\mathcal{T}'$ over the states $S' = S_E \cup S_F$, s.t.

1. $T' \subseteq T$ and $S'_* \subseteq S_*$,
2. all states in $S_E \cup S_F$ are reachable from s_0,
3. if $s \in S_E$ and $(s, s', l) \in T$ then $(s, s', l) \in T'$, i.e., all successors of s are in $\mathcal{T}'$,
4. if s is in S_F, then none of its successors are in $\mathcal{T}'$, i.e., s has no outgoing edges in $\mathcal{T}'$, and lastly
5. if $s_* \in S'_*$ then $s_* \in S_F$.

Note that $\mathcal{T}'$ corresponds to the Best First Search (*BFS*) paradigm, where we gradually traverse the LTS by expanding nodes one by one by taking candidates from a fringe. In our case, S_E corresponds to the expanded nodes and S_F to the open nodes in the fringe. Unlike a regular *BFS*, instead of a search tree, we keep the full graph structure. We also note that there is no point in expanding the goal states, since we are interested in the shortest path to the goal. The pseudocode for computing a PDB heuristic h_{PDB} is provided in Algorithm 1. There, S_E corresponds to the states in open $\cup$ goals and S_F corresponds to the states in closed.

For this finite sub-LTS $\mathcal{T}'$, we define the following heuristic for the states in $S_E \cup S_F$:

$$h_{\text{PDB}}(s) = \min_{s' \in S_F} \{\mathsf{cost}^*(s, s') + d(s')\},$$

where $d(s') = 0$ if s' is a goal state and the minimal cost of any action applicable in s' otherwise.

Proposition 3. *h_{PDB} is admissible on the states in $S_E \cup S_F$.*

Proof. Note that no state in $S_F \cup S_E$ can be reached from $s \in S_F$, so $h_{\text{PDB}}(s)$ is admissible by the definition of $d(s)$ for all $s \in S_F$. It remains to show admissibility for $s \in S_E$.

Let π^*_s be an optimal s-plan. Since, by definition of a sub-LTS $\mathcal{T}'$, for each $s \in S_E$, along each s-plan π there is at least one state s' such that $s' \in S_F$. In the LTS $\mathcal{T}'$ we are always looking at the first such state, since states in S_F have no outgoing edges. Let s^*_F be such a state for π^*_s. Then,

$$\begin{aligned} h^*(s) &= \mathsf{cost}^*(s, s^*_F) + h^*(s^*_F) \\ &\geq \min_{s' \in S_F} \{\mathsf{cost}^*(s, s') + h(s')\} = h_{\text{PDB}}(s). \end{aligned}$$ □

For a pattern $P \subseteq \mathcal{V}$, the projection of Π onto P is a valid planning task, thus using the heuristic $h_{\text{PDB}}(s)$ we can construct a PDB for a bounded number of states.

We proposed the first adaptation of PDB heuristics for SNP, addressing the unbounded nature of numeric variables. We now explore adapting pattern generation for IRT.

Pattern Generation for Numeric Variables

The success of PDB heuristics in classical planning relies on *generating* multiple patterns to capture different aspects of a task and *combining* several PDB heuristics admissibly. These components are tightly connected and often studied together, see Haslum (2007) and Pommerening et al. (2013).

A strong method for both components, called iPDB, uses hill-climbing to optimize heuristic quality on sampled states, creating a pattern collection evaluated during the process (Haslum et al. 2007). The hill-climbing extends patterns by adding variables, improving the pattern collection over time. Another approach, *systematic pattern generation*, enumerates all patterns with k variables, focusing on *interesting* patterns that include goal and causally related variables (Pommerening, Röger, and Helmert 2013). Both methods combine patterns with the *canonical PDB heuristic*, identifying additive patterns by checking action dependencies.

We adapt pattern generation methods to handle numeric variables. As with finite-domain variables, we start with goal variables and use the numeric CG to identify related variables, constructing systematic patterns without additional reasoning. For iPDBs, we guide the hill-climbing process to prevent overly large abstract state spaces.

The base iPDB variant bounds the pattern size by the domain-size product of variables, which does not apply to numeric variables. Instead, we approximate the *relevant range of values* for each variable using explicit numeric intervals $[M^-_x, M^+_x]$ (ENI) and maximum additive constants C^+_x and C^-_x (Shleyfman, Gnad, and Jonsson 2023). We estimate reachable values in $[M^-_x - C^-_x, M^+_x + C^+_x]$ by dividing the range by the GCD of the effects on x. While not a sound bound, this provides a reasonable approximation.

Experimental Evaluation

Our planner is based on the Numeric Fast Downward framework (NFD) (Aldinger and Nebel 2017; Kuroiwa et al.

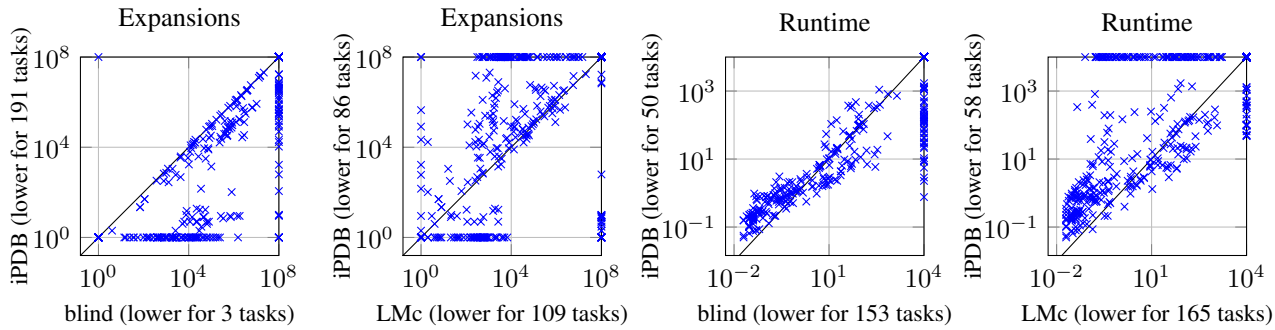

Figure 2: Per-instance comparison of the search-space size (number of state expansions until last f-layer) and runtime of blind search and A* with LM-cut vs. numeric iPDB. Points below the diagonal favor our approach.

2022). We adapted the implementation of pattern-database heuristics from Fast Downward (Helmert 2006) to support numeric variables. All experiment data and code is available online (Gnad et al. 2024), and our code is maintained in a public repository.[1] The experiments were conducted on a cluster of Intel Xeon Gold 6130 CPUs using Downward Lab 8.2 (Seipp et al. 2017), adopting the runtime and memory limits of 30 min and 8GiB from the recent International Planning Competition (IPC) 2023 (Taitler et al. 2024). We also adopt the simple numeric benchmark set from IPC'23. We extend it with existing benchmarks from the literature (Scala, Haslum, and Thiébaux 2016; Scala et al. 2017, 2020; Shleyfman, Kuroiwa, and Beck 2023) and introduce two benchmark sets ourselves, forestfire and minecraft. For minecraft we generated 20 so-called "advanced" instances for each of the two variants of the domain, crafting a pogostick, respectively a sword (Benyamin et al. 2024).

We compare our approach against the interval-relaxed $h^{\max}$ heuristic (**imax**) (Aldinger and Nebel 2017), the repetition $h^{\max}$ heuristic (**rmax**) (Scala et al. 2020), the numeric landmark heuristic (**LM**) (Scala et al. 2017), and the winning heuristics from IPC'23's numeric track, numeric LM-cut (**LMc**) (Kuroiwa et al. 2022) and numeric operator counting (**OPC**) (Kuroiwa et al. 2021). For consistency, orbit-space search (Shleyfman, Kuroiwa, and Beck 2023), orthogonal to heuristics, is disabled in the IPC planner configurations. Results for rmax and LM use NFD reimplementations, shown to closely match the original ENHSP versions (Kuroiwa et al. 2021). All configurations use NFD with A*.

For the numeric PDB heuristics, we translate the simple numeric PDDL tasks using NFD's translator component as usual, and then transform them to restricted tasks (RT) for the heuristics. We impose a limit on the number of abstract states that are generated during the progression phase for numeric PDBs. This limit has a large impact both on the accuracy of the resulting heuristic and the computational cost of constructing the abstract state space. We empirically determined these limits for each pattern generation method.

We evaluate a single-PDB heuristic with the greedy pattern generation of FD as a baseline (**greedy**), the canonical PDB heuristic over pattern collections generated with systematic patterns (**sysC**), and **iPDB**, the hill-climbing approach of Haslum et al. (2007). For the greedy PDB, we limit the number of abstract states to 100k. For the systematic pattern generation, we compute all interesting patterns with two variables, which performed best in our evaluation, and limit the number of abstract states by 50k. iPDB is the state-of-the-art method for finite-domain pattern generation when combining patterns with the canonical heuristic. We limit the collection size (sum of individual pattern sizes) of iPDB by 10 million abstract states, the hill-climbing time by 15min, and initially sample 1000 states to estimate improvement, similar to the finite-domain variant of iPDB. We limit the number of abstract states to 10k and use 1 million as an upper bound on the domain-size product of variables allowed in a pattern. The latter is used by iPDB as a mechanism to obtain many small patterns. State sampling is done using NFD's default random seed.

As a first sanity check, we compare our numeric PDBs with finite-domain PDBs that ignore all numeric variables. Figure 1 illustrates the heuristic accuracy by showing the search-space size (number of states expanded until the last f-layer in A*) on a per-instance basis. We compare finite-domain iPDB (x-axis) to numeric iPDB (y-axis). It is clearly visible that taking numeric variables into account has a huge impact on the search effort, leading to a reduction of up to six orders of magnitude in terms of state expansions.

Figure 2 sheds more light onto the differences between blind search, respectively LMc, and numeric iPDB. The two leftmost plots show the search-space size, the two rightmost plots show overall runtime. Compared to blind search, we observe that numeric PDBs are indeed an informed heuristic that provides a strong guidance to the A* search. The LM-cut heuristic shows complementary strengths to iPDB, with many instances ending up on both sides of the diagonal. Overall, though, there are more instances in which LMc offers better search guidance. In the runtime plots, we observe the common pattern for abstraction heuristics, where the initial computation of the heuristic takes some time to complete. After that, iPDB is often faster than blind search and shows complementarity compared to LMc.

Table 1 presents coverage results (number of solved instances) for all planners, summarizing domains where all planners perform equally under "others". Numeric PDBs

[1] See link on first page.

	Domain	#	blind	imax	rmax	LM	OPC	LMc	FDR iPDB	Numeric PDBs greedy	Numeric PDBs sysC	Numeric PDBs iPDB	Time Score LMc	Time Score iPDB	% Missed greedy	% Missed iPDB
IPC 2023	delivery	20	2	2	2	2	2	**3**	2	2	2	2	**1.73**	1.55	0	0
	drone	20	3	2	3	3	3	3	3	**4**	3	**4**	3.00	**3.38**	0	0
	expedition	20	5	0	**6**	5	5	**6**	**6**	**6**	**6**	**6**	4.02	**4.82**	0	0
	farmland	20	4	4	**15**	**15**	**15**	**15**	4	4	4	4	**12.32**	3.20	26	0
	hydropower	20	9	8	**11**	8	9	**11**	9	9	9	9	4.38	**6.75**	24	66
	mprime	20	6	10	10	5	14	**15**	12	11	11	12	**11.28**	8.04	0	0
	rover	20	**4**	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	3.53	**3.56**	51	0
	sailing	20	0	0	5	4	**9**	8	0	0	1	0	**3.84**	0	-	-
	sugar	20	2	0	2	1	**12**	**12**	2	4	3	3	**5.03**	1.67	0	0
	zenotravel	20	6	6	6	6	**8**	**8**	6	6	6	6	**6.43**	5.43	0	0
from literature	counters	20	3	3	4	**5**	**5**	**5**	3	3	**5**	**5**	**4.70**	3.82	0	0
	counters-sym	11	2	2	3	10	**11**	**11**	2	2	5	9	**7.71**	5.51	0	0
	depots	20	4	5	5	3	**7**	**7**	**7**	6	**7**	**7**	4.70	**4.96**	0	0
	depots-sym	20	4	5	4	2	6	**7**	6	6	5	6	**3.98**	3.92	0	0
	farmland	30	12	11	**30**	**30**	**30**	**30**	12	12	12	12	**23.92**	10.53	24	0
	fn-counters-small	8	6	**7**	**7**	**7**	**7**	**7**	6	6	**7**	**7**	**6.42**	6.39	30	0
	petri-net	20	2	2	6	4	8	**9**	2	3	7	**9**	2.26	**2.98**	72	0
	plant-watering	63	**63**	21	**63**	**63**	**63**	**63**	**63**	**63**	**63**	**63**	60.72	**60.74**	0	0
	rover-unit	20	4	2	4	4	**7**	**7**	6	4	5	6	**5.37**	4.87	32	0
	sailing	40	10	12	28	20	**40**	**40**	11	11	17	14	**37.63**	9.63	23	2
	satellite	20	1	0	1	1	**2**	**2**	**2**	1	1	**2**	1.55	**1.77**	0	0
	zenotravel	23	6	7	7	7	**12**	**12**	9	7	10	11	**9.83**	6.85	4	0
novel	forestfire	20	10	0	10	6	9	**11**	10	10	10	10	6.81	**7.81**	0	0
	minecraft-pogo	20	14	9	0	0	5	5	14	**19**	17	18	2.11	**4.43**	0	0
	minecraft-sword	20	**20**	17	0	0	9	9	**20**	**20**	**20**	**20**	4.93	**10.77**	0	0
	others	80	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1.00**	0.91	0	0
	$\sum$	635	203	136	237	216	303	**311**	222	224	241	250	**239.2**	184.3	11	3

Table 1: Coverage results of the baseline planners (columns "blind"–"FDR iPDB") and a single numeric PDB (greedy), numeric canonical PDBs with systematic patterns (sysC), and numeric iPDB. We refer to the text for a detailed explanation.

significantly outperform the baseline $h^{\max}$ and landmark heuristics, but fall short of OPC and LMc by 53 and 61 instances, respectively. This gap is largely due to the farmland and sailing domains, where OPC and LMc solve 63 more instances combined. Conversely, iPDB excels in the two minecraft variants, solving 24 additional instances. Finite-domain iPDB shows similar performance to single-pattern numeric PDBs but lags behind the numeric variant overall.

The "Time Score" columns provide per-domain runtime statistics, calculated as $1 - \log(t)/\log(1800)$, where t is the solution time in seconds. iPDB remains competitive with LMc in several domains, reflecting LMc's slower runtime.

Lastly, the final two columns report how often greedy and iPDB planners evaluate states where no abstract state was generated (lookup misses). For iPDB, this only includes cases where all PDBs in the collection missed. The average percentage of lookup misses across solved instances is significantly lower for iPDB, indicating the benefit of using multiple smaller PDBs to cover different parts of the task.

Overall, our evaluation shows that numeric PDB heuristics perform well for optimal numeric planning, even though the evaluated variants using the canonical heuristic cannot compete with LMc and OPC in terms of total coverage.

Conclusion

We introduce the first adaptation of pattern-database (PDB) heuristics, which are among the state of the art in optimal classical planning, to tasks with numeric variables. A major obstacle is the possibly unbounded abstract state space, for which we propose several solutions. Our empirical evaluation confirms that these solutions work well on common numeric planning benchmarks and bring numeric PDBs in reach of the strongest admissible heuristics in simple numeric planning. There are many possibilities to further improve the results, such as computing lower bounds for unexpanded abstract states using other heuristics. We leave the investigation of these optimizations for future work.

More generally, there is the question of how to combine multiple abstraction heuristics for numeric planning efficiently in a better way using cost partitioning. Another interesting direction is the application of PDBs and merge-and-shrink heuristics tailored to detecting unsolvability in classical planning (Hoffmann, Kissmann, and Torralba 2014) to its numeric counterpart. Finally, the approach of constructing finite sub-transition systems for infinite projections can be naturally combined with strong numeric heuristics such as LM-cut (Kuroiwa et al. 2022; Kuroiwa, Shleyfman, and Beck 2022). We want to investigate how different heuristics interact and can be made compatible to each other.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. Lee-or Alon is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship. Eyal Weiss's work was supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities and by Bar-Ilan University's President Scholarship. Alexander Shleyfman's work was partially supported by ISF grant 2443/23.

References

Aldinger, J.; and Nebel, B. 2017. Interval Based Relaxation Heuristics for Numeric Planning with Action Costs. In *the Annual German Conference on Artificial Intelligence (KI)*, 15–28. Springer.

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial Pattern Databases. In *the International Symposium on Abstraction, Reformulation, and Approximation (SARA)*, 20–34.

Bacchus, F.; and Yang, Q. 1994. Downward Refinement and the Efficiency of Hierarchical Problem Solving. *Artificial intelligence (AIJ)*, 71(1): 43–100.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11(4): 625–655.

Benyamin, Y.; Mordoch, A.; Shperberg, S.; Piotrowski, W.; and Stern, R. 2024. Crafting a Pogo Stick in Minecraft with Heuristic Search. In *the International Symposium on Combinatorial Search (SoCS)*, volume 17, 261–262.

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

Brafman, R. I.; and Domshlak, C. 2003. Structure and Complexity in Planning with Unary Operators. *the Journal of Artificial Intelligence Research (JAIR)*, 18: 315–349.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2013. A Hybrid LP-RPG Heuristic for Modelling Numeric Resource Flows in Planning. *the Journal of Artificial Intelligence Research (JAIR)*, 46: 343–412.

Culberson, J. C.; and Schaeffer, J. 1996. Searching with Pattern Databases. In *the Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI-96)*, volume 1081 of *LNAI*, 402–416.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence*, 14(3): 318–334.

Edelkamp, S. 2002. Symbolic Pattern Databases in Heuristic Search Planning. In *the International Conference on AI Planning and Scheduling (AIPS)*, 274–283.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 130–137.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive Pattern Database Heuristics. *the Journal of Artificial Intelligence Research (JAIR)*, 22: 279–318.

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

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial intelligence (AIJ)*, 172(8-9): 899–944.

Gigante, N.; and Scala, E. 2023. On the compilability of bounded numeric planning. In *the International Joint Conferences on Artificial Intelligence (IJCAI)*, volume 23, 5341–5349.

Gnad, D.; Alon, L.; Weiss, E.; and Shleyfman, A. 2024. Code and experiment data of the AAAI 2025 paper "PDBs Go Numeric: Pattern-Database Heuristics for Simple Numeric Planning". https://doi.org/10.5281/zenodo.14502394.

Gnad, D.; Helmert, M.; Jonsson, P.; and Shleyfman, A. 2023. Planning over Integers: Compilations and Undecidability. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 148–152.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *the Association for the Advancement of Artificial Intelligence (AAAI)*, 1163–1168.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *the Association for the Advancement of Artificial Intelligence (AAAI)*, 1007–1012.

Helal, H.; and Lakemeyer, G. 2024. An Analysis of the Decidability and Complexity of Numeric Additive Planning. In *the International Conference on Automated Planning and Scheduling (ICAPS)*.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *the International Conference on AI Planning and Scheduling (AIPS)*, 303–312.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 161–170.

Helmert, M. 2006. The Fast Downward Planning System. *the Journal of Artificial Intelligence Research (JAIR)*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial intelligence (AIJ)*, 173: 503–535.

Helmert, M.; and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 19, 162–169.

Hoffmann, J. 2003a. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *the Journal of Artificial Intelligence Research (JAIR)*, 20: 291–341.

Hoffmann, J. 2003b. *Utilizing Problem Structure in Planning, A Local Search Approach*, volume 2854 of *LNCS*. Springer.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In *the European Conference on Artificial Intelligence (ECAI)*, volume 263, 441–446.

Holte, R.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over Multiple Pattern Databases Speeds up Heuristic Search. *Artificial intelligence (AIJ)*, 170(16–17): 1123–1136.

Holte, R.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple Pattern Databases. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 122–131.

Horčík, R.; and Fišer, D. 2021. Endomorphisms of Classical Planning Tasks. In *the Association for the Advancement of Artificial Intelligence (AAAI)*, 11835–11843.

Illanes, L.; and McIlraith, S. A. 2017. Numeric Planning via Abstraction and Policy Guided Search. In *the International Joint Conferences on Artificial Intelligence (IJCAI)*, 4338–4345.

Katz, M.; and Domshlak, C. 2009. Structural-Pattern Databases. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 186–193.

Knoblock, C. A. 1994. Automatically Generating Abstractions for Planning. *Artificial intelligence (AIJ)*, 68(2): 243–302.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-Cut Heuristics for Optimal Linear Numeric Planning. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 203–212.

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 *the International Conference on Automated Planning and Scheduling (ICAPS)*, 210–218.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2022. The LM-Cut Heuristic Family for Optimal Numeric Planning with Simple Conditions. *the Journal of Artificial Intelligence Research (JAIR)*, 75: 1477–1548.

Li, D.; Scala, E.; Haslum, P.; and Bogomolov, S. 2018. Effect-Abstraction Based Relaxation for Linear Numeric Planning. In *the International Joint Conferences on Artificial Intelligence (IJCAI)*, 4787–4793.

Libkin, L. 2004. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Piacentini, C.; Castro, M. P.; Ciré, A. A.; and Beck, J. C. 2018a. Compiling Optimal Numeric Planning to Mixed Integer Linear Programming. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 383–387.

Piacentini, C.; Castro, M. P.; Ciré, A. A.; and Beck, J. C. 2018b. Linear and Integer Programming-Based Heuristics for Cost-Optimal Numeric Planning. In *the Association for the Advancement of Artificial Intelligence (AAAI)*, 6254–6261.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *the International Joint Conferences on Artificial Intelligence (IJCAI)*, 2357–2364.

Scala, E.; Haslum, P.; Magazzeni, D.; and Thiébaux, S. 2017. Landmarks for Numeric Planning Problems. In *the International Joint Conferences on Artificial Intelligence (IJCAI)*, 4384–4390.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *the International Joint Conferences on Artificial Intelligence (IJCAI)*, 3228–3234.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *the Journal of Artificial Intelligence Research (JAIR)*, 68: 691–752.

Scala, E.; Ramírez, M.; Haslum, P.; and Thiébaux, S. 2016. Numeric Planning with Disjunctive Global Constraints via SMT. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 276–284.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Shin, J.; and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial intelligence (AIJ)*, 166(1-2): 194–253.

Shleyfman, A.; Gnad, D.; and Jonsson, P. 2023. Structurally Restricted Fragments of Numeric Planning–a Complexity Analysis. In *the Association for the Advancement of Artificial Intelligence (AAAI)*, 10, 12112–12119.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and Symmetries in Classical Planning. In *the Association for the Advancement of Artificial Intelligence (AAAI)*, 3371–3377.

Shleyfman, A.; Kuroiwa, R.; and Beck, J. C. 2023. Symmetry Detection and Breaking in Linear Cost-Optimal Numeric Planning. In Koenig, S.; Stern, R.; and Vallati, M., eds., *the International Conference on Automated Planning and Scheduling (ICAPS)*, 393–401.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient Implementation of Pattern Database Heuristics for Classical Planning. In *the International Symposium on Combinatorial Search (SoCS)*, 49–56.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fiser, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Mag.*, 45(2): 280–296.