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

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

[1]Linköping University, Linköping, Sweden
[2]Bar-Ilan University, Ramat Gan, 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. This approach aims to restrict the initially infinite projections, thereby bounding the number of states and ultimately constraining the resulting PDBs. We argue that our PDB heuristics obtained with our approach can provide strong guidance for the search.

Introduction

In this work, we concentrate on simple numeric planning (SNP) with instantaneous actions. Numeric state fluents introduce a further degree of complexity over classical planning, making plan existence 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 with so-called *simple* numeric conditions, where numeric variables can be increased or decreased by constant quantities and pre-conditions are inequalities involving linear expressions.

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 does so in a perfect precision. 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 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.

Unfortunately, unlike in classical planning, the transition systems induced by a projection of a task onto a set of variables that include numeric fluents may have infinitely many states. Therefore, to adopt the multiple approaches used in classical planning to generate informative PDBs, one needs to somehow bound the numeric transition system. We introduce several strategies that can be employed to do so, bounding the numeric transition system effectively. One approach is to discretize the numeric fluents within the transition system, reducing then the infinite state space to an incomplete finite one. Another approach involves defining 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. In what follows, we discuss this approaches, providing them with a theoretical background.

Preliminaries

Numeric Planning

We work over a fragment of numeric planning based on the finite-domain planning (FDR) formalism (Bäckström and Nebel 1995; Helmert 2009) extended with numeric fluents 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 shows that RTs are equivalent to simple numeric tasks (Hoffmann 2003a), and Helmert (2002) demonstrates that any RT can be transformed into an IRT. For the expres-

sive power of IRTs see (Gnad et al. 2023).

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 the task, 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 equivalently $\langle v, d \rangle \in s$ 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 of partial states $\langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$ forms an action $a \in \mathcal{A}$, where $\mathsf{pre}(a)$ is the *precondition* and $\mathsf{eff}(a)$ is the *effect* of a. *Precondition* $\mathsf{pre}(a) := \mathsf{pre}_p(a) \cup \mathsf{pre}_n(a)$ is formed out of propositional and numeric conditions, respectively. Similarly, *effects* $\mathsf{eff}(a) := \mathsf{eff}_p(a) \cup \mathsf{eff}_n(a)$ are given as sets 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 given by $s[\![a]\!] := s'_p \cup s'_n$, 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. In this work we assume that each action has at most one effect on each variable. $\mathsf{cost}(a)$ denotes the cost of an action.

The goal description, denoted as 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., $s_* \models G$. A *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 a 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 a s-plan is denoted by $h^*(s)$. If no goal state is reachable from s we say $h^*(s) = \infty$. A plan for Π is a s_0-plan, and its cost is $h^*(s_0)$.

PDBs for Classical Planning

One of the most employed algorithms to solve optimally a 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, an abstraction of the state space is created. A projection Π_P 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. In the perfect heuristic, the distances from each state to the goal are optimal. This projection can be computed by eliminating all occurrences of variables from $\mathcal{V} \setminus P$ in Π. PDBs are 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 $\Pi\big|_P$.

Instead of starting from the initial abstract state and compute the distance of each abstract state to the set of abstract goal states, as done in progression search, regression search is often used for this purpose. A regression search starts from the set of abstract goal states and searches backwards until reaching the initial abstract state. Unlike progression search which may require multiple iterations over some abstract states, regression search avoids these redundant distance computations. By exploring the abstract state space in a backward direction, it computes state distances from goal states in a single exploration. This could also detect deadends along the way, as the search progresses from the set of abstract goals towards the initial abstract state.

During the search process, concrete states s are projected onto the variables in P using a perfect hash function (Sievers, Ortlieb, and Helmert 2012). This hash function is well-defined for all partial states s^{pt} defined over the variables $Q \supseteq P$. We use the notation $\Pi\big|_P(s^{pt})$ for the heuristic computation of such states. Given that PDBs grow exponentially in the number of included variables, single PDB heuristics alone typically do not offer sufficient guidance. Consequently, state-of-the-art planners employ various techniques to admissibly combine multiple PDB heuristics.

Combining Multiple PDBs

Let H be a set of admissible heuristics. For any state s a maximum over this set of heuristics is an admissible estimate. We say that H is *additive* if $\sum_{h \in H} h(s)$ is admissible. A tighter bound for a set of PDB heuristics H was proposed by Haslum et al. (2007) who introduced a 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., $h^C = \max_{A \in \mathcal{A}(C)} \sum_{P \in A} h\big|_P$, where $\mathcal{A}(C)$ is a set of maximal disjoint-additive subsets of C. h^C is admissible.

A general combination technique for a set of arbitrary admissible non-additive heuristics $\{h_i\}_{i=1}^n$ is cost partitioning (Katz and Domshlak 2010; Pommerening et al. 2015). Cost partitioning computes each heuristic h_i for the task Π under a cost function cost_i different from the original cost. The sum $\sum_{i=1}^n h_i$ is admissible if $\sum_{i=1}^n \mathsf{cost}_i(a) \leq \mathsf{cost}(a)$ for all actions $a \in \mathcal{A}$. Unfortunately, computing the optimal cost partitioning is usually infeasible in practice. State-of-the-art approaches to approximate the optimal cost partition-

ing are for example saturated cost partitioning (SCP) (Seipp, Keller, and Helmert 2020) and post-hoc optimization (PhO) (Pommerening, Röger, and Helmert 2013).

In a nutshell, SCP considers the heuristics in $\{h_i\}_{i=1}^n$ in an arbitrary but fixed order. When computing h_i, it computes the saturated costs scf_i, which are the minimum costs needed by the heuristic computation. They can be computed efficiently for abstraction heuristics by looping over all abstract transitions of the abstract state space. The costs not needed by h_i, $\mathsf{cost}_i - \text{scf}_i$, are the costs cost_{i+1} available to the next heuristic h_{i+1} and so on. The order of the heuristics in this iterative approach may affect the heuristic estimates.

In post-hoc optimization, the algorithm identifies which actions are relevant for each abstraction-based heuristic. That is, determines which action affects variables in each pattern. For each action that affects an abstract heuristic, the algorithm assigns a value called the action's *factor*, which determines the effect of the action on the heuristic in the projection onto that pattern. The cost partitioning is then calculated by the sum of each action's factor with the action's cost. Thus, the algorithm combines several PDBs considering the effect of actions within each heuristic.

Cost partitioning is a general approach that combines heuristics while maintaining an admissible bound. PDB, on the other hand, is a specific abstraction heuristic. Previous works have emerged cost partitioning to combine several PDB heuristics to improve the estimates in classical planning (Seipp, Keller, and Helmert 2020; Pommerening et al. 2015). In cost partitioning projections, suboptimal cost partitions are computed, and are then composed to a single (optimal) partitioning.

Causal Graphs for Numeric Planning

Since planning tasks are usually structurally complex, multiple means were introduced to study this structure. One of the them is the *causal graphs* (CG) that were introduced for classical planning (e.g., Knoblock 1994; Bacchus and Yang 1994; Brafman and Domshlak 2003). Recently, Shleyfman et al. (2023) adopted the compact definition of CG by Helmert (2004) for RTs. The main observation made in this adaptation was that in RTs all conditions and effects, numeric or propositional, involve exactly one variable per formula. Denote by $vars(\Psi)$ be the function that returns the set of variables that are involved in a a set of formulas Ψ. For each individual RT formula $\psi \in \Psi$ the function $vars$ returns a single variable. Thus, the definition of CG for classical tasks can be adopted to RTs almost as-is.

The CG of a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$, then, is a digraph $CG(\Pi) = \langle \mathcal{V}, \mathcal{E} \rangle$, where $(u, v) \in \mathcal{E}$ if $u \neq v$ and there exists $a \in \mathcal{A}$, s.t. $u \in vars(\mathsf{pre}(a)) \cup vars(\mathsf{eff}(a))$ and $v \in vars(\mathsf{eff}(a))$. In essence, in a causal graph (CG), 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.

One of the most commonly used CG structure were introduced by for classical planning by Domshlak and Dinitz (2001) are *forks* and *inverted forks*. Recall that in digraphs, *roots* are vertices that have only outgoing edges, and *leaves* are the vertices that have only in incoming edges. A CG, then, is a *fork* if there is a $y \in \mathcal{V}$ such that for any other variable x the CG contains an edge (y, x), and there are no other edges. A CG is an *inverted fork* if (x, y) instead.

PDBs Heuristics for Integer Variables

Common techniques employed to compute PDBs for classical planning involve computing the entire transition system (TS) 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 TS. In this section we introduce several approaches that are capable of dealing with potentially infinite TS. We start by defining infinite TS and discussing the challenges involved in having PDBs with infinite state spaces.

Infinite Transition Systems

Let $\mathcal{T} = \langle S, L, \mathsf{cost}, T, s_0, S_* \rangle$ be a directed weighted labeled graph called a *transition system* (TS), 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 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 a TS, which we denote by $\mathcal{T}_\Pi$. The main difference between classical and numeric planning is that in the former the number of states in the induced TS is finite, while in the latter it often occurs that $|S| = \infty$.

A path from $s \in S$ to $s' \in S$, denoted $\text{path}(s, s')$ 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 $\mathsf{cost}(s, s') := \mathsf{cost}(\text{path}(s, s')) = \sum_{i=1}^n \mathsf{cost}(l_i)$, the cost of an empty path is zero. $\mathsf{cost}^*(s, s')$ is the minimal cost of a path from s to s'. A path from some state s to some goal state $s_* \in S_*$ is also called an s-plan, an s_0-plan is called a plan. A plan is *optimal* if there exists no other plan with lower cost.

How to Deal with Numeric Variables?

The first question one may ask is what kind of numbers can actually be supported in PDB heuristics. In this work, we opt for integers, which still covers a significant fragment of numeric planning, as shown in previous work (Helmert 2002; Hoffmann 2003a; Gnad et al. 2023). Note that this is not a strict requirement, though. As long as only a finite set of distinct values needs to be distinguished for every numeric variable, this is supported by our approach.

A more serious limitation is in the representation of action effects that can be dealt with effectively. In the IRT formalism, numeric effects are restricted to addition by a constant. If we allowed for, e. g. assignment effects of other variables values, such as $x := y$, then if x is contained in a pattern P, but y is not, that would cause an infinite branching in every abstract state in which the respective action is applicable. The same is true whenever the new value of a numeric variable in the pattern depends on a numeric variable outside the pattern. In order to avoid infinite branching, we require that

such effects are compiled away in a preprocessing step that results in an IRT task.

The common approach to compute a PDB heuristic is to construct the *abstract state space*, i. e., the TS induced by a pattern P, and compute minimal goal distances for every abstract state. To avoid duplicate computation, the abstract state spaces are constructed starting from the set of abstract goal states, in a regression search. This enables the construction of the TS and the computation of abstract goal distances in a single pass. For patterns containing numeric non-goal variables, this is not possible, though. The reason is that there are infinitely many abstract goal states, so a regression is not possible. We address this issue by performing a progression search instead, starting from the (fully specified) abstract initial state $s_0[P]$. We explore the abstract state space either until exhaustion, in case it is bounded by the task structure, or until a maximum number of states N has been generated. On exhaustion, we can compute the minimum goal distances for all generated states by a single regression pass from the goal states. If we hit the limit N, we stop the construction. For abstract states that have not been expanded at that point, i. e., states in the fringe, we do not have any information about their goal distance. A variant we discuss below is to compute such distance using another heuristic. Our default algorithm simply assumes a goal distance equal to the minimum action cost. With that, as before, we can invoke a regression starting in the goal and fringe states, computing the goal distances.

Once the (partial) abstract state space is constructed and we computed the goal distances, we can build a look-up table using perfect hashing very similarly to how this is done in classical planning. Observe that for every numeric variable only a finite number of distinct values can be reached during the state-space construction. Hence we can view these numeric variables as finite-domain variables, and compute the hashes as before.

A downside of partially constructing the abstract state, when the upper bound N on the number of states is hit, is that we can visit a concrete state s in the main search without abstract counterpart $s|_P$ in the PDB. That is because an insufficient part of the domain of some numeric variables $x \in P \cap \mathcal{V}_n$ has been generated during PDB construction. We suggest two possible solutions to this: (1) ignore this PDB, i. e., use a heuristic value of 0, or (2) update the heuristic by invoking another progression from $s|_P$ and merging the look-up tables. For (1), since most commonly large collections of PDB heuristics are used, we expect that some other PDB heuristic will cover state s. Option (2) takes a more practical approach, refining the heuristic during the search, as larger parts of the search space are generated that have not initially been covered by a PDB.

Next, we describe two approaches that are more clever in constructing the initial abstract state space of a PDB than the blind variant that we just outlined.

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 interested for our work are results that impose conditions on the variables dependencies of a planning task, e. g., by restricting the structure of the causal graph. If we select the pattern $P \subseteq \mathcal{V}$ such that the projection $\Pi|_P$ 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) even provide a recipe to compute the bounded domain intervals, which we use to estimate the size of a PDB.

Care must be taken with this approach, though, because ignoring variable dependencies outside P is an approximation. If a numeric variable $v_n \in P$ is in a cyclic dependency with another one not in P, then in the full task the bounds that were computed for numeric variables within the projection are not necessarily valid. Furthermore, even if the bounds are in fact valid on Π, we can still run into the issue described in the previous section, where concrete states reached in the search may have numeric values outside the bound. That is because in particular the causal-graph related analyses imply that it is *sufficient* to consider numeric values only within the bounds, but not that values outside the bound are not reachable.

We can address this problem in the same way as before, either by ignoring such PDB heuristics for states with out-of-bound values, or by re-invoking the abstract state space construction from the previously unseen abstract state.

Bounding Infinite TS with an Admissible Heuristic

Let $\mathcal{T}$ be a TS with the initial state s_0. Let us look at the states that are reachable from s_0. Consider a sub-transition system $\mathcal{T}' = \langle S', L, \mathsf{cost}, T', s_0, S'_* \rangle$ over the states $S' = S_E \cup S_F$, such that

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 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 TS by expanding nodes one by one by taking candidates from the fringe. In our case, S_E corresponds to the expanded nodes and S_F to the expanded nodes in the fringe. Unlike a regular *BFS* instead of a search tree, we keep the whole graph structure. We also note that there is no point in expanded the goal states, since we are interested in the shortest path to the goal.

For this finite sub-TS $\mathcal{T}'$ and a heuristic h we define the following heuristic:

$$\tilde{h}(s) = \begin{cases} \min_{s' \in S_F} \{\mathsf{cost}^*(s, s') + h(s')\} & \text{if } s \in S_E \\ h(s) & \text{otherwise.} \end{cases}$$

Note that if a goal state s_* is in S_F we have $\tilde{h}(s_*) = 0$.

Proposition 1. *If h is admissible so is $\tilde{h}$. If in addition h is consistent, then so is $\tilde{h}$ and $h \leq \tilde{h}$.*

Proof. Note that for each $s \notin S_E$ we have $h(s) \leq \tilde{h}(s)$. Thus, we are interested only in the proof for $s \in S_E$.
Admissibility: Let π_s^* be an optimal s-plan. Since, by definition of sub-TS $\mathcal{T}'$, for each $s \in S_E$, along each s-plan π there is at least one state s' such that $s' \in S_F$. Note that in the transition system $\mathcal{T}'$ 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,

$$h^*(s) = \mathsf{cost}^*(s, s_F^*) + h^*(s_F^*)$$
$$\geq \min_{s' \in S_F} \{\mathsf{cost}^*(s, s') + h(s')\} = \tilde{h}(s).$$

Consistency: Since h is consistent it holds that $h(s) \leq \mathsf{cost}^*(s, s') + h(s')$ for each s' that is reachable from s. Let $s, s' \in S$, such that $(s, s'; l) \in T$. We should prove that $\tilde{h}(s) \leq \mathsf{cost}(l) + \tilde{h}(s')$. The case is obvious for $s, s' \notin S_E$ since h and $\tilde{h}$ coincide on these states.

Assume then that $s \in S_E$, the successor of s denoted s' can be either in S_F or in S_E. If s' is in S_F we have that

$$\tilde{h}(s) = \min_{s'' \in S_F} \{\mathsf{cost}^*(s, s'') + h(s'')\}$$
$$\leq \mathsf{cost}^*(s, s') + h(s') \leq \mathsf{cost}(l) + \tilde{h}(s'),$$

since $h(s') = \tilde{h}(s')$. Now, assume that $s' \in S_F$. Since h is consistent we have

$$\tilde{h}(s) = \min_{s'' \in S_F} \{\mathsf{cost}^*(s, s'') + h(s'')\} \leq$$
$$\mathsf{cost}^*(s, s') + \min_{s'' \in S_F} \{\mathsf{cost}^*(s', s'') + h(s'')\} \leq$$
$$\mathsf{cost}^*(s, s') + \tilde{h}(s') \leq \mathsf{cost}(l) + \tilde{h}(s'). \qquad \square$$

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

Summary

Overall, we described several approaches that are effective at handling the unbounded nature of numeric variables in PDB heuristics. These approaches can be combined in various ways to a full implementation of numeric PDBs, embodying, to the best of our knowledge, the first adaptation of pattern-database heuristics to numeric planning. In the following, we discuss how the pattern generation can be adapted to numeric variables, as well as how existing cost-partitioning techniques can be applied.

Adaptations to Pattern Generation and Cost Partitioning for Numeric Variables

The great success of PDB heuristics in classical planning stems to a large degree from a sophisticated integrated architecture that (1) creates multiple patterns that capture different parts of a planning task, and (2) a mechanism that admissibly combines these PDB heuristics. These two components are closely connected and have traditionally been investigated in combination (Haslum et al. 2007; Pommerening, Röger, and Helmert 2013). Early works proposed a hill-climbing approach, called iPDB, that creates a collection of

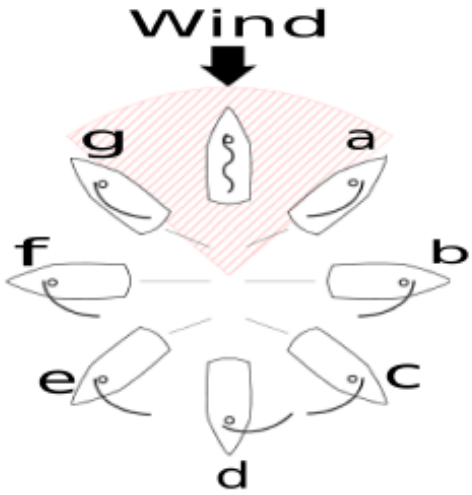

Figure 1: Points of sail (https://en.wikipedia.org) for the SAILING domain (Scala, Haslum, and Thiébaux 2016).

patterns which are evaluated during the process to optimize the heuristic quality on a set of sampled states (Haslum et al. 2007). Here, multiple pattern are combined using the canonical heuristic, which computes sets of additive patterns by checking if two patterns are affected by a joint action. A key operation in the hill-climbing is to extend a pattern from the collection by one variable, which is the reason why it tends to produce relatively few large patterns. We argue that this approach is not well-suited to numeric PDB heuristics, as the construction of the heuristic takes more time compared to non-numeric PDBs. Thus, much fewer candidate collections can be generated, which severely limits the practical applicability of the approach in our setting.

Instead, we opt for a more flexible approach that generates small patterns and combines them using cost partitioning (Pommerening, Röger, and Helmert 2013). More concretely, it enumerates the set of all *interesting* patterns that contain up to a given number of variables, often no more than three. Here, a pattern P is interesting if its variables are weakly connected in the causal graph, and every variable has a path to a goal variable $v_G \in P$. This approach is perfectly suited for numeric planning, as it automatically keeps the size of the PDBs at bay. We can also extend it nicely to consider only patterns with a causal structure that falls into a fragment of IRT for which bounds can be computed.

Examples from Numeric IPC Benchmarks

The missing piece is the admissible combination of the generated pattern collection. Pommerening, Röger, and Helmert (2013) propose the post-hoc optimization method to encode the cost partitioning as a linear program, that considers which operators are relevant for a pattern, i. e., contribute to the heuristic value, and weight every PDB in the collection so that the used action costs do not exceed the original cost. A more fine-grained approach is saturated cost-partitioning, which distributes the cost per PDB heuristic individually for

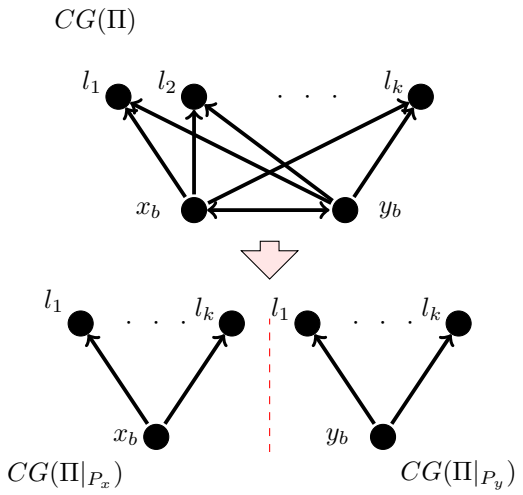

Figure 2: The causal graphs of a task Π from the SAILING domain, and the fork causal graph that are the result of the projections on the task onto P_x and P_y.

every action, to make sure the original cost is not exceeded. Both methods can be adapted to numeric planning tasks.

A key difference lies in the representation of actions in IRT vs. FDR. While in FDR all action parameters are instantiated with concrete objects, i.e., the actions are fully *grounded*, numeric variables are represented in a more declarative way, specifically with additive effects. We can exploit that to distinguish the occurrence of the same action a for different absolute values of the involved numeric variables. Assume a has an additive effect $x \mathrel{+}= c$ on variable x, and further two states s, s' in which x does not have the same value. Then we can distinguish the two occurrences of a in s, respectively s', for cost partitioning, since they will always label different transitions. So we can "split" the label a into sub-labels, one for every distinct value of x for which there is a state in which a is applicable. This is particularly relevant if x is contained in two patterns P, P', where the set of values assigned to x in the reachable states of P is disjoint from the values x has in P'. In such case, we can assign full cost to a in both PDB heuristics without sacrificing admissibility. Intuitively, such action occurrences would correspond to different groundings of a in a classical planning task.

With the necessary components for pattern database collection heuristics in place, we next discuss an example from an established numeric IPC domain, which showcases how our approach will perform on a common numeric planning benchmark.

The SAILING Domain

Consider the SAILING domain where a boat on an unbounded grid needs to visit a set of given locations. Each location is given by a set of coordinates $\{(x_i, y_i)\}_{i=1}^{k} \subseteq \mathbb{Z}^2$, the location of the boat is also given by the coordinates $(x_b, y_b) \in \mathbb{Z}^2$. The wind is blowing from the north, and the boat has 7 movement actions that correspond to the wind rose (see Fig. 1), and a collect action that marks that the boat visited the location $i \in [k]$. For example, the MOVE NORTH EAST action has the numeric effects $x \mathrel{+}= 3$ and $y \mathrel{+}= 3$, the MOVE WEST action has the effect $x \mathrel{+}= -6$, and the MOVE SOUTH action has the effect $y \mathrel{+}= 4$ All move actions have no preconditions. On the other hand, the action COLLECT I has the preconditions $\{x_b = x_i, y_b = y_i\}$ and the effect $\{l_i = \text{visited}\}$. Overall, the problem has 2 numeric variables $\{x_b, y_b\}$ initialized to $(0, 0)$, and k propositional variables, where each variable is initialized to $s_0[l_i] =$ not visited. The goal is to obtain $l_i =$ visited for all k locations.

Note also that the SAILING domain can be transformed into Numeric Additive Planning by Helal and Lakemeyer (Helal and Lakemeyer 2024), and therefore lies in NP.

Projections and Causal Graphs

Consider the two projections of the task on the variables $\mathcal{V} \setminus \{x_b\}$ and $\mathcal{V} \setminus \{y_b\}$, denote these patterns P_x and P_y respectively. Since these pattern are not disjoint-additive, we need to introduce a cost partitioning to obtain additive heuristics. For each action that affects only x_b or only y_b, we give the full cost to the appropriate projection, e.g., $\mathsf{cost}_x(\text{MOVE WEST}) = 1$ and $\mathsf{cost}_y(\text{MOVE WEST}) = 0$. The cost of all other actions, such as MOVE NORTH EAST or COLLECT I, is divided equally between the projections, i.e., its 0.5 both for P_x and P_y.

Next, we note that the variable x_b is a numeric root in the projection on P_x that forms a fork, and y_b has exactly the same causal structure in the projection on P_y (For the causal graphs see Fig. 2). According to Shleyfman et al. (Shleyfman, Gnad, and Jonsson 2023), the numeric values of x_b can be bounded by the intervals $[-2 + \min_{i\in[k]} x_i, 2 + \max_{i\in[k]} x_i]$. The values of y_b are bounded by $[-2 + \min_{i\in[k]} y_i, 2 + \max_{i\in[k]} y_i]$, respectively. Thus, we get two bounded transition system.

Cost-Partitioning for Projections Location

Another way to use PDBs to construct a heuristic for the SAILING domain, would be to divide the variables into k sets of the form $P_i = \{x, y, l_i\}$ with $i \in [k]$. Note that there are only 7 actions that affect either x or y, and there is only one location where the boat should be in P_i. Thus, the problem can be represented as an ILP

$$\min_{\vec{n}\in\mathbb{N}^7} \sum_{i=1}^{7} \mathsf{cost}(a_i) \cdot n_i$$
$$\text{s.t. } A \cdot \vec{n} = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

The matrix $A \in \mathbb{Z}^7 \times \mathbb{Z}^2$ represents the effects of the 7 move actions, here for brevity denoted a_i, each column in this matrix associated with the effects of an action on the $\{x, y\}$ variables. For example, the $a_i =$ MOVE NORTH EAST action has with the effects $x \mathrel{+}= 3$ and $y \mathrel{+}= 3$, correspond to the vector $(+3, +3)^T$ as the ith column of A. In $\vec{n} := (n_1, \ldots, n_7)$ each n_i corresponds to the number of times action a_i is applied,for details on ILP representation see (Helal and Lakemeyer 2024).

Note that the solution to this problem can be computed in polynomial time (Micciancio and Warinschi 2001). Intuitively, this means that we can construct a sufficiently small TS based on the projection onto P_i that will include at least one goal state. Then, using the methods described above, we can apply SCP (Seipp, Keller, and Helmert 2020) and PhO (Pommerening, Röger, and Helmert 2013) to combine the projections $\{P_i\}_{i=1}^{k}$. It's important to note that we may only obtain a PDB that accounts for a subset of states in the original transition system. However, as mentioned earlier, this PDB may be used to enhance an already existing numeric heuristic.

Conclusion

We introduced a novel approach to adopting pattern-database (PDB) heuristics, the current 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 proposed several solutions. Furthermore, we addressed the question of how to combine multiple numeric PDB heuristics admissibly using cost partitioning.

Short of a working implementation, we argue that our approach works well on common numeric planning benchmarks, giving an example of a domain from the International Planning Competition. As numeric PDBs are conceptually very similar to classical PDBs, which have a great success on common non-numeric benchmarks, we believe that these results will carry over also to numeric planning domains.

In future work, we want to investigate how multiple abstraction heuristics for numeric planning can be combined efficiently in a better way, possibly 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

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 *Proc. SOCS*, 155–156.

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial Pattern Databases. In *SARA*, 20–34.

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

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

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

Brafman, R. I.; and Domshlak, C. 2003. Structure and Complexity in Planning with Unary Operators. *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. *JAIR*, 46: 343–412.

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

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

Domshlak, C.; and Dinitz, Y. 2001. Multi-Agent Off-line Coordination: Structure and Complexity. In *ECP*, 277–288.

Edelkamp, S. 2002. Symbolic Pattern Databases in Heuristic Search Planning. In *AIPS*, 274–283.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proc. ICAPS*, 130–137.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive Pattern Database Heuristics. *JAIR*, 22: 279–318.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *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. *AIJ*, 172(8-9): 899–944.

Gigante, N.; and Scala, E. 2023. On the compilability of bounded numeric planning. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence, IJCAI*, volume 23, 5341–5349.

Gnad, D.; Helmert, M.; Jonsson, P.; and Shleyfman, A. 2023. Planning over Integers: Compilations and Undecidability. In *ICAPS*, 148–152. AAAI Press.

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 *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 *Proc. AAAI*, 1007–1012.

Helal, H.; and Lakemeyer, G. 2024. An Analysis of the Decidability and Complexity of Numeric Additive Planning. In *ICAPS*.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proc. AIPS*, 303–312.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *Proc. ICAPS*, 161–170.

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

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

Hoffmann, J. 2003a. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *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 *ECAI*, volume 263, 441–446. IOS Press.

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

Holte, R.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple Pattern Databases. In *icaps*, 122–131.

Illanes, L.; and McIlraith, S. A. 2017. Numeric Planning via Abstraction and Policy Guided Search. In *Proc. IJCAI*, 4338–4345.

Katz, M.; and Domshlak, C. 2009. Structural-Pattern Databases. In *ICAPS*, 186–193.

Katz, M.; and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *AIJ*, 174(12–13): 767–798.

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

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-Cut Heuristics for Optimal Linear Numeric Planning. In *ICAPS*, 203–212. AAAI Press.

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. *J. Artif. Intell. Res.*, 75: 1477–1548.

Li, D.; Scala, E.; Haslum, P.; and Bogomolov, S. 2018. Effect-Abstraction Based Relaxation for Linear Numeric Planning. In *Proc. IJCAI*, 4787–4793.

Micciancio, D.; and Warinschi, B. 2001. A linear space algorithm for computing the Hermite Normal Form. In *International Symposium on Symbolic and Algebraic Computation*, 231–236. ACM.

Piacentini, C.; Castro, M. P.; Ciré, A. A.; and Beck, J. C. 2018a. Compiling Optimal Numeric Planning to Mixed Integer Linear Programming. In *Proc. 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 *Proc. AAAI*, 6254–6261.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proc. AAAI*, 3335–3341.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *IJCAI*, 2357–2364. IJCAI/AAAI.

Scala, E.; Haslum, P.; Magazzeni, D.; and Thiébaux, S. 2017. Landmarks for Numeric Planning Problems. In *Proc. IJCAI*, 4384–4390.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *Proc. IJCAI*, 3228–3234.

Scala, E.; Ramírez, M.; Haslum, P.; and Thiébaux, S. 2016. Numeric Planning with Disjunctive Global Constraints via SMT. In *Proc. ICAPS*, 276–284.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *J. Artif. Intell. Res.*, 67: 129–167.

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

Shleyfman, A.; Gnad, D.; and Jonsson, P. 2023. Structurally Restricted Fragments of Numeric Planning–a Complexity Analysis. In *AAAI*, 10, 12112–12119.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient Implementation of Pattern Database Heuristics for Classical Planning. In *SOCS*, 49–56. AAAI Press.