Getting the Most Out of Pattern Databases for Classical Planning

Florian Pommerening and **Gabriele Röger** and **Malte Helmert**
Universität Basel
Basel, Switzerland
{florian.pommerening,gabriele.roeger,malte.helmert}@unibas.ch

Abstract

The iPDB procedure by Haslum et al. is the state-of-the-art method for computing additive abstraction heuristics for domain-independent planning. It performs a hill-climbing search in the space of pattern collections, combining information from multiple patterns in the so-called *canonical heuristic*.

We show how stronger heuristic estimates can be obtained through linear programming. An experimental evaluation demonstrates the strength of the new technique on the IPC benchmark suite.

1 Introduction

Heuristic search with *abstraction heuristics* is a standard approach for solving state-space search problems. Algorithms in this family are among the state of the art both for domain-specific solvers for combinatorial puzzles [e.g., Felner *et al.*, 2004; Yang *et al.*, 2008] and for domain-independent classical planning systems [e.g., Haslum *et al.*, 2007; Nissim *et al.*, 2011; Sievers *et al.*, 2012].

The most commonly used abstraction heuristics are based on so-called *pattern databases* (PDBs), which provide perfect cost estimates for subproblems [Culberson and Schaeffer, 1998; Edelkamp, 2001]. The main drawback of PDBs is that the aspects of the overall problem that they can perfectly cover (the *pattern*) must usually be quite small due to memory constraints. Therefore, state-of-the-art solvers generally use multiple patterns that cover different problem aspects.

This leaves us with two major challenges in designing efficient PDB-based search algorithms:

- *pattern selection*: how do we choose suitable patterns?
- *heuristic combination*: how can we efficiently derive informative admissible heuristics from multiple PDBs?

The state-of-the-art pattern selection method in classical planning is the *iPDB* algorithm by Haslum *et al.* [2007], which performs a hill-climbing search in the space of pattern collections to select a suitable set of patterns. In Section 4, we point out some problems of this approach and suggest a different method for pattern selection, which systematically generates all patterns up to a certain size.

Systematic pattern generation is not new: for example, it has been successfully applied to combinatorial puzzles and the minimum vertex cover problem by Felner *et al.* [2004]. We contribute a new criterion for filtering out uninteresting patterns (ones that cannot influence the heuristic value), which is the best possible such criterion among all criteria based on the *causal graph* of a problem instance.

On the topic of heuristic combination, there exists a certain tension between theory and practice. On the one hand, Katz and Domshlak [2010] introduce a very general notion of *cost partitioning* for admissible heuristics and show how optimal cost partitionings for PDB heuristics can be computed in polynomial time. On the other hand, practical implementations only exploit a much weaker concept of *additivity*, a sufficient condition for sums of PDB heuristics to be admissible. This notion of additivity gives rise to the so-called *canonical heuristic* for a set of patterns [Haslum *et al.*, 2007].

The reason why Katz and Domshlak's elegant theory has not yet had an influence on practical PDB-based planning systems is that it requires solving huge linear programs for every heuristic computation – up to millions of variables and billions of constraints for realistic problem sizes. In Section 3, we describe a novel way of combining PDB heuristics, motivated by the theory of cost partitioning, which can be computed quickly and dominates the canonical heuristic.

In Section 5, we experimentally explore the interaction of several pattern selection and heuristic combination methods and show that our new heuristic with a systematic pattern generation significantly outperforms the previous state of the art in PDB-based search algorithms.

2 Background

We consider planning tasks in finite-domain representation. A planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, c \rangle$, where $\mathcal{V}$ is a finite set of *state variables* v, each with a finite domain $\mathcal{D}_v$. A *state* is a complete variable assignment over $\mathcal{V}$. Each *operator* $\langle pre; eff \rangle$ in the set of operators $\mathcal{O}$ consists of a precondition and an effect, which are partial variable assignments over $\mathcal{V}$. An operator is *applicable* in a state if the precondition is consistent with the state. The *successor state* results from updating the state according to the operator effect. The function $c : \mathcal{O} \to \mathbb{R}_0^+$ assigns a cost to each operator. The aim is to find a sequence of operators whose sequential application leads from the *initial state* s_0 to a state consistent with the *goal description* $s_\star$, which is given as a partial variable assignment. An *optimal* plan is an operator sequence with

$$
\begin{aligned}
\mathcal{V} &= \{A, B, C\} \\
\mathcal{D}_v &= \{0, 1, 2, 3, 4\} \text{ for all } v \in \mathcal{V} \\
s_0 &= \{A = 0, B = 0, C = 0\} \\
s_\star &= \{A = 3, B = 3, C = 3\} \\
\mathcal{O} &= \{\mathit{inc}_x^v \mid v \in \mathcal{V}, x \in \{0, 1, 2\}\} \cup \{\mathit{jump}^v \mid v \in \mathcal{V}\}, \\
&\quad \text{where} \quad \mathit{inc}_x^v = \langle v = x; v := x + 1 \rangle \\
&\qquad\quad \mathit{jump}^v = \langle v' = 4 \text{ for all } v' \neq v; v := 3 \rangle \\
c(o) &= 1 \text{ for all } o \in \mathcal{O}
\end{aligned}
$$

Figure 1: Example planning task.

minimal cost among all solutions, where the cost is defined as the sum of the cost of all operators in the sequence.

Figure 1 shows a planning task that we use as a running example. There are three variables, all with domain $\{0, \ldots, 4\}$. The objective is to change their values from 0 to 3. For each variable v there are operators that can increment it by 1 if the value is not already 4. In addition there are "jump" operators which immediately set a variable to 3 but are only applicable if all other variables have value 4. All operators have unit cost. Clearly, the jump operators are useless for solving the task, and hence all optimal solutions must increment each variable three times for a total cost of 9.

PDB heuristics. PDB heuristics estimate the cost of reaching the goal from a state s by computing the exact cost-to-goal in a smaller planning task defined by a *pattern* $P \subseteq \mathcal{V}$. This smaller planning task is obtained by projecting away all information (in s, $s_\star$ and the operators) about variables outside the pattern. For example, with the pattern $\{A, C\}$, the operator jump^A in the running example is projected to $\langle C = 4; A := 3 \rangle$. Such a projection is a special case of (homomorphic) *abstraction* [e.g., Helmert and Domshlak, 2009], and states obtained in this way are called *abstract states*.

Standard PDB heuristics (as considered in this paper) precompute cost-to-goal information for all abstract states and store them in a table, the eponymous *pattern database* (PDB). This assumes that the number of abstract states is small enough to fit into memory.

In our running example, heuristics for singleton patterns like $\{A\}$ give an estimate of 1 for the initial state because jump^A immediately solves the abstract task. (All its preconditions disappear when projecting to a singleton pattern.) All PDBs on two variables yield initial state estimates of 6, counting the increment operators for these variables.

Canonical heuristic. The maximum of multiple PDB heuristics is obviously admissible and dominates the individual heuristics. However, stronger estimates are often possible. Consider a set of patterns (a *pattern collection*) where no operator affects more than one pattern. (That is, operators have a non-empty effect in at most one of the projected tasks.) Then the sum of heuristic estimates is also admissible. Such patterns are called *additive* [e.g., Edelkamp, 2001]. Haslum *et al.* [2007] generalize this notion of additivity to arbitrary (not necessarily additive) pattern collections $\mathcal{C}$. Let $\mathit{MAS}(\mathcal{C})$ denote all maximal (w.r.t. set inclusion) additive subsets of $\mathcal{C}$. Then the *canonical heuristic* value $h^{\mathcal{C}}(s)$ is defined as

$$h^{\mathcal{C}}(s) = \max_{\mathcal{A} \in \mathit{MAS}(\mathcal{C})} \sum_{P \in \mathcal{A}} h^P(s).$$

iPDB pattern selection. Haslum *et al.* [2007] also introduced a hill-climbing algorithm for pattern selection called the *iPDB procedure*. It starts with an initial collection with singleton patterns for all goal variables. Given the current collection $\mathcal{C}$, a new collection $\mathcal{C}'$ is created by selecting a pattern $P \in \mathcal{C}$, extending it with a new variable $v \notin P$, and adding the resulting pattern to the collection: $\mathcal{C}' = \mathcal{C} \cup \{P \cup \{v\}\}$. All such collections whose PDBs do not exceed a fixed memory limit form the search neighborhood of $\mathcal{C}$. The procedure then selects the "best" neighbor by evaluating each candidate on sample states. The process repeats until all candidates offer no or negligible improvements over $\mathcal{C}$.

For efficiency, the hill-climbing search uses two pruning criteria to rule out candidates which cannot possibly lead to improved heuristic estimates. First, extending a pattern P with a non-goal variable which does not causally influence a variable in P does not increase the estimates of the resulting PDB. Second, a pattern should never contain causally disconnected variables, because these are better included as separate patterns that can be combined additively.

3 Post-Hoc Optimization Heuristic

Consider the behavior of the canonical heuristic on the initial state of our running example. We assume that the pattern collection includes all patterns that are proper abstractions: $\mathcal{C} = \{\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}\}$. On s_0, all single-variable PDBs yield a heuristic estimate of 1 and all two-variable PDBs yield an estimate of 6. The best additive combinations of such patterns take a two-variable pattern like $\{A, B\}$ and combine it with a one-variable pattern like $\{C\}$ for an overall heuristic estimate of 7.

Can we do better than this without performing explicit cost partitioning in the abstract planning tasks, i. e., without recomputing any of the PDBs? The answer to this is "yes!" if we also take into account *which operators* are relevant for each pattern. Each operator in the task modifies exactly one variable, and the operators relevant for pattern P are those which modify a variable in P. Let us call an operator modifying variable v a "type-v operator". From $h^{\{A,B\}} = 6$, we can conclude that any solution includes at least 6 operators that are type A or type B; from the other two-variable patterns we derive that there must also be at least 6 operators that are type A or type C, and at least 6 operators that are type B or type C. A bit of algebra shows that at least 9 operators are necessary to simultaneously satisfy these three constraints.

We now show how this argument can be generalized to arbitrary planning tasks, where operators may affect multiple variables, and how the reasoning process can be efficiently automated. For this purpose, we construct a linear program (LP) with one variable X_o for each operator $o \in \mathcal{O}$. Intuitively, the LP variable X_o represents the costs incurred by operator o in some (fixed, but unknown) optimal plan. For example, if o occurs twice in the plan and has a cost of 3, then $X_o = 6$. Clearly we must have $X_o \geq 0$ for all $o \in \mathcal{O}$. Since PDB heuristics are admissible, we know that the PDB heuristic value cannot exceed the total cost incurred by all operators:

$$h^P(s) \leq \sum_{o \in \mathcal{O}} X_o$$

Since operators which do not affect any variable in a pattern do not induce state changes in the abstract task, they cannot contribute to the estimate of the PDB. Therefore we can tighten the constraint for h^P by omitting variables for such "irrelevant" operators to

$$h^P(s) \leq \sum_{o \in \mathcal{O}: o \text{ affects } P} X_o$$

The total cost of the given plan is $\sum_{o \in \mathcal{O}} X_o$. Minimizing this value subject to the constraints for all PDB heuristics yields an admissible estimate for s because all accounted costs are justified by the admissibility of some heuristic.

We can reduce the size of the LP by aggregating variables which always occur together in the constraints. This happens when several operators are relevant for exactly the same PDBs. In the example, the operators inc_x^A and $direct^A$ are relevant for exactly the PDBs with patterns including A, so we can combine their LP variables to a new variable X_A. Similarly, we can aggregate the remaining operators as X_B and X_C. The resulting LP for an arbitrary state s is:

$$\begin{aligned}
&\text{Minimize } X_A + X_B + X_C \text{ subject to}\\
X_A &\geq h^{\{A\}}(s)\\
X_B &\geq h^{\{B\}}(s)\\
X_C &\geq h^{\{C\}}(s)\\
X_A + X_B &\geq h^{\{A,B\}}(s)\\
X_A + X_C &\geq h^{\{A,C\}}(s)\\
X_B + X_C &\geq h^{\{B,C\}}(s) \text{ and}\\
X_A \geq 0, X_B &\geq 0, X_C \geq 0
\end{aligned}$$

To solve the LP for the initial state of the example, we remind the reader that $h^P(s_0) = 1$ for single-variable patterns and $h^P(s_0) = 6$ for two-variable patterns. The LP then has the unique optimal solution $X_A = X_B = X_C = 3$, yielding a heuristic value of 9, which is perfect for this example and better than the canonical heuristic estimate of 7.

We will now formalize these considerations for general planning tasks. We begin by observing that the approach is not limited to PDB heuristics. The only properties of PDB heuristics that we exploited are that they are admissible and that certain operators do not contribute to the heuristic value of certain PDBs. To express these requirements succinctly, we borrow some concepts from the theory of *cost partitioning* [Katz and Domshlak, 2010]. If Π is a planning task with operators $\mathcal{O}$ and $c' : \mathcal{O} \to \mathbb{R}_0^+$ is an arbitrary cost function on its operators, we write $\Pi_{c'}$ for the planning task which is equal to Π except that its operator cost function is c'. Then the *relevant operator partition* specifies sets of operators whose LP variables can be combined:

Definition 1 (Relevant operator partition)**.** *Let Π be a planning task with cost function c. For all $i = 1, \ldots, n$, let $c_i \leq c$ be a cost function and h_i be an admissible heuristic for Π_{c_i}.*

The relevant operator partition *$\mathcal{O}/\!\sim$ for $h_1, \ldots, h_n$ is the partition of $\mathcal{O}$ induced by the equivalence relation $\sim$ with:*

$$o \sim o' \quad \textit{iff} \quad \{i \mid c_i(o) > 0\} = \{i \mid c_i(o') > 0\}.$$

We can now define the *post-hoc optimization heuristic* that generalizes the above example as follows:

Definition 2 (Post-hoc optimization heuristic)**.** *Let Π be a planning task with cost function c. For all $i = 1, \ldots, n$, let $c_i \leq c$ be a cost function and h_i be an admissible heuristic for Π_{c_i}. Let s be a state of Π.*

The estimate of the post-hoc optimization heuristic $h^{\text{PhO}}(s)$ *is the objective value of the linear program:*

$$\textit{Minimize } \textstyle\sum_{[o] \in \mathcal{O}/\sim} X_{[o]} \textit{ subject to}$$

$$\begin{aligned}
\sum_{[o] \in \mathcal{O}/\sim : c_i(o) > 0} X_{[o]} &\geq h_i(s) \qquad \textit{for all } i \in \{1, \ldots, n\}\\
X_{[o]} &\geq 0 \qquad \textit{for all } [o] \in \mathcal{O}/\sim.
\end{aligned}$$

The admissibility of its component heuristics leads to admissibility of the post-hoc optimization heuristic:

Theorem 3 (Admissibility of h^{PhO})**.** *The post-hoc optimization heuristic is admissible.*

Proof. Let π be an optimal plan for $\Pi = \langle \mathcal{V}, \mathcal{O}, s, s_\star, c \rangle$. (If the task is unsolvable in state s, every heuristic is trivially admissible in s.) We write $h_i^*(s)$ for the cost of an optimal plan for Π_{c_i}. For any operator set $O \subseteq \mathcal{O}$ and cost function c', let $cost_\pi(O, c')$ denote the cost incurred by the operators from set O in plan π under some cost function c'. We set each variable $X_{[o]}$ to the cost of the operators in $[o]$ under the normal cost function c: $X_{[o]} = cost_\pi([o], c)$. We will show that this variable assignment is feasible for the linear program. The constraints $X_{[o]} \geq 0$ are obviously satisfied. For the other constraints we can see that

$$\begin{aligned}
h_i(s) &\overset{(*)}{\leq} h_i^*(s) \overset{(**)}{\leq} cost_\pi(\mathcal{O}, c_i) = \sum_{[o] \in \mathcal{O}/\sim} cost_\pi([o], c_i)\\
&= \sum_{\substack{[o] \in \mathcal{O}/\sim:\\ c_i(o) > 0}} cost_\pi([o], c_i) \overset{(***)}{\leq} \sum_{\substack{[o] \in \mathcal{O}/\sim:\\ c_i(o) > 0}} cost_\pi([o], c)\\
&= \sum_{\substack{[o] \in \mathcal{O}/\sim:\\ c_i(o) > 0}} X_{[o]}.
\end{aligned}$$

Inequality (*) holds because h_i is admissible for Π_{c_i}. Inequality (**) holds because $cost_\pi(\mathcal{O}, c_i)$ denotes the cost of π in Π_{c_i} and hence cannot be lower than $h_i^*(s)$, the cost of an optimal plan for Π_{c_i}. Inequality (***) follows from $c_i(o) \leq c(o)$. We conclude that $h_i(s) \leq \sum_{[o] \in \mathcal{O}/\sim : c_i(o) > 0} X_{[o]}$ and hence the chosen assignment for $X_{[o]}$ is feasible.

The objective value for the chosen assignment is $\sum_{[o] \in \mathcal{O}/\sim} X_{[o]}$, which equals the cost of π with the chosen values of $X_{[o]}$. This is equal to $h^*(s)$ by the optimality of π.

Any feasible solution for a minimization LP provides an upper bound on the optimal objective value, so the value is at most $h^*(s)$. This shows that the heuristic is admissible. □

To gain some more insight into the post-hoc optimization heuristic, we have a look at the corresponding dual program:

$$\text{Maximize } \sum_{i=1}^{n} Y_i h_i(s) \text{ subject to}$$

$$\sum_{i \in \{1,\ldots,n\}: c_i(o) > 0} Y_i \leq 1 \qquad \text{for all } [o] \in \mathcal{O}/\!\sim$$

$$Y_i \geq 0 \qquad \text{for all } i \in \{1, \ldots, n\}.$$

By the duality theorem, a bounded feasible LP and its dual have the same optimal value. However, the dual reveals a different perspective: the objective function shows that we compute a state-specific cost partitioning which cannot change individual operator costs but can only scale the operator costs within each heuristic by a factor Y_i that depends on the heuristic, but not on the operator. The constraints ensure that the sum of the scaled estimates stays admissible.

3.1 Relation to the Canonical Heuristic

In addition to providing an interpretation of the heuristic in terms of cost partitioning, the dual LP also reveals an interesting relation between the post-hoc optimization heuristic for PDB heuristics and the canonical heuristic for the same PDBs. For a pattern collection $\mathcal{C}$ the post-hoc optimization heuristic is defined in the obvious way:

Definition 4. *Let Π be a planning task with cost function c, and let $\mathcal{C} = \{P_1, \ldots, P_n\}$ be a pattern collection for Π.*

The post-hoc optimization heuristic $h^{\text{PhO}}_{\mathcal{C}}$ for $\mathcal{C}$ *is* h^{PhO} *where $h_i = h^{P_i}$, $c_i(o) = c(o)$ for all operators o that affect a variable contained in P_i, and $c_i(o) = 0$ otherwise.*

We now show that, in a certain sense, the post-hoc optimization heuristic computes the LP relaxation of an integer program formulation of the canonical heuristic.

Theorem 5. *Consider the dual D of the LP solved by $h^{\text{PhO}}_{\mathcal{C}}$ in state s for a given pattern collection $\mathcal{C}$. If we restrict the variables in D to integers, the objective value is the canonical heuristic value $h^{\mathcal{C}}(s)$.*

Proof. Let V^* be the optimal value of the integer program corresponding to D (i. e., where the domain of all Y_i variables is set to $\{0, 1\}$). We will establish $h^{\mathcal{C}}(s)$ as an upper and lower bound of V^*, thus proving equality.

For clarity, in the following we denote the variables of the dual with Y_P for the variable related to h^P (in contrast to our previous notation Y_i for the variable related to heuristic h_i).

Let $\mathbf{Y}^*$ be an optimal solution of the integer program. Then $\mathcal{A} = \{P \in \mathcal{C} \mid Y^*_P = 1\}$ is an additive subset of $\mathcal{C}$, which we show by contradiction: assume that there are two different patterns $P_i, P_j \in \mathcal{A}$ which are *not* additive, i. e., there is an operator o which affects both patterns. Then $c_{P_i}(o) > 0$ and $c_{P_j}(o) > 0$ and $\mathbf{Y}^*$ violates the constraint $1 \geq \sum_{P \in \mathcal{C}: c_P(o) > 0} Y^*_P \geq Y^*_{P_i} + Y^*_{P_j} = 2$. Thus $\mathcal{A}$ is additive and we get $h^{\mathcal{C}}(s) \geq \sum_{P \in \mathcal{A}} h^P(s) = V^*$.

Conversely, let $\mathcal{A}^*$ be a maximal additive subset of $\mathcal{C}$ that maximizes $h^{\mathcal{A}^*}(s)$, i. e., establishes the estimate $h^{\mathcal{C}}(s)$. Then $\mathbf{Y}$ with $Y_P = 1$ for $P \in \mathcal{A}^*$ and $Y_P = 0$ otherwise is a feasible solution for the IP version of D: obviously, all variables are in the range $\{0, 1\}$. Assume that the variable assignment violates a constraint $\sum_{P \in \mathcal{C}: c_P(o) > 0} Y_P \leq 1$ for an operator block $[o]$. Then there must be two different variables Y_P with $Y_P = 1$ and $c_P(o) > 0$. From the definition of Y_P, both patterns are included in $\mathcal{A}^*$ and hence additive. This contradicts that $c_P(o) > 0$ for both of them. We conclude that $\mathbf{Y}$ is feasible and its value $h^{\mathcal{C}}(s)$ is a lower bound for V^*.

Overall we get $h^{\mathcal{C}}(s) \leq V^* \leq h^{\mathcal{C}}(s)$, so $V^* = h^{\mathcal{C}}(s)$. □

Since every feasible variable assignment of an integer program (IP) is feasible for the corresponding LP, the value of the IP is a lower bound for the LP value. We conclude:

Corollary 1. *The post-hoc optimization heuristic $h^{\text{PhO}}_{\mathcal{C}}$ dominates the canonical heuristic $h^{\mathcal{C}}$ for the same pattern collection $\mathcal{C}$.*

Our introductory example already showed that $h^{\text{PhO}}_{\mathcal{C}}$ can give *strictly* larger estimates than $h^{\mathcal{C}}$, which is also confirmed by the experimental evaluation.

3.2 Computation Effort

We conclude our discussion of the post-hoc optimization heuristic by considering its computational cost. We assume that the component heuristics h_i are efficiently computable, as in the case of PDB heuristics.

To compute $h^{\text{PhO}}(s)$, we must solve an LP with one variable per block in the operator partition and one constraint per component heuristic. The number of blocks in the partition can never exceed the number of operators and is often much smaller. For example, in tasks where all operators have only one effect, the number of blocks is bounded by the number of state variables. Similarly, due to the exponentially growing space requirements of PDB heuristics, the number of component heuristics is usually not too large. In particular, this analysis shows that the post-hoc optimization heuristic can be computed in polynomial time (in the planning task size and the number of component heuristics), as LPs can be solved in polynomial time in the number of variables and constraints.

We also note that only the bounds $h_i(s)$ in the LP depend on the evaluated state s, which allows LP solvers to reuse computations from previous states. If we use the dual LP to compute the heuristic value, only the objective function differs from state to state while the feasible set remains constant.

In contrast, the computation of the canonical heuristic requires iterating over all maximal additive subsets of the pattern collection. The number of such subsets can grow exponentially with the number of patterns (= component heuristics), so despite being dominated by h^{PhO}, the canonical heuristic can be exponentially more expensive to compute.

Considering that h^{PhO} can be understood in terms of cost partitioning by means of linear programs, another obvious comparison is to the LP-based optimal cost partitioning of abstraction heuristics by Katz and Domshlak [2010]. Their approach can partition costs much more flexibly than post-hoc cost partitioning because the cost of each individual operator in each abstraction can be set individually. When applied to PDBs, their approach builds an LP with one constraint for *every state transition* of every abstract planning task. For larger PDBs, this can amount to billions or trillions of constraints and is hence practically infeasible to solve even once, let alone for every evaluated state within a heuristic search algorithm. Consequently, no implementation of the approach has been discussed in the literature.

	HC^{C}			HC^{PhO}	Sys^{1}			Sys^{2}			Sys^{3}			Sys^{*}		
	h^{C}	h^{PhO}	h^{OCP}	h^{PhO}	h^{C}	h^{PhO}	h^{OCP}	h^{C}	h^{PhO}	h^{OCP}	h^{C}	h^{PhO}	h^{OCP}	h^{PhO}	h^{blind}	$h^{\text{LM-Cut}}$
barman (20)	**4**	**4**	0	0	**4**	**4**	0	**4**	**4**	0	0	0	0	4 (1–2)	4	4
elevators (20)	**16**	**16**	0	**16**	9	9	4	**16**	15	0	**16**	14	0	15 (2)	9	18
floortile (20)	**2**	**2**	0	**2**	**2**	**2**	**2**	**2**	**2**	0	**2**	**2**	0	2 (1–3)	2	7
nomystery (20)	16	16	3	16	12	12	8	18	18	6	**19**	**19**	3	19 (3–4)	8	14
openstacks (20)	**14**	**14**	5	**14**	**14**	**14**	5	5	**14**	0	2	9	0	14 (1–2)	8	14
parcprinter (20)	8	8	8	8	11	11	7	7	13	15	5	**18**	2	20 (4)	6	13
parking (20)	**5**	**5**	1	**5**	**5**	**5**	1	0	1	0	0	0	0	5 (1)	0	3
pegsol (20)	0	0	0	0	**17**	**17**	12	5	**17**	1	1	16	0	17 (1–2)	17	17
scanalyzer (20)	**10**	**10**	1	7	**10**	**10**	5	**10**	8	1	**10**	4	0	10 (1)	9	12
sokoban (20)	**20**	**20**	18	**20**	19	19	13	**20**	**20**	2	**20**	13	0	20 (2)	16	20
tidybot (20)	**14**	**14**	6	11	13	13	5	**14**	**14**	6	8	**14**	1	14 (2–3)	10	14
transport (20)	6	6	2	6	6	6	4	6	6	0	**9**	8	0	8 (3)	6	6
visitall (20)	**16**	**16**	10	**16**	**16**	**16**	15	**16**	**16**	10	**16**	**16**	6	16 (1–3)	9	11
woodworking (20)	2	2	2	1	5	5	2	3	**10**	2	1	9	0	10 (2)	2	12
Sum IPC 2011 (280)	133	133	56	122	143	143	83	126	**158**	43	109	142	12	174	112	165
IPC 1998–2008 (1116)	456	459	241	426	449	449	355	446	**475**	231	406	422	129	501	394	598
Sum (1396)	589	592	297	548	592	592	438	572	**633**	274	515	564	141	675	506	763
Column	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p

Table 1: Coverage on the IPC benchmark suite. Best results among PDB-based heuristics highlighted in bold.

4 Generating Pattern Collections

The iPDB procedure performs hill-climbing to find a pattern collection. However, certain informative patterns are not reachable by this search because they would require extending a pattern by several variables at once to get a heuristic improvement. Consider for example a task with variables $\{A, B_1, B_2, G_1, G_2\}$ with initial values 0 and goals $G_1 = 1$ and $G_2 = 1$. The operators are $\langle A = 0; A := i\rangle$, $\langle A = i; A := 0\rangle$, $\langle A = i; B_i := 1\rangle$ and $\langle B_i = 1; G_i := 1\rangle$ for $i = 1, 2$.

Growing patterns one variable at a time, iPDB can extend the goal-variable pattern $\{G_1\}$ by B_1 and A but not by B_2, because this would not increase the estimates of the corresponding PDB. G_2 would not be included because it is causally disconnected. Starting from the pattern $\{G_2\}$ leads to a symmetric situation where B_1 cannot be added. However, the pattern consisting of all variables would contribute to the heuristic estimate because it is the only one detecting that the value of variable A needs to be changed back to 0 at some point. Indeed, in many domains from the IPC benchmark suite (e. g., blocks, parking and pathways) iPDB hill-climbing already stops at the first iteration, so that the pattern collection only consists of the initial single-variable patterns.

To counter this problem, we propose a different approach of generating pattern collections. It is based on the trivial idea of systematically including all patterns up to a given size, which has been shown to lead to informative heuristics in several classical search domains [Felner *et al.*, 2004]. Without further modifications, this approach would often lead to prohibitively large collections with many uninformative patterns, so we use a finer version of the iPDB pruning criteria to exclude useless patterns.

Like the original approach for pruning unnecessary patterns in iPDB, our criteria are defined in terms of the *causal graph* of a planning task. The causal graph of Π is a directed graph whose nodes are the variables of Π. It has a *precondition* arc from node u to node v if there is an operator which has a precondition on u and an effect on $v \neq u$. If an operator affects both u and v, the causal graph contains *co-effect* arcs from u to v and vice versa. Our systematic method generates all patterns up to a certain size which are *interesting* according to the following definition:

Definition 6. *A pattern P is* interesting *if*

1. *the subgraph of the causal graph induced by P is weakly connected, and*
2. *the subgraph of the causal graph induced by P contains a directed path via precondition arcs from each node to some goal variable node.*

This definition is similar to the one used in iPDB, but finer because it distinguishes the role of precondition arcs and co-effect arcs. (The latter are only used for condition 1.) It is the *most restrictive* possible causal-graph-based definition of interestingness in the following sense:

(A) Every pattern that is *not interesting* according to Definition 6 is redundant in the sense that it could be replaced by a smaller pattern or several (additive) smaller patterns providing the same heuristic estimates.

(B) Every *more restrictive* definition of interestingness based on the causal graph (that would consider some pattern uninteresting which our definition considers interesting) does not satisfy (A).

For space reasons, we leave out a proof of (B) and only briefly sketch a proof of (A) here. If P fails to satisfy condition 1., it consists of independent abstract subtasks and can be replaced without loss by a set of additive patterns, one for each weakly connected component. If P fails to satisfy condition 2., it contains a non-goal variable v which is never required as precondition of an operator that is necessary for solving the abstract task. Therefore we can remove v from the pattern without reducing any heuristic estimate.

5 Experimental Evaluation

For the experimental evaluation, we extended the iPDB implementation by Sievers *et al.* [2012] in the Fast Downward planning system [Helmert, 2006] with an implementation of the post-hoc optimization heuristic h^{PhO} and of the optimal

PDB cost partitioning approach h^{OCP} of Katz and Domshlak [2010]. We conducted experiments on all IPC benchmark tasks for optimal planners with a time limit of 30 minutes and a memory limit of 2 GB. All experiments were performed on machines with two 8-core Intel Xeon E5-2660 processors under full load (16 concurrent runs). For iPDB we used the default parameters, which are commonly accepted as a good choice.

For each benchmark task and planner configuration, we measure the following features:

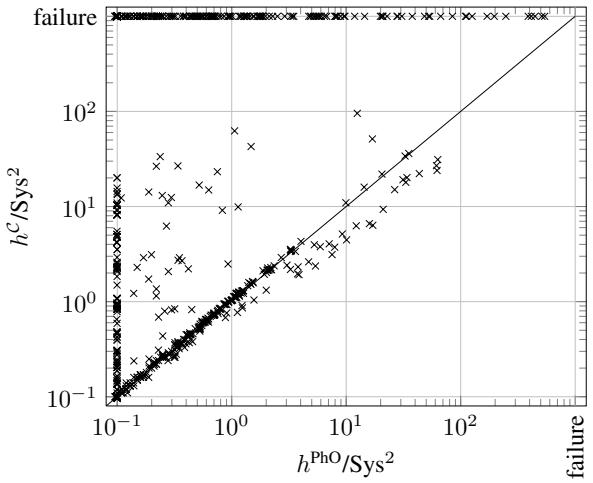
- *coverage*: was the task solved within 30 minutes?
- *setup time*: time taken to select and compute the PDBs, to compute the maximal additive pattern sets (for $h^{\mathcal{C}}$), and to set up the LP (for h^{PhO} and h^{OCP})
- *initial h computation time*: time to perform the heuristic computation for the initial state (after all setup)
- *initial h value*: the heuristic value of the initial state
- *expansions until last jump* (for solved tasks): number of expanded nodes until the last f layer is reached by A*
- *node evaluation rate* (for solved tasks): number of heuristic evaluations to solve the task divided by search time (excluding setup time)

Overall results. Table 1 shows coverage results on the IPC benchmark suite. For space reasons, we only include per-domain results for the IPC-2011 instances and present aggregated numbers for older tasks. Each configuration in columns a–m is specified by the pattern selection method and the heuristic.

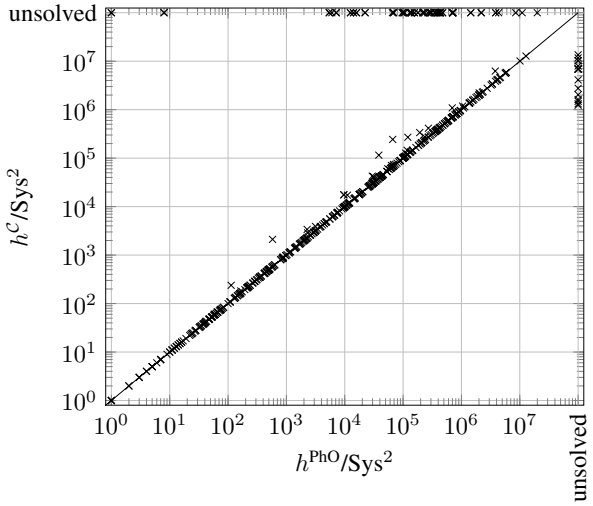
Columns a and b compare the canonical and the post-hoc optimization heuristic on the pattern collection generated by the iPDB hill-climbing procedure. There is little difference between the two heuristics in coverage, and setup time (not shown) is also comparable for both heuristics. A comparison of node expansions (also not shown) reveals that h^{PhO} is not much better informed than $h^{\mathcal{C}}$ in this setting. One possible explanation is that iPDB hill-climbing uses the canonical heuristic to determine the "best" neighbor and therefore prefers pattern collections which work especially well with this heuristic.

To test this conjecture, we modified the hill-climbing search to use the post-hoc optimization heuristic instead of the canonical heuristic to evaluate the hill-climbing neighborhood. The result is shown in column d. Instead of the expected improvement, coverage drops significantly from 592 to 548 tasks. This surprise brought up the question if there is a conceptual problem with the hill-climbing method and led to the considerations in Section 4, which gave rise to the systematic pattern generation approach.

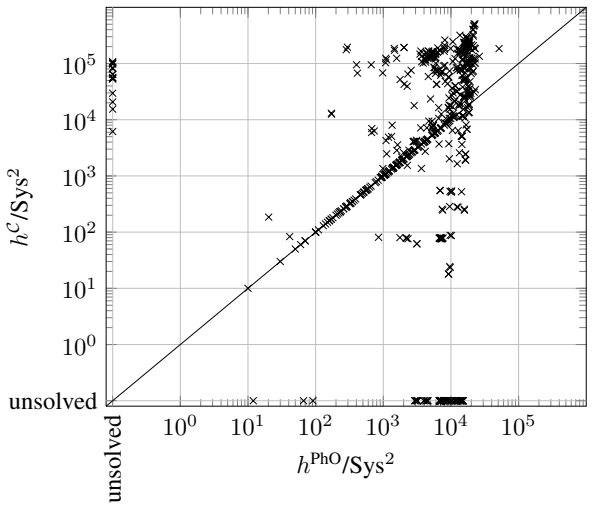
On the systematically generated patterns, h^{PhO} and $h^{\mathcal{C}}$ perform equally on patterns of size 1 (columns e and f), but on larger (and therefore more) patterns h^{PhO} performs far better than $h^{\mathcal{C}}$ (columns h, i and k, l). While the behavior is not consistent over *all* domains, the post-hoc optimization heuristic solves significantly more instances in many domains, overall outperforming the canonical heuristic by 61 instances on the patterns up to size 2 and by 49 instances on the patterns up to size 3.



(a) Heuristic setup time in seconds.

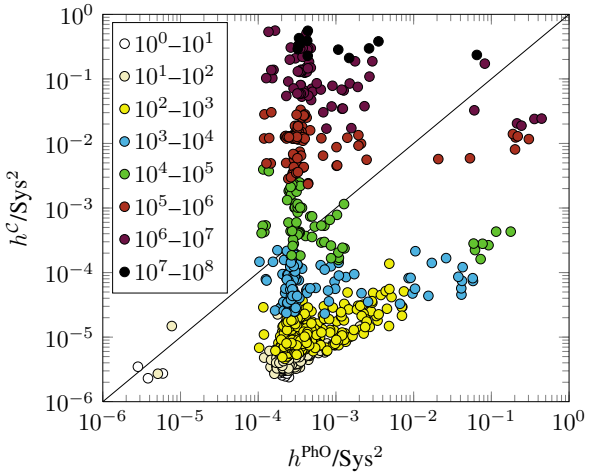


(b) Number of expansions until last jump.

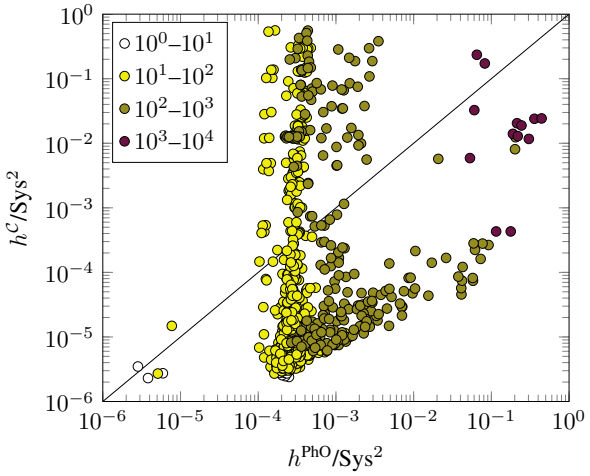


(c) Node evaluation rate in evaluations per second.

Figure 2: Comparison of h^{PhO} and $h^{\mathcal{C}}$ with all patterns up to size 2.



(a) Number of maximal additive pattern sets.



(b) Number of patterns.

Figure 3: Evaluation time for the initial state with h^{PhO} and h^C and systematically generated size-2 patterns.

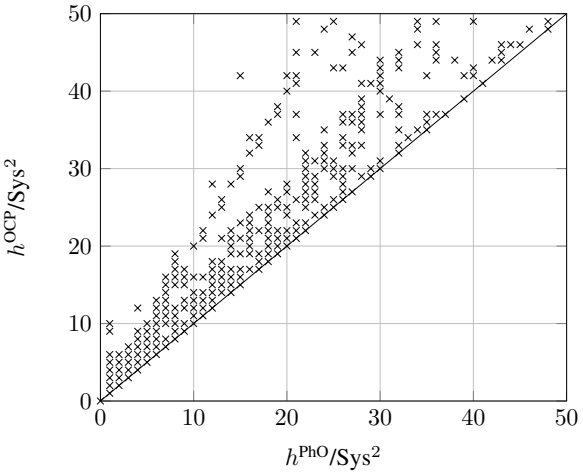


Figure 4: Initial h values.

A deeper look. To understand the reasons for the better performance of h^{PhO} compared to h^C, Figure 2 compares (a) the heuristic setup time, (b) the number of expansions, and (c) the evaluation rate of both heuristics with the systematically generated patterns up to size 2. Panel (b) shows that there are only a few instances where the theoretical dominance of the post-hoc optimization heuristic translates into better guidance, and panel (c) shows that on tasks solved by both approaches, the canonical heuristic computations tend to be faster.

However, panel (a) reveals that the canonical heuristic often requires far more setup time (almost all the exceptions are in the pipesworld-tankage domain) and more importantly often fails to complete heuristic setup within the available resources. In all cases, the cause of the failure is running out of memory. On 9 of the 1396 task, the memory limit is not sufficient to build all PDBs for the pattern collection, which affects both heuristics. However, for the canonical heuristic, 360 additional instances run out of memory during the generation of the maximal additive pattern sets. This indicates that in the case of systematic pattern collections, the number of maximal additive sets is often prohibitively large.

To clarify the influence of the characteristics of the pattern collection on heuristic evaluation time, Figure 3 plots the time required for the initial heuristic evaluation on all tasks where this evaluation completed within the timeout for both heuristics. In panel (a), the shading of the points indicates the number of maximal additive pattern sets. In panel (b), it indicates the number of patterns. The figure confirms that the computation time for the canonical heuristic is closely linked to the number of maximal additive pattern sets, while the computation time for the post-hoc optimization heuristic is more closely linked to the number of patterns.

Optimal cost partitioning. Independently of the pattern selection method, the optimal cost partitioning approach h^{OCP} of Katz and Domshlak (columns c, g, j and m in Table 1) is not competitive with the other heuristics because it is too costly to compute. With singleton patterns (column g) it achieves a total coverage of 438 tasks, which drops below 300 for any of the other pattern selection methods. For comparison, A* with the blind heuristic solves 506 tasks (column o).

Nevertheless, it is an interesting question how well-informed the post-hoc optimization heuristic is compared to this "holy grail" for the given abstractions. To answer this, we ran experiments with a timeout of 24 hours to determine the initial heuristic estimates of h^{OCP} for a wide range of tasks. Figure 4 shows the result for systematically generated pattern collections with maximal pattern size 2. To increase readability, the figure only includes instances with heuristic values smaller than 50.

While optimal cost partitioning optimizes individual operator costs for each abstraction, the post-hoc optimization heuristic may only *scale all* operator costs for each PDB. The results show that this restriction indeed limits the quality of the heuristic estimates. However, the experiment also confirms how expensive optimal cost partitioning is: for 206 of the 1396 tasks it runs out of memory while setting up the LP, and for 43 of the remaining 1190 tasks, computation of the initial state heuristic value fails to finish within 24 hours.

Pattern selection methods. The preceding discussion mostly focused on a comparison of the three combination methods for PDB heuristics (h^C, h^{PhO} and h^{OCP}). We now switch attention and discuss the different pattern selection methods in the context of each of the three combination methods. The picture that emerges is quite different for each combination method.

With optimal cost partitioning, computation time grows fast with the number and size of patterns, so it is not surprising that it performs best with the method that generates the smallest number of patterns (systematic patterns of size 1; column g).

The canonical heuristic also does not combine well with the larger systematically generated pattern sets (due to the high number of additive patterns as discussed above). Note that Sys^1 generates exactly the initial pattern set of the iPDB method. Comparing columns a and e shows that hill-climbing does not pay off on average, but this depends heavily on the domain. One problem is that the iPDB procedure often requires too much time. This indicates that better adaptive stopping criteria for the hill-climbing search could be helpful.

With the post-hoc optimization heuristic, the systematic pattern generation performs very well, with a sweet spot at a maximal pattern size of 2. Comparing this best systematic pattern generation (column i) to the iPDB pattern selection (column b) shows an impressive improvement of 41 instances.

State of the art. We conclude with a brief comparison to the state of the art in optimal planning. The LM-Cut heuristic [Helmert and Domshlak, 2009], which is based on the ideas of landmarks and delete relaxation, currently offers the best performance on the IPC benchmark suite. Its results (column p in Table 1) show that there is still a gap to the best-performing abstraction heuristics. However, comparing the previous state of the art of *abstraction heuristics* in column a to our best new configuration in column i (post-hoc optimization on the systematic size-2 pattern collection) shows an impressive improvement of 44 additionally solved tasks.

To show that further significant improvements are possible with better pattern selection strategies, column n evaluates the potential of h^{PhO} under the assumption that there is an oracle predicting the optimal value for the maximal pattern size within the range 1–4 for each domain (best parameters shown in parentheses). This leads to a total of 675 solved tasks, another improvement by 42. (The corresponding number for the canonical heuristic is 648, not shown in the table.)

6 Conclusion

In this work, we attempted to push the current limits of additive abstraction heuristics in optimal planning. From our perspective, the most interesting contribution is the post-hoc optimization heuristic, which explores the middle ground between the "trivial cost partitioning" of the canonical heuristic, which set the previous state of the art in abstraction-based planning systems, and the rich notions of cost partitioning pioneered by Katz and Domshlak. We believe that both the theoretical notions and the experimental results explored in this paper offer many opportunities for future investigations.

7 Acknowledgments

We thank the anonymous reviewers for their useful comments, which helped improve the paper. This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

[Culberson and Schaeffer, 1998] Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In Amedeo Cesta and Daniel Borrajo, editors, *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 13–24, Toledo, Spain, 2001.

[Felner *et al.*, 2004] Ariel Felner, Richard Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

[Haslum *et al.*, 2007] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 1007–1012. AAAI Press, 2007.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169. AAAI Press, 2009.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Katz and Domshlak, 2010] Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12–13):767–798, 2010.

[Nissim *et al.*, 2011] Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 1983–1990, 2011.

[Sievers *et al.*, 2012] Silvan Sievers, Manuela Ortlieb, and Malte Helmert. Efficient implementation of pattern database heuristics for classical planning. In Daniel Borrajo, Ariel Felner, Richard Korf, Maxim Likhachev, Carlos Linares López, Wheeler Ruml, and Nathan Sturtevant, editors, *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SOCS 2012)*, pages 105–111. AAAI Press, 2012.

[Yang *et al.*, 2008] Fan Yang, Joseph Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32:631–662, 2008.