Pattern Databases for Goal-Probability Maximization in Probabilistic Planning

Thorsten Klößner [1], **Jörg Hoffmann** [1], **Marcel Steinmetz** [1], **Álvaro Torralba** [2]
[1]Saarland University, Saarland Informatics Campus, Germany
[2] Aalborg University, Department of Computer Science, Denmark
{kloessner, hoffmann, steinmetz}@cs.uni-saarland.de, alto@cs.aau.dk

Abstract

Heuristic search algorithms for goal-probability maximization (MaxProb) have been known since a decade. Yet prior work on heuristic functions for MaxProb relies on determinization, not actually taking the probabilities into account. Here we begin to fix this, by introducing MaxProb pattern databases (PDB). We show that, for the special case of PDBs in contrast to more general abstractions, abstract transitions have a unique probability so that the abstract planning task is still an MDP. The resulting heuristic functions are admissible, i.e., they upper-bound the real goal probability. We identify conditions allowing to admissibly multiply heuristic values across several PDBs. Our experiments show that even non-probabilistic PDB heuristics often outperform previous MaxProb heuristics, and that our new probabilistic PDBs can in turn yield significant performance gains over non-probabilistic ones.

Introduction

Heuristic search is a prominent approach to probabilistic planning (e. g. (Hansen and Zilberstein 2001; Bonet and Geffner 2003, 2006; Trevizan et al. 2017), in various forms. Here we address goal-probability maximization, *MaxProb*, the problem of finding an MDP policy maximizing the probability of reaching the goal. Heuristic search algorithms for MaxProb have been known since a decade (Kolobov et al. 2011). Yet the current MaxProb heuristic functions are lacking: they do not actually take the probabilities into account.

Heuristic functions that do take probabilities into account are known for probabilistic conformant planning (Bryce, Kambhampati, and Smith 2006; Domshlak and Hoffmann 2007); for finding a maximum-likelihood sequential plan, which can be compiled into a classical planning problem (Keyder and Geffner 2008; E-Martín, Rodríguez-Moreno, and Smith 2014); and, most closely related to our work, for MDP planning in the stochastic shortest path problem (SSP) setting where Trevizan et al. (2017) design the occupancy heuristics h^{pom} and h^{roc}. Yet for MaxProb, the current heuristic functions all rely on determinization and classical planning heuristics (Little and Thiébaux 2006; Steinmetz, Hoffmann, and Buffet 2016b,a; Trevizan, Teichteil-Königsbuch, and Thiébaux 2017; Klauck et al. 2020). This approach only employs dead-end detection: the heuristic function is

$$h^{\text{MaxProb}}(s) = \begin{cases} 0 & h(s) = \infty \\ 1 & \text{otherwise} \end{cases}$$

where h is the classical planning heuristic. Yet, states with goal probability between one and zero cannot be meaningfully estimated by such heuristics. Here we begin to address this deficiency, adapting pattern databases (PDB) to MaxProb planning.[1]

PDB heuristics in classical planning admissibly estimate (provide lower bounds on) cost-to-goal (Culberson and Schaeffer 1998; Edelkamp 2001). This is done through a particular type of abstraction, where the planning task is projected onto a subset P of its state variables (the *pattern*). This yields an abstract state space where states that agree on P are not distinguished. PDB heuristics (Haslum et al. 2007; Franco et al. 2017; Rovner, Sievers, and Helmert 2019) and other more general abstraction heuristics (Helmert et al. 2014; Seipp and Helmert 2018) are of paramount importance for effective cost-optimal classical planning.

Abstraction heuristics have, however, not yet been explored in probabilistic planning (except for the limited special case of single-variable projections employed as part of h^{pom} and h^{roc}). Part of the reason for this might be that, when the original state space is an MDP, an abstraction of that state space is not, in general, an MDP anymore: transitions between abstract states may not have a unique probability as transitions between different member states may have different probabilities. This complicates the computation of heuristic functions (e. g. through bounded-parameter MDPs (Givan, Leach, and Dean 2000)), and it leads to pathologies where refining the abstraction may result in worse heuristic estimates (Tagorti et al. 2013).

Here we introduce *MaxProb pattern databases (MaxProb PDB)*, as an abstraction method to admissibly estimate goal probability, i. e., provide an *upper bound* on the maximum probability to reach the goal. Like PDBs, MaxProb PDBs project the task onto a subset of state variables, so that we get a probabilistic abstraction. We show that, in contrast to more general abstractions as discussed above, this abstract

[1]We also adapt Trevizan et al.'s (2017) SSP heuristics h^{pom} and h^{roc} to MaxProb as part of our experiments, see below.

planning task is still an MDP. We show that MaxProb PDB heuristics are admissible, and that they do not suffer from the aforementioned refinement pathologies.

A crucial element of the success of PDB heuristics in classical planning is the ability to admissibly combine multiple patterns. A trivial means to accomplish this is taking the *maximum*, but often much stronger estimates can be obtained by taking the *sum*. The latter is admissibly possible under certain conditions, where the PDBs are *orthogonal* and thus *additive* (Felner, Korf, and Hanan 2004; Haslum et al. 2007).[2] Here we devise a similar methodology for MaxProb PDBs, where the corresponding operators are *minimum* and *product*. The minimum of two upper-bounding probabilities trivially remains an upper bound; for their product that is not so in general. We identify conditions under which a collection of MaxProb PDBs is *multiplicative*, i. e., the product of heuristic functions is admissible. Specifically, we identify three syntactic criteria, that we call *orthogonality*, *weak orthogonality* and *independence*, which provide sufficient conditions for multiplicativity.

For comparison, we also implement MaxProb versions of h^{pom} and h^{roc} (Trevizan, Thiébaux, and Haslum 2017), where the necessary changes are relatively straightforward. We furthermore implement non-probabilistic PDB heuristics based on determinization, to directly assess the impact of taking the probabilities into account. We run experiments on a large collection of MaxProb benchmarks. The results show that even the non-probabilistic PDBs often outperform previous MaxProb heuristics as well as our adaptations of h^{pom} and h^{roc}. Our new probabilistic PDBs dominate their non-probabilistic counterparts almost universally, and can yield additional significant performance gains.

Preliminaries

We represent planning tasks in probabilistic SAS$^+$ notation (Trevizan, Thiébaux, and Haslum 2017). A probabilistic SAS$^+$ task is a tuple $\Pi = \langle \mathbb{V}, \mathcal{A}, s_{\mathcal{I}}, \mathcal{G} \rangle$. $\mathbb{V}$ denotes the *variables*, each v has a finite domain $\mathcal{D}_v$. A *variable assignment* is a function $\sigma : \mathbb{V} \mapsto \bigcup_{v \in \mathbb{V}} \mathcal{D}_v \cup \{\bot\}$ with $\sigma(v) \in \mathcal{D}_v$, if defined. We write $\sigma(v) = \bot$ in case that v has not been assigned any value. We denote by $\mathbb{V}(\sigma)$ the set of all variables v with $\sigma(v) \neq \bot$. σ is *complete* if $\mathbb{V}(\sigma) = \mathbb{V}$. For a set of variables $U \subseteq \mathbb{V}(\sigma)$, we denote by $\sigma|_U$ the *projection of σ onto U*. Slightly abusing notation, we denote the *composition* of two variable assignments as $\sigma_2 \circ \sigma_1$ where $(\sigma_2 \circ \sigma_1)(v) = \sigma_2(v)$ for all $v \in \mathbb{V}(\sigma_1) \cap \mathbb{V}(\sigma_2)$, and $(\sigma_2 \circ \sigma_1)(v) = \sigma_1(v)$ for all $v \in \mathbb{V}(\sigma_1) \setminus \mathbb{V}(\sigma_2)$. The *initial state* $s_{\mathcal{I}}$ is a complete variable assignment. The *goal* $\mathcal{G}$ is a variable assignment. $\mathcal{A}$ is the set of *actions*. An action a specifies its *precondition* pre_a and a set of *effects* eff_a, all variable assignments, as well as a probability distribution P_a over the effects. The *states* $\mathcal{S}$ of Π are all complete variable assignments. An action a is *applicable* in state s if $pre_a(v) = s(v)$ for all $v \in \mathbb{V}(pre_a)$. $\mathcal{A}(s)$ denotes the set of all actions applicable in s.

[2] A more advanced method is to apply cost partitioning (Katz and Domshlak 2010; Pommerening et al. 2015; Seipp, Keller, and Helmert 2020).

Π induces the MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{T}, s_{\mathcal{I}}, \mathcal{S}_{\mathcal{G}} \rangle$ whose states and initial state are those of Π. The *goal states* $\mathcal{S}_{\mathcal{G}}$ are all states $s_{\mathcal{G}}$ where $s_{\mathcal{G}}(v) = \mathcal{G}(v)$ for all $v \in \mathbb{V}(\mathcal{G})$. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the *transition function*. For every state s and action $a \in \mathcal{A}(s)$, $T(s, a)$ yields a probability distribution over the possible successors states t where $\mathcal{T}(s, a, t) = \sum_{e \in eff_a, t=(e \circ s)} P_a(e)$. Whenever $\mathcal{T}(s, a, t) > 0$, we say that $\mathcal{M}$ has an a-transition from s to t. We say that a transition is *deterministic* if $\mathcal{T}(s, a, t) = 1$; otherwise it is *stochastic*. An action a is stochastic if there is a stochastic a-transition.

We are interested in *MaxProb* analysis. For simplicity, we assume that $\mathcal{A}(s) \neq \emptyset$ for all states s.[3] MaxProb is given by the piecewise minimal function $\mathcal{V}^* : \mathcal{S} \mapsto [0, 1]$ that satisfies the following equation

$$\mathcal{V}^*(s) = \begin{cases} 1 & \text{if } s \in \mathcal{S}_{\mathcal{G}} \\ \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}} \mathcal{T}(s, a, t) \mathcal{V}^*(t) & \text{else} \end{cases}$$

$\mathcal{V}^*(s_{\mathcal{I}})$ can be computed in many different ways. Relevant for this paper are in particular *value iteration (VI)* (Puterman 1994) and *heuristic search* (Kolobov et al. 2011; Steinmetz, Hoffmann, and Buffet 2016a). The details of MaxProb heuristic search are not of interest here. It suffices to know that these algorithms assume an initial estimate of $\mathcal{V}^*$, the *heuristic*, $h : \mathcal{S} \mapsto [0, 1]$ as input. They guarantee to find $\mathcal{V}^*(s_{\mathcal{I}})$ upon termination if h is *admissible*, i. e., optimistically bounds $\mathcal{V}^*$, $h(s) \geq \mathcal{V}^*(s)$ for all states s.

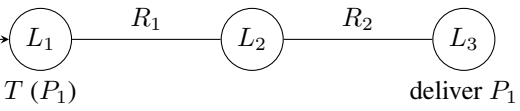
VI will play an important role to automatically derive admissible heuristics. In a nutshell, VI computes a sequence of value functions $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_n, \mathcal{V}_{n+1}$ until convergence, i.e., $\mathcal{V}_n = \mathcal{V}_{n+1}$. At this point, it is guaranteed that $\mathcal{V}_n = \mathcal{V}^*$. The sequence starts with $\mathcal{V}_0(s_{\mathcal{G}}) = 1$ for all goal states $s_{\mathcal{G}} \in \mathcal{S}_{\mathcal{G}}$, and $\mathcal{V}_0(s) = 0$ for all others. At each iteration i, $\mathcal{V}_i(s_{\mathcal{G}})$ remains 1 for goal states, and $\mathcal{V}_i(s) = \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}} \mathcal{T}(s, a, t) \mathcal{V}_{i-1}(t)$ for the others.

As running example, we consider a variant of logistics, depicted in Figure 1, where roads may become blocked. A *truck* (T) moves over three *locations* L_1, L_2, L_3 connected by *roads* R_1, R_2. A *package* (P) must be delivered to L_3. The roads can be blocked, making them impassable. Initially, each road has an unknown status. Trying to use a road (`try_drive`) has a probabilistic effect setting the (then immutable) status of the road. The road becomes clear with a probability of 80%, and becomes blocked with a probability of 20%. The agent can use (`drive`) on clear roads. The package can be delivered using `drop`. To keep the example simple, the package cannot be picked up. Obviously, the maximum goal probability is $0.8 * 0.8 = 0.64$.

MaxProb Pattern Databases

We first define what a projection of the induced MDP $\mathcal{M}$ is. Assuming a subset $V \subseteq \mathbb{V}$ of variables, we define the state set $\mathcal{S}_V$ as the set of complete variable assignments to V only, like in classical planning. The set of goal states is defined by $\mathcal{S}_{\mathcal{G}_V} := \{s_{\mathcal{G}}|_V \mid s_{\mathcal{G}} \in \mathcal{S}_{\mathcal{G}}\}$. We define $\mathcal{T}_V(\sigma, a, \tau) = 0$ if

[3] This can easily be achieved by introducing an artificial self-loop action.



$$s_{\mathcal{I}} = \{P \mapsto T, T \mapsto L_1, R_{1,2} \mapsto \mathsf{unknown}\}$$
$$\mathcal{G} = \{P \mapsto L_3\}$$

drive(l_x, r, l_y) :
$pre : \{T \mapsto l_x, r \mapsto \mathsf{clear}\}$
$P : \{T \mapsto l_y\} \mapsto 100\%$

drop(l, p) :
$pre : \{T \mapsto l, p \mapsto T\}$
$P : \{p \mapsto l\} \mapsto 100\%$

try_drive(l_x, r, l_y) :
$pre : \{T \mapsto l_x, r \mapsto \mathsf{unknown}\}$
$P : \{r \mapsto \mathsf{blocked}\} \mapsto 20\%, \{r \mapsto \mathsf{clear}, T \mapsto l_y\} \mapsto 80\%$

Figure 1: Logistics-style running example.

$pre_a|_V \not\subseteq \sigma$. Otherwise, we choose any representative $s \in \mathcal{S}$ with $\sigma = s|_V$ and $a \in \mathcal{A}(s)$ and define $\mathcal{T}_V(\sigma, a, \tau) = \sum_{t \in \mathcal{S}.\ \tau = t|_V} \mathcal{T}(s, a, t)$. This transition function $\mathcal{T}_V$ is well-defined, the transition probability is the same for any chosen representative. To see this, observe that we can rewrite the term used on the right hand side to a term independent of s:

$$\sum_{t.\tau = t|_V} \mathcal{T}(s, a, t) = \sum_{t.\tau = t|_V} \sum_{\substack{e \in \mathit{eff}(a) \\ t = e \circ s}} P_a(e) = \sum_{\substack{e \in \mathit{eff}(a) \\ \tau = e|_V \circ \sigma}} P_a(e)$$

where in the last step, we substitute $t|_V$ with $(e \circ s)|_V = e|_V \circ \sigma$. The transition function can thus also be stated as

$$\mathcal{T}_V(\sigma, a, \tau) = \begin{cases} \sum_{e \in \mathit{eff}_a . \tau = e|_V \circ \sigma} P_a(e) & \text{if } pre_a|_V \subseteq \sigma \\ 0 & \text{else} \end{cases}$$

As advertised in the introduction, the projected task is hence still an MDP in contrast to more general abstractions.

We say that the MDP $\mathcal{M}_V = \langle \mathcal{S}_V, \mathcal{A}, \mathcal{T}_V, \mathcal{S}_{\mathcal{G}V} \rangle$ is the *projection* of $\mathcal{M}$ onto V. We henceforth denote with $\mathcal{V}^*_V$ and $\mathcal{V}^V_i$ the MaxProb value function $\mathcal{V}^*$ of $\mathcal{M}_V$, and the ith VI value function $\mathcal{V}_i$ of $\mathcal{M}_V$ respectively. We define the MaxProb PDB heuristic for pattern V as $h^V(s) := \mathcal{V}^*_V(s|_V)$. A fundamental observation is that h^V is indeed an upper-bounding MaxProb-heuristic for any pattern $V \subseteq \mathbb{V}$:

Theorem 1 *h^V is admissible for any $V \subseteq \mathbb{V}$.*

Proof (sketch). Show $\mathcal{V}_n(s) \leq \mathcal{V}^V_n(s|_V)$ by induction on n. The only non-trivial case is the induction step with $s \notin \mathcal{S}_{\mathcal{G}}$:

$$\begin{aligned} \mathcal{V}_{n+1}(s) &= \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}} \mathcal{T}(s, a, t) \mathcal{V}_n(t) \\ &\leq \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}} \mathcal{T}(s, a, t) \mathcal{V}^V_n(t|_V) \\ &= \max_{a \in \mathcal{A}(s)} \sum_{\tau \in \mathcal{S}_V} \sum_{t \in \mathcal{S}. t|_V = \tau} \mathcal{T}(s, a, t) \mathcal{V}^V_n(\tau) \\ &= \max_{a \in \mathcal{A}(s)} \sum_{\tau \in \mathcal{S}_V} \mathcal{T}_V(s|_V, a, \tau) \mathcal{V}^V_n(\tau) \\ &\leq \max_{a \in \mathcal{A}(s|_V)} \cdots = \mathcal{V}^V_{n+1}(s|_V) \end{aligned}$$

□

Consider the example of Figure 1 and the pattern $P = \{T, P, R_1\}$. A determinized PDB finds out that all states where $R_1 = \mathit{clear}$ have the $\mathcal{V}^*$ value 1, and all states where $R_1 = \mathit{blocked}$ have $\mathcal{V}^*$ value 0. On the cases where $R_1 = \mathit{unclear}$, however, a determinized PDB returns 1 while the MaxProb PDB provides the more informative estimate 0.8.

Like in classical planning, it is possible to characterize the projected MDP $\mathcal{M}_V$ syntactically at the level of the input task Π, and therewith provide a means to compute $\mathcal{M}_V$ without computing the full state space, i. e., the induced $\mathcal{M}$, beforehand. Namely, we define the syntactic projection Π^V of Π to have variables V, preconditions $pre^V_a := pre_a|_V$, and effect probabilities $P^V_a(\epsilon) := \sum_{e \in \mathit{eff}_a . e|_V = \epsilon} P_a(e)$.

Theorem 2 *The MDP induced by Π^V is identical to $\mathcal{M}_V$.*

Proof (sketch). Let $\mathcal{T}'$ be the transition function of this MDP. We show $\mathcal{T}' = \mathcal{T}_V$. If $pre_a|_V = pre^V_a \not\subseteq \sigma$ then $\mathcal{T}'(\sigma, a, \tau) = 0$. Otherwise if $pre_a|_V = pre^V_a \subseteq \sigma$:

$$\mathcal{T}'(\sigma, a, \tau) = \sum_{\substack{\epsilon \in \mathit{eff}^V_a \\ \tau = \epsilon \circ \sigma}} \sum_{\substack{e \in \mathit{eff}_a \\ e|_V = \epsilon}} P_a(e) = \sum_{\substack{e \in \mathit{eff}_a \\ \tau = e|_V \circ \sigma}} P_a(e)$$

□

Hence, to build a MaxProb PDB for V, we can construct $\mathcal{M}_V$ from the syntactic projection Π^V of the planning task. Afterwards, we compute the maximal goal probability $\mathcal{V}^*_V$ of each abstract state $\sigma \in \mathcal{S}_V$, and store the outcome values in a lookup table. The MaxProb PDB heuristic h^V for s is computed by looking up the value of $s|_V$ in that table.

By Theorem 1, h^V is admissible for arbitrary V. Given a collection $\mathcal{C}$ of patterns, we can obviously define an admissible heuristic as $h^{\mathcal{C}}_{\min}(s) := \min_{V \in \mathcal{C}} h^V(s)$.

In contrast to the aforementioned pathologies of general abstractions where refinements can lead to worse heuristic estimates (Tagorti et al. 2013), larger patterns will always result in tighter goal probability bounds: if $V \subseteq W$ then $\mathcal{V}^* \leq h^W \leq h^V$. This is simply because $\mathcal{M}_V$ can be cast as a projection of $\mathcal{M}_W$, showing $h^W \leq h^V$ by admissibility.

Multiplicative MaxProb PDBs

Inspired by additive PDBs, we now investigate whether individual MaxProb PDB heuristics can be combined in better ways than taking the minimum. In the classical setting, this can be done if the participating patterns are orthogonal (Felner, Korf, and Hanan 2004; Haslum et al. 2007) or via cost-partitioning (Katz and Domshlak 2010; Pommerening et al. 2015; Seipp, Keller, and Helmert 2020), and allows to sum over individual PDB heuristics. In the MaxProb setting, this translates to *multiplicative* patterns.

Given a pattern collection $\mathcal{C}$, we define the multiplicative heuristic $h^{\mathcal{C}}_{\mathrm{mul}}(s) := \prod_{V \in \mathcal{C}} h^V(s)$. In what follows we devise three conditions on $\mathcal{C}$ sufficient to ensure admissibility of the multiplicative heuristic.

Orthogonality

First, we consider classical orthogonality and cast it to a multiplicativity criterion for MaxProb PDBs. We say that an action $a \in \mathcal{A}$ *affects* a pattern $V \subseteq \mathbb{V}$, iff $\mathcal{M}_V$ contains non-loop a-transitions, i.e. iff $\mathcal{T}_V(\sigma, a, \tau) > 0$ for some $\sigma \neq \tau$. Similarly to the classical case, this property can be checked syntactically on the planning task: a affects V if and only if there is an effect e with $P_a(e) > 0$ which assigns a variable $v \in V$ to a value different from $pre_a(v)$.

The classical notion of orthogonality requires that no action affects more than one pattern. For MaxProb PDBs, it is sufficient to consider *stochastic* actions, i. e. actions with two distinct possible effects.

Definition 1 (Orthogonality) *Let $\mathcal{C}$ be a pattern collection. Then $\mathcal{C}$ is called* orthogonal *if every stochastic action in $\mathcal{A}$ affects at most one $V \in \mathcal{C}$.*

Note that we still allow deterministic actions to affect multiple patterns. Therefore, contrary to additive pattern databases, orthogonality of multiplicative MaxProb PDBs does not imply disjointness. In our case, two orthogonal patterns may contain the same variable if actions affecting that variable are deterministic.

Theorem 3 *If $\mathcal{C}$ is orthogonal then $h^{\mathcal{C}}_{\mathrm{mul}}$ is admissible.*

We omit the proof since this theorem is implied by admissibility of a more permissive notion (proved in Theorem 4 below). Namely, one can relax orthogonality further by identifying stochastic actions that behave as if they were deterministic. There are two possible reasons for this. First, we say that a stochastic action is *pseudo-deterministic* if, for all s, there exists only a single $t \neq s$ such that $\mathcal{T}(s, a, t) > 0$; otherwise, we say that a is *fully stochastic*. For MaxProb, pseudo-deterministic actions are effectively deterministic because one can reach t from s with probability 1 by repeatedly applying a. Second, an action may be fully stochastic on the original task but not within the PDB. Given a variable subset V, we say that a is *fully stochastic on* V iff there are two effects e, e' of a such that $pre_a|_{V'}, e|_{V'}$ and $e'|_{V'}$ are pairwise distinct, where $V' = V \cap (\mathbb{V}(e) \cup \mathbb{V}(e'))$.

Definition 2 (Weak Orthogonality) *Let $\mathcal{C}$ be a pattern collection. Then $\mathcal{C}$ is called* weakly orthogonal *if every action in $\mathcal{A}$ is fully-stochastic on at most one $V \in \mathcal{C}$.*

The benefits of weak orthogonality can be observed in our running example. Consider the patterns $P_1 = \{T, P, R_1\}$ and $P_2 = \{T, P, R_2\}$. Under each of these, the initial state has goal probability 0.8. The two patterns are not orthogonal because the stochastic $try\_drive(l_x, r, l_y)$ actions affect both of them through their effect on the truck. However, all actions of the form $try\_drive(l_x, R_2, l_y)$ are pseudo-deterministic in P_1 because their only effect is $T \mapsto l_y$; and similarly all actions of the form $try\_drive(l_x, R_1, l_y)$ are pseudo-deterministic in P_2. Therefore P_1 and P_2 are weakly orthogonal, allowing us to admissibly derive the correct goal probability $0.8 * 0.8 = 0.64$ with the multiplicative heuristic.

Theorem 4 *If $\mathcal{C}$ is weakly orthogonal, $h^{\mathcal{C}}_{\mathrm{mul}}$ is admissible.*

Proof. W.l.o.g. we assume that pseudo-deterministic actions do not exist in any projection. This is justified by converting these actions to deterministic actions selecting the non-loop successor immediately with probability 1 without changing the optimal value function. Hence, every action is stochastic on at most one pattern, and deterministic on all others by weak orthogonality.

We show $\mathcal{V}_n(s) \leq \prod_{V \in \mathcal{C}} \mathcal{V}^V_n(s|_V)$ by induction on n. Only the case $s \notin \mathcal{S}_\mathcal{G}$ in the induction step is non-trivial:

$$\mathcal{V}_{n+1}(s) = \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}} \mathcal{T}(s, a, t) \mathcal{V}_n(t)$$

Now, let $\bar{a} \in \mathcal{A}(s)$ be an action maximizing the right hand side term. By weak orthogonality, we have that $\bar{a}$ is stochastic on at most one pattern $W \in \mathcal{C}$. By applying the induction hypothesis

$$\mathcal{V}_{n+1}(s) \leq \sum_{t \in \mathcal{S}} \mathcal{T}(s, \bar{a}, t) \prod_{V \in \mathcal{C}} \mathcal{V}^V_n(t|_V)$$

We only need to consider t such that $\mathcal{T}(s, \bar{a}, t) > 0$ in this sum. For such t we also have $\mathcal{T}_V(s|_V, \bar{a}, t|_V) > 0$ and since $\bar{a}$ is deterministic on $V \neq W$, we conclude that $t|_V$ must be the unique a-successor of $s|_V$ in $\mathcal{M}_V$ for $V \neq W$. It follows that $\mathcal{V}^V_n(t|_V) \leq \mathcal{V}^V_{n+1}(s|_V)$ for $V \neq W$, hence

$$\mathcal{V}_{n+1}(s) \leq \sum_{t \in \mathcal{S}} \mathcal{T}(s, \bar{a}, t) \mathcal{V}^W_n(t|_W) \prod_{V \in \mathcal{C} \setminus \{W\}} \mathcal{V}^V_{n+1}(s|_V)$$

Lastly, the term $\sum_{t \in \mathcal{S}} \mathcal{T}(s, \bar{a}, t) \mathcal{V}^W_n(t|_W)$ is bounded by:

$$\begin{aligned}
\ldots &= \sum_{\tau \in \mathcal{S}_W} \sum_{t \in \mathcal{S}. t|_W = \tau} \mathcal{T}(s, \bar{a}, t) \mathcal{V}^W_n(\tau) \\
&= \sum_{\tau \in \mathcal{S}_W} \mathcal{T}_W(s|_W, \bar{a}, \tau) \mathcal{V}^W_n(\tau) \\
&\leq \max_{a \in \mathcal{A}(s|_W)} \sum_{\tau \in \mathcal{S}_W} \mathcal{T}_W(s|_W, a, \tau) \mathcal{V}^W_n(\tau) \\
&= \mathcal{V}^W_{n+1}(s|_W)
\end{aligned}$$

□

Observe finally that, similarly as in the classical case, the multiplicative heuristic is dominated by the heuristic derived from the union of the underlying patterns. We henceforth denote $U := \bigcup_{V \in \mathcal{C}} V$.

Corollary 1 *If $\mathcal{C}$ is weakly orthogonal, $h^U \leq h^{\mathcal{C}}_{\mathrm{mul}}$.*

This follows directly from Theorem 4 because we can perceive $\mathcal{C}$ as a collection of projections on U.

We utilize orthogonality by finding the maximal (weakly) orthogonal subcollections of a generated pattern collection. The max-clique approach is analogous to the one for classical planning (Haslum et al. 2007) and remains sound. Building the compatibility graph with respect to (weak) orthogonality is reduced to inspection of the syntactic projections as outline above. The resulting admissible heuristic is called the *canonical (weak) orthogonality heuristic*.

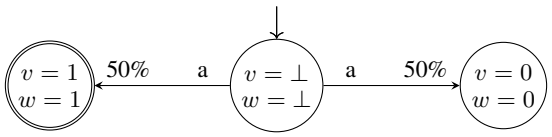


Figure 2: $\{v\}$ and $\{w\}$ are disjoint but not independent.

Definition 3 *Let $\mathcal{C}$ be a pattern collection and let Q be the set of max-cliques in the compatibility graph of $\mathcal{C}$ with respect to (weak) orthogonality. The canonical (weak) orthogonality heuristic $h^{\mathcal{C}}_{(\mathrm{w})\mathrm{orth}}$ is defined as*

$$h^{\mathcal{C}}_{(\mathrm{w})\mathrm{orth}}(s) := \min_{D \in Q} h^D_{\mathrm{mul}}(s) = \min_{D \in Q} \prod_{V \in D} h^V(s)$$

Independence

Lastly, we identify a different form of multiplicativity that arises from probabilistic independence, namely when the probability of an effect on $V \cup W$ equals the product of the individual sub-effect probabilities on each of V and W. Since a variable always depends on itself, we want V and W to be disjoint. We call $\mathcal{C}$ *disjoint* if all its patterns are pairwise disjoint. In what follows, denote again $U := \bigcup_{V \in \mathcal{C}} V$.

Definition 4 (Independence) *Let $\mathcal{C}$ be a pattern collection. Then $\mathcal{C}$ is called* independent *if $\mathcal{C}$ is disjoint and $\mathcal{T}_U(\sigma, a, \tau) = \prod_{V \in \mathcal{C}} \mathcal{T}_V(\sigma|_V, a, \tau|_V)$.*

Disjointness on its own is not enough for $h^{\mathcal{C}}_{\mathrm{mul}}$ to be admissible. Figure 2 gives a simple counter example. The goal probability is 50%, but the goal probability in each pattern is also 50% leading to the inadmissible product 25%.

When $\mathcal{C}$ is independent, this cannot happen. We prove this by showing that $h^{\mathcal{C}}_{\mathrm{mul}}$ is dominated by h^U, so that admissibility of $h^{\mathcal{C}}_{\mathrm{mul}}$ follows from admissibility of h^U.

Theorem 5 *If $\mathcal{C}$ is independent, $h^U \leq h^{\mathcal{C}}_{\mathrm{mul}}$.*

Proof (sketch). Show $\mathcal{V}^U_n(s) \leq \prod_{i=1}^k \mathcal{V}^{V_i}_n(s|_{V_i})$ by induction on n. The only non-trivial case is the induction step where $s \notin \mathcal{S}_{\mathcal{G}}$. Here we have

$$\begin{aligned}\mathcal{V}^U_{n+1}(s) &= \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}_U} \mathcal{T}_U(s, a, t) \mathcal{V}^U_n(t) \\ &\leq \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}_U} \prod_{1 \leq i \leq k} \mathcal{T}_{V_i}(s|_{V_i}, a, t|_{V_i}) \mathcal{V}^{V_i}_n(t|_{V_i})\end{aligned}$$

by independence and induction hypothesis. Furthermore, a state $t \in S_U$ can be split into the k substates $\langle t_1, \ldots, t_k \rangle \in S_{V_1}, \ldots, S_{V_k}$ where $t_i = t|_{V_i}$ because U is a disjoint union. Hence we may sum over these substates instead:

$$\cdots = \max_{a \in \mathcal{A}(s)} \sum_{\substack{\langle t_1, \ldots, t_k \rangle \in \\ \mathcal{S}_{V_1} \times \cdots \times \mathcal{S}_{V_k}}} \prod_{1 \leq i \leq k} \mathcal{T}_{V_i}(s|_{V_i}, a, t_i) \mathcal{V}^{V_i}_n(t_i)$$

Next, we use the following fact, which is easy to prove by induction over k. Let $f_i : \mathcal{S}_i \to \mathbb{R}$ be a family of functions.

$$\sum_{\substack{\langle t_1, \ldots, t_k \rangle \in \\ \mathcal{S}_1 \times \cdots \times \mathcal{S}_k}} \prod_{1 \leq i \leq k} f_i(t_i) = \prod_{1 \leq i \leq k} \sum_{t \in \mathcal{S}_i} f_i(t)$$

This allows us to swap the sum and the product, obtaining

$$\begin{aligned}\cdots &= \max_{a \in \mathcal{A}(s)} \prod_{1 \leq i \leq k} \sum_{t \in \mathcal{S}_{V_i}} \mathcal{T}_{V_i}(s|_{V_i}, a, t) \mathcal{V}^{V_i}_n(t) \\ &\leq \prod_{1 \leq i \leq k} \max_{a \in \mathcal{A}(s)} \sum_{t \in \mathcal{S}_{V_i}} \mathcal{T}_{V_i}(s|_{V_i}, a, t) \mathcal{V}^{V_i}_n(t) \\ &= \prod_{1 \leq i \leq k} \mathcal{V}^{V_i}_{n+1}(s|_{V_i})\end{aligned}$$

□

Corollary 2 *If $\mathcal{C}$ is independent, $h^{\mathcal{C}}_{\mathrm{mul}}$ is admissible.*

While these results are positive, it turns out that finding maximal independent pattern collections is not straightforward. There exist pattern collections for which all patterns are pairwise independent, but the collection itself is not. Therefore, maximal independence cannot be computed using a compatibility graph.

Theorem 6 *There exist non-independent pattern collections $\mathcal{C}$ where all pairs $P_i, P_j \in \mathcal{C}$ are pairwise independent.*

Proof. Consider the following planning task with three variables v_1, v_2, v_3, each with two values 0 and 1, and a single action a which has no precondition and the effects:

$$\begin{aligned}P_a([1,0,0]) &= \tfrac{1}{4} \qquad P_a([0,1,0]) = \tfrac{1}{4} \\ P_a([0,0,1]) &= \tfrac{1}{4} \qquad P_a([1,1,1]) = \tfrac{1}{4}\end{aligned}$$

To acknowledge that the patterns are pairwise independent, observe that for any two variables $v_i, v_j \in \mathbb{V}$, $i \neq j$:

$$\begin{aligned}\mathcal{T}_{\{v_i, v_j\}}([d_i, d_j], a, [d'_i, d'_j]) &= \tfrac{1}{4} \\ \mathcal{T}_{\{v_i\}}([d_i], a, [d'_i]) \cdot \mathcal{T}_{\{v_j\}}([d_j], a, [d'_j]) &= \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{1}{4}\end{aligned}$$

However, there are impossible transitions when considering all variables: In particular, $\mathcal{T}([0,0,0], a, [0,0,0]) = 0$ but $\prod_{1 \leq i \leq 3} \mathcal{T}_{\{v_i\}}([0], a, [0]) = \frac{1}{8}$, refuting independence. □

Given this, in our current implementation we consider orthogonality and weak orthogonality only. It remains an open question how to operationalize the notion of independence.

Relation between Multiplicativity Criteria

Trivially, orthogonality is a special case of weak orthogonality. To clarify the relation to independence, we observe:

Theorem 7 *If $\mathcal{C}$ is weakly orthogonal and disjoint, then $\mathcal{C}$ is independent.*

Proof (sketch). Define $U := \bigcup_{V \in \mathcal{C}} V$ and let $a \in A$ and $\sigma, \tau \in \mathcal{S}_U$. The equation to prove is:

$$\mathcal{T}_U(\sigma, a, \tau) = \prod_{V \in \mathcal{C}} \mathcal{T}_V(\sigma|_V, a, \tau|_V)$$

By weak orthogonality, a is stochastic on at most one $W \in \mathcal{C}$. That means for $V \neq W$, there is only one a-successor τ_V and $\mathcal{T}_V(\sigma|_V, a, \tau_V) = 1$. If $\tau|_V \neq \tau_V$ for some $V \neq W$ then both sides are zero. If $\tau|_V = \tau_V$, it remains to show that $\mathcal{T}_U(\sigma, a, \tau) = \mathcal{T}_W(\sigma|_V, a, \tau|_V)$. We have

$$\mathcal{T}_U(\sigma, a, \tau) = \sum_{e \in eff_a, \tau|_U = e|_U \circ \sigma|_U} P_a(e)$$

Since a is deterministic on $V \neq W$, every effect e already satisfies $\tau|_V = \tau_V = e|_V \circ \sigma|_V$ for $V \neq W$. Therefore:

$$\cdots = \sum_{e \in eff_a, \tau|_W = e|_W \circ \sigma|_W} P_a(e) = \mathcal{T}_W(\sigma|_W, a, \tau|_W) \quad \square$$

Without disjointness, independence and weak orthogonality are incomparable notions. In the example underlying the proof of Theorem 6, every variable pair is independent but not weakly orthogonal. Vice versa, in our running example the patterns $P_1 = \{T, P, R_1\}$ and $P_2 = \{T, P, R_2\}$ are weakly orthogonal but not independent.

Note that Theorem 7 is of a purely theoretical interest. One can use it to identify independent pattern collections $\mathcal{C}$ in practice, but these are weakly orthogonal anyhow and so nothing is gained over the use of that latter notion.

Experiments

We run experiments on a large collection of MaxProb benchmarks, evaluating a range of competing algorithms. We first describe the latter, then the benchmark collection, then summarize the results. All our source code, benchmarks, and results are publicly available (Klößner et al. 2021).

Competing Algorithms

For heuristic search on MaxProb, an important distinction is whether or not the MDP contains cycles. If not, then SSP algorithms like LRTDP (Bonet and Geffner 2003) can be used, and LAO* (Hansen and Zilberstein 2001) simplifies to AO*. If there are cycles, then the FRET outer loop (Kolobov et al. 2011) must be used, iterating these algorithms until there are no more "traps" under the found policy. We hence distinguish acyclic vs. cyclic benchmarks in our experiments, and run the corresponding algorithms on each sub-class. All algorithms are run until the Bellman residual of the initial state becomes less than ϵ, where $\epsilon = 10^{-5}$. When selecting greedy actions, all algorithms stick to the previous choice if possible and choose an arbitrary greedy action otherwise.

Regarding heuristic functions, previous work experimented mostly with dead-end pruning heuristics using determinization and $h^{\max}$ (Bonet and Geffner 2001). To nevertheless compare against prior work on heuristic functions taking probabilities into account, we implemented MaxProb versions of h^{pom} and h^{roc} (Trevizan, Thiébaux, and Haslum 2017). In a nutshell, the adaptations to the underlying LP encodings consist in changing the optimization condition to maximize the flow reaching goal states, and allowing flow to sink at arbitrary states. We run these heuristics in (1) LAO*, LRTDP, and FRET like for the other heuristics, (2) their native i-dual (Trevizan et al. 2017) framework where an LP encoding of the MDP state space is incrementally built, and (3) i^2-dual (Trevizan, Thiébaux, and Haslum 2017) where that LP is conjoined with the LP encoding the heuristic function. We remark that h^{pom} and h^{roc} were designed for different kinds of MDPs and are not particularly well suited to MaxProb due to limited dead-end detection capabilities. The reader should keep this in mind when comparing across heuristics in the results below.

From our own results presented here, we run MaxProb PDB configurations which minimize over all patterns (*min*) as well as multiplicative variants leveraging orthogonality (*orth*) and weak orthogonality (*weak*). To obtain a direct evaluation of the benefit of taking the probabilities into account, we also run the exact same PDB heuristics using determinization (thus limiting them to dead-end pruning).

The design of intelligent methods for finding good pattern collections $\mathcal{C}$ is a complex and ongoing research challenge, which we plan to address in depth in future work. For now, we adopt previous work suggesting to systematically enumerate "interesting" patterns up to a given size limit K (Pommerening, Röger, and Helmert 2013). We experiment with $K \in \{2, 3, 4\}$. For MaxProb PDBs, we discard a projection if every abstract state has goal probability 1. We do the same for the determinized PDBs if no dead-end is found. The computed pattern collections are identical across all PDB configurations in our experiments.

Our implementation is based on the probabilistic extension of Fast Downward (Helmert 2006; Steinmetz, Hoffmann, and Buffet 2016a), using PPDDL (Younes et al. 2005) as the front-end language. All experiments were run using the Downward Lab toolkit (Seipp et al. 2017) on a cluster of Intel Xeon E5-2650 v3 processors @2.30GHz, with a time limit of 30 minutes and a memory limit of 4 GB. For configurations requiring an LP solver, we used Soplex 3.1.1.

Benchmarks

Our benchmark collection is based on that of Steinmetz, Hoffmann, and Buffet (2016a), which consists of (a) domains from IPPC 2006 and 2008, (b) these same domains but with a consumed budget, (c) probabilistic resource-constrained planning, RCP in short (canadian-nomystery-rcp, canadian-rovers-rcp, canadian-tpp-rcp), and (d) simulated penetration testing (coresec). The acyclic benchmarks encompass (b) and (c), where every action consumes a non-zero discrete amount of budget/resource (encoded into PPDDL); as well as (d) which is acyclic by nature. The cyclic benchmarks encompass (a) as well as variants of (b) without resource consumption. For domains (a) and (b) that appeared in multiple IPPC iterations, we use the instances of all editions. From (a) we excluded domains without unavoidable dead-ends (blocksworld, elevators, random, sysadmin, zenotravel), where the goal probability is universally 1.

In addition to these benchmarks, we created a new cyclic testbed with many unavoidable dead-ends. Such situations arise naturally in resource-constrained planning (RCP) when some but not all actions consume a resource, leading to cycles such as loading/unloading packages repeatedly at the same location. We created such benchmarks (canadian-nomystery-rcp-cyclic, canadian-rovers-rcp-cyclic, canadian-tpp-rcp-cyclic) by starting from the

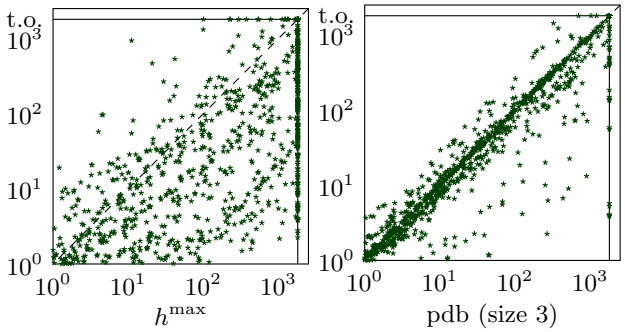


Figure 3: Total runtime (sec). MaxProb PDBs (y-axis) with weak orthogonality and pattern size 3 vs. $h^{\max}$ (x-axis, left) and determinized PDBs (x-axis, right).

RCP benchmarks (c) and limiting resource consumption to actions like driving which are the primary consumers of resources in the underlying applications.

Results

Table 1 shows coverage results, Table 2 shows the number of generated states. For acyclic planning, PDBs achieve highest coverage, beating all competing algorithms considerably. On the cyclic non-resource-constrained benchmarks, $h^{\max}$ performs best. However, PDBs outperform $h^{\max}$ in the cyclic resource-constrained setting. Without a budget/resource variable available, PDBs tend to require large patterns to derive impactful estimates, which becomes infeasible with the pattern generation we employ. Figure 3 (left) corroborates this picture, depicting the runtime of $h^{\max}$ vs. MaxProb PDBs with weak orthogonality.

The occupation measure heuristics h^{pom} and h^{roc} perform poorly across all of our benchmarks. In the MaxProb setting, the cost of repeatedly solving linear programs does not pay off. The encoded atomic projections (i. e. projections including a single variable) almost always induce a $\mathcal{V}^*$ estimate of 1 in the MaxProb setting, and the tying constraints rarely compensate for this. Like we said before, these results should not be over-interpreted as h^{pom} and h^{roc} were designed for different classes of MDPs.

Even determinized PDBs already achieve much of the above described advantage over competing techniques. Nevertheless, our MaxProb PDBs can achieve significantly better performance than determinized PDBs, especially on acyclic and cyclic-RCP domains. The number of generated states is reduced in about half of the domains. The advantage tends to increase with pattern size, which makes sense as larger patterns retain more probabilistic information. This sometimes pays off in coverage. Figure 3 (right) shows that, similarly, it can pay off in runtime, though we can also see the larger overhead of constructing MaxProb PDBs.

Evaluating lastly the impact of multiplicativity, we find that its benefits here are often marginal, but are significant in some domains, especially coresec and some benchmarks relating to resource-constrained planning. These results seem to be largely a function of how we select the patterns: on

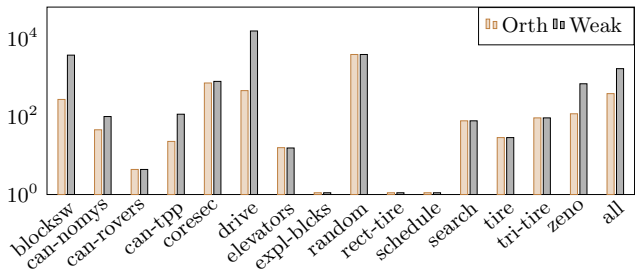


Figure 4: Average number of generated cliques, on acyclic benchmarks with pattern size 4.

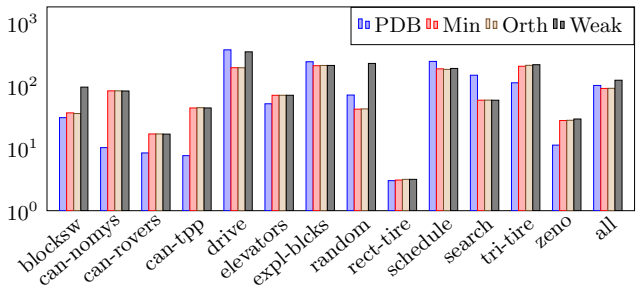


Figure 5: Average PDB construction time, on acyclic benchmarks with pattern size 4. Instances with time < 1 second are excluded (all instances of coresec and tireworld).

the one hand, each pattern is small, limiting the amount of information gained by multiplicativity; on the other hand, there are many patterns which can result in substantial effort for multiplicativity analysis. In the good cases, small patterns yield sufficient information to outweigh this overhead. In coresec, e. g., an attacker tries to compromise a system by executing exploits. Individual exploits can be captured in small patterns, and most exploits can be used independently. Multiplicativity across several patterns thus captures the joint success probability of combinations of exploits.

Figures 4 and 5 provide data elucidating the computational overhead of creating the PDB heuristics. This overhead is a combination of PDB construction effort plus, in some domains, substantial effort spent on multiplicativity-clique computations. Regarding the latter, note in Figure 4 that some domains reach well over 1000 generated cliques on average. This often results in timeouts.

Conclusion

We have introduced PDB heuristics for MaxProb planning, and shown that the basic concepts and properties from classical planning – admissibility and orthogonality – transfer naturally. The results show that significant empirical benefits are possible even when just systematically generating patterns of a fixed size. Our next step will be to investigate more intelligent pattern selection strategies. Interesting questions for the longer term concern PDB heuristics for stochastic shortest-path problems, as well as more general abstraction heuristics for probabilistic planning, beyond PDBs.

			AO* / FRET-π + LAO*																i-dual	
							Pattern Size $K=2$				Pattern Size $K=3$				Pattern Size $K=4$					
	Domain	N	blind	$h^{\max}$	h^{pom}	h^{roc}	pdb	min	orth	weak	pdb	min	orth	weak	pdb	min	orth	weak	h^{pom}	h^{roc}
acyclic	canadian-nomys-rcp	120	76	57	2	7	74	74	74	74	96	98	98	98	109	**112**	111	**112**	0	0
	canadian-rovers-rcp	120	104	100	47	64	103	103	103	103	103	103	103	103	**107**	106	106	106	21	21
	canadian-tpp-rcp	120	57	45	3	7	56	56	56	56	70	73	73	74	70	81	81	**84**	1	3
	coresec	30	12	12	14	14	12	14	16	16	12	14	**17**	**17**	12	14	**17**	**17**	8	8
	budget-blocks	180	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
	budget-drive	90	**90**	**90**	54	60	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	71	74	74	72	54	60
	budget-elevators	90	**81**	77	52	61	**81**	**81**	**81**	**81**	78	77	77	77	60	60	60	60	42	42
	budget-expl-blocks	150	68	91	58	63	**103**	**103**	**103**	**103**	99	101	101	101	89	89	89	89	54	58
	budget-random	72	42	42	**48**	**48**	43	43	43	43	43	44	44	44	30	31	31	16	37	45
	budget-rectangle-tire	36	**12**	**12**	11	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	11	**12**
	budget-schedule	138	**60**	**60**	56	**60**	**60**	**60**	**60**	**60**	**60**	**60**	**60**	**60**	54	59	59	59	53	56
	budget-search&rescue	90	66	57	37	42	66	66	66	66	72	72	72	72	74	**78**	73	74	28	28
	budget-tireworld	90	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**
	budget-triangle-tire	120	48	44	30	30	58	58	58	58	58	**61**	**61**	**61**	54	57	57	57	30	30
	budget-zenotravel	78	37	25	8	11	42	36	36	35	44	42	42	41	46	**47**	**47**	**47**	6	7
	Σ	1542	897	856	564	623	944	940	942	941	981	991	**994**	**994**	932	964	961	949	489	514
cyclic	canadian-rovers	10	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	8	8
	canadian-tpp	10	5	7	5	5	5	5	5	5	5	6	6	6	6	**9**	**9**	**9**	2	2
	drive	15	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
	exploding-blocks	30	9	**21**	9	9	17	17	17	17	17	17	17	17	16	17	17	17	8	8
	rectangle-tireworld	14	**14**	**14**	12	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	13	**14**
	schedule	30	5	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	8	8
	search&rescue	15	4	**5**	4	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	3	3
	tireworld	15	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
	triangle-tireworld	10	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	**6**	**6**
	Σ	174	81	**101**	84	86	95	95	95	95	95	96	96	96	95	99	99	99	79	78
cyc-rcp	can-nomys-rcp-cyclic	120	10	25	8	9	9	9	10	9	46	45	45	45	67	**72**	**72**	66	0	0
	can-rovers-rcp-cyclic	120	98	103	96	97	98	98	98	98	98	98	98	98	**106**	**106**	**106**	**106**	76	76
	can-tpp-rcp-cyclic	120	58	69	26	20	61	61	61	61	78	79	79	79	89	96	96	**101**	6	9
	Σ	360	166	197	130	126	168	168	169	168	222	222	222	222	262	**274**	**274**	273	82	85

Table 1: Coverage. Best results highlighted in boldface. Omitted results: i^2-dual, which performs much worse than i-dual; LRTDP, whose results are very similar to those for AO* and LAO*; domains with 0 coverage for all planners (budget-boxworld, boxworld, canadian-nomystery).

			Pattern Size $K=2$				Pattern Size $K=3$				Pattern Size $K=4$			
	Domain	Scale	pdb	min	orth	weak	pdb	min	orth	weak	pdb	min	orth	weak
acyclic	canadian-nomystery-rcp	10^4	**7529.0**	**7529.0**	**7529.0**	**7529.0**	1957.5	1672.3	1672.3	**1644.4**	1601.2	824.4	824.4	**796.4**
	canadian-rovers-rcp	10^4	**6025.7**	**6025.7**	**6025.7**	**6025.7**	**4898.8**	**4898.8**	**4898.8**	**4898.8**	1075.9	**1064.5**	**1064.5**	**1064.5**
	canadian-tpp-rcp	10^5	**927.0**	**927.0**	**927.0**	**927.0**	1021.5	572.7	572.7	**468.6**	598.0	159.0	159.0	**109.6**
	coresec	10^4	1998.4	404.9	**61.3**	**61.3**	1998.4	404.9	**61.3**	**61.3**	1998.4	404.9	**61.3**	**61.3**
	budget-blocksworld	10^3	2676.7	**2269.9**	**2269.9**	2429.5	1054.0	**598.2**	**598.2**	677.0	530.6	**113.9**	**113.9**	126.5
	budget-drive	10^3	8345.0	**7999.1**	**7999.1**	**7999.1**	**8039.1**	8390.9	8390.9	8343.3	**1428.2**	1608.0	1608.0	1578.0
	budget-elevators	10^4	**8202.0**	**8202.0**	**8202.0**	**8202.0**	**3805.5**	5955.0	5955.0	5955.0	**194.9**	1037.7	1037.7	1037.7
	budget-exploding-blocks	10^4	**1757.8**	**1757.8**	**1757.8**	**1757.8**	**316.4**	**316.4**	**316.4**	**316.4**	**73.2**	**73.2**	**73.2**	**73.2**
	budget-random	10^4	1058.5	1103.6	1103.6	**973.5**	82.47	36.95	36.95	**27.62**	2.65	**0.25**	**0.25**	**0.25**
	budget-rectangle-tireworld	10	**1134.2**	**1134.2**	**1134.2**	**1134.2**	**264.3**	**264.3**	**264.3**	**264.3**	264.3	**19.47**	**19.47**	**19.47**
	budget-schedule	100	**1283.4**	**1283.4**	**1283.4**	**1283.4**	**595.0**	**595.0**	**595.0**	**595.0**	**167.6**	**167.6**	**167.6**	**167.6**
	budget-search&rescue	10^4	**5413.5**	**5413.5**	**5413.5**	**5413.5**	5473.2	**4531.7**	**4531.7**	**4531.7**	2238.2	**360.7**	**360.7**	**360.7**
	budget-tireworld	1	**378.4**	**378.4**	**378.4**	**378.4**	201.1	**108.7**	**108.7**	**108.7**	164.1	**97.8**	**97.8**	**97.8**
	budget-triangle-tireworld	10^4	**2052.0**	**2052.0**	**2052.0**	**2052.0**	1356.5	**302.1**	**302.1**	**302.1**	304.1	**1.36**	**1.36**	**1.36**
	budget-zenotravel	10^5	**799.5**	2234.4	2234.4	2617.8	**739.2**	1895.0	1895.0	2225.0	1242.6	**950.6**	**950.6**	970.3
	All	10^4	**3494.5**	4342.9	4320.0	4568.0	**2560.8**	2913.4	2890.4	3038.8	1735.1	998.1	975.2	**953.4**
cyclic	canadian-rovers	10^4	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**	**4717.5**
	canadian-tpp	10^5	**4743.8**	**4743.8**	**4743.8**	**4743.8**	4736.8	**4260.2**	**4260.2**	**4260.2**	5594.3	3859.3	3859.3	**3826.3**
	drive	10	1736.8	**1588.2**	**1588.2**	**1588.2**	1736.8	**1523.6**	**1523.6**	**1523.6**	1736.7	**1490.1**	**1490.1**	**1490.1**
	exploding-blocks	10^4	**5111.4**	**5111.4**	**5111.4**	**5111.4**	**5111.4**	**5111.4**	**5111.4**	**5111.4**	**477.9**	**477.9**	**477.9**	**477.9**
	rectangle-tireworld	10^3	**1851.2**	**1851.2**	**1851.2**	**1851.2**	**1629.5**	**1629.5**	**1629.5**	**1629.5**	**1629.5**	**1629.5**	**1629.5**	**1629.5**
	schedule	10	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**	**6238.2**
	search&rescue	10^5	**3466.8**	**3466.8**	**3466.8**	**3466.8**	3466.8	**2672.4**	**2672.4**	**2672.4**	3466.8	**1334.1**	**1334.1**	**1334.1**
	tireworld	10^4	**6274.6**	**6274.6**	**6274.6**	**6274.6**	**4436.0**	**4436.0**	**4436.0**	**4436.0**	**3081.1**	3140.0	3140.0	3140.0
	triangle-tireworld	10^3	**5447.3**	**5447.3**	**5447.3**	**5447.3**	**4951.1**	**4951.1**	**4951.1**	**4951.1**	1233.7	**1231.8**	**1231.8**	**1231.8**
	All	10^4	**9999.7**	**9999.7**	**9999.7**	**9999.7**	9805.4	**8534.4**	**8534.4**	**8534.4**	9922.6	6060.8	6060.8	**6027.7**
cyc-rcp	can-nomystery-rcp-cyclic	10^5	**8648.2**	**8648.2**	**8648.2**	**8648.2**	3528.3	3448.7	3448.7	**3446.3**	2502.9	1870.1	1870.1	**1810.7**
	can-rovers-rcp-cyclic	10^4	**8551.5**	**8551.5**	**8551.5**	**8551.5**	**7506.6**	**7506.6**	**7506.6**	**7506.6**	6086.0	5942.0	5942.0	**5941.9**
	can-tpp-rcp-cyclic	10^5	**2740.9**	**2740.9**	**2740.9**	**2740.9**	1629.0	1537.8	1537.8	**1525.9**	1185.0	770.7	770.7	**725.3**
	All	10^5	**4081.4**	**4081.4**	**4081.4**	**4081.4**	1969.3	1912.4	1912.4	**1907.6**	1432.2	1078.3	1078.3	**1043.4**

Table 2: Number of generated states (average over commonly solved instances) for PDB approaches, separated by pattern size K as coverage often varies with K. Best results within each category highlighted in boldface. Scale gives a common multiplication factor for each row, to reduce table width.

Acknowledgments

This work was funded by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science). Special thanks to Bob Givan for his help in analyzing the IPPC Random domain.

References

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *AI* 129(1–2): 5–33.

Bonet, B.; and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proc. ICAPS'03*, 12–21.

Bonet, B.; and Geffner, H. 2006. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs. In *Proc. ICAPS'06*, 142–151.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Sequential Monte Carlo in probabilistic planning reachability heuristics. In *Proc. ICAPS'06*, 233–242.

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

Domshlak, C.; and Hoffmann, J. 2007. Probabilistic Planning via Heuristic Forward Search and Weighted Model Counting. *JAIR* 30: 565–620.

E-Martín, Y.; Rodríguez-Moreno, M. D.; and Smith, D. E. 2014. Progressive heuristic search for probabilistic planning based on interaction estimates. *Expert Systems - The Journal of Knowledge Engineering* 31(5): 421–436.

Edelkamp, S. 2001. Planning with Pattern Databases. In *Proc. ECP'01*, 13–24.

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

Franco, S.; Torralba, A.; Lelis, L. H.; and Barley, M. 2017. On Creating Complementary Pattern Databases. In *Proc. IJCAI'17*, 4302–4309.

Givan, R.; Leach, S. M.; and Dean, T. 2000. Bounded-parameter Markov decision processes. *AI* 122(1–2): 71–109.

Hansen, E. A.; and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *AI* 129(1-2): 35–62.

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'07*, 1007–1012.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR* 26: 191–246.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *JACM* 61(3): 16:1–16:63.

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

Keyder, E.; and Geffner, H. 2008. The HMDP Planner for Planning with Probabilities. In *IPC 2008 planner abstracts*.

Klauck, M.; Steinmetz, M.; Hoffmann, J.; and Hermanns, H. 2020. Bridging the Gap Between Probabilistic Model Checking and Probabilistic Planning: Survey, Compilations, and Empirical Comparison. *JAIR* 68: 247–310.

Klößner, T.; Hoffmann, J.; Steinmetz, M.; and Torralba, Á. 2021. Code and Benchmarks of the ICAPS'21 submission "Pattern Databases for Goal-Probability Maximization in Probabilistic Planning". https://doi.org/10.5281/zenodo.4604720.

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H. 2011. Heuristic Search for Generalized Stochastic Shortest Path MDPs. In *Proc. ICAPS'11*.

Little, I.; and Thiébaux, S. 2006. Concurrent Probabilistic Planning in the Graphplan Framework. In *Proc. ICAPS'06*, 263–273.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proc. AAAI'15*, 3335–3341.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *Proc. IJCAI'13*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley. ISBN 978-0-47161977-2. doi:10.1002/9780470316887.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In *Proc. ICAPS'19*, 362–367.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR* 62: 535–577.

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

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

Steinmetz, M.; Hoffmann, J.; and Buffet, O. 2016a. Goal Probability Analysis in MDP Probabilistic Planning: Exploring and Enhancing the State of the Art. *JAIR* 57: 229–271.

Steinmetz, M.; Hoffmann, J.; and Buffet, O. 2016b. Revisiting Goal Probability Analysis in Probabilistic Planning. In *Proc. ICAPS'16*.

Tagorti, M.; Scherrer, B.; Buffet, O.; and Hoffmann, J. 2013. Abstraction Pathologies in Markov Decision Processes. In *Proceedings of the 8th Journées Francophones Planification, Décision, et Apprentissage (JFPDA-13)*.

Trevizan, F. W.; Teichteil-Königsbuch, F.; and Thiébaux, S. 2017. Efficient solutions for Stochastic Shortest Path Problems with Dead Ends. In Elidan, G.; Kersting, K.; and Ihler, A. T., eds., *Proc. the 33rd Conference on Uncertainty in Artificial Intelligence (UAI'17)*. AUAI Press.

Trevizan, F. W.; Thiébaux, S.; and Haslum, P. 2017. Occupation Measure Heuristics for Probabilistic Planning. In *Proc. ICAPS'17*, 306–315.

Trevizan, F. W.; Thiébaux, S.; Santana, P. H.; and Williams, B. 2017. I-dual: Solving Constrained SSPs via Heuristic Search in the Dual Space. In *Proc. IJCAI'17*, 4954–4958.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The First Probabilistic Track of the International Planning Competition. *JAIR* 24: 851–887.