Cartesian Abstractions and Saturated Cost Partitioning in Probabilistic Planning

Thorsten Klößner[a], Jendrik Seipp[b] and Marcel Steinmetz[a]

[a]Saarland University, Saarbrücken, Germany
[b]Linköping University, Linköping, Sweden

Abstract. Stochastic shortest path problems (SSPs) capture probabilistic planning tasks with the objective of minimizing expected cost until reaching the goal. One of the strongest methods to solve SSPs optimally is heuristic search guided by an admissible (lower-bounding) heuristic function. Recently, probability-aware pattern database (PDB) abstractions have been highlighted as an efficient way of generating such lower bounds, with significant advantages over traditional determinization-based approaches. Here, we follow this work, yet consider a more general type, Cartesian abstractions, which have been used successfully in the classical setting. We show how to construct probability-aware Cartesian abstractions via a counterexample-guided abstraction refinement (CEGAR) loop akin to classical planning. This method is complete, meaning it guarantees convergence to the optimal expected cost if not terminated prematurely. Furthermore, we investigate the admissible combination of multiple such heuristics using saturated cost partitioning (SCP), marking its first application in the probabilistic setting. In our experiments, we show that probability-aware Cartesian abstractions yield much more informative heuristics than their determinization-based counterparts. Finally, we show that SCP yields probability-aware abstraction heuristics that are superior to the previous state of the art.

1 Introduction

Probabilistic planning deals with sequential decision making problems with stochastic actions, i.e., each action can lead to one out of possibly multiple outcomes with a fixed probability. We focus here on the setting of stochastic shortest path problems (SSPs) [3], where the goal is to find a policy that leads to some goal state with a probability of one, while minimizing the expected costs. Heuristic search algorithms (e.g., [9, 5, 33]) can be used to find such an optimal policy, provided they are used in conjunction with an admissible heuristic, i.e., one that provides lower-bounding estimates on the true expected cost to reach a goal state. The traditional approach to derive such heuristics is through the use of *all-outcome determinization*, which casts the probabilistic planning problem into a classical planning problem in which every possible action effect can be chosen freely (e.g., [6, 19]). By design, the major weakness of all-outcome determinization is that it ignores the probabilistic nature of the problem, which often leads to uninformed heuristics.

Recently, several *probability-aware* heuristics have been proposed to overcome this drawback. *Occupation measure heuristics* [32] are a generalization of operator-counting heuristics [23] for probabilistic planning. They estimate the expected number of times an action must be applied to reach a goal state and come in two variants. First, the *projection occupation measure heuristic* h^{pom} considers the *atomic projections* of the probabilistic planning task, which is given in a finite-domain representation [1]. An atomic projection is a probability-aware abstraction that considers a single state variable of the task. Second, the *regrouped operator counting heuristic* h^{roc} extends the state equation heuristic [4] over the determinization with so-called regrouping constraints, ensuring that the operator count of a deterministic operator is proportional to the probability of its inducing effect. Both heuristics are postulated to be equal and a recent analysis further substantiates this hypothesis [15].

Going beyond single-variable projections, probability-aware pattern database (PDB) heuristics [14] compute projections to arbitrary sets of variables, called *patterns*. Each PDB heuristic maintains a lookup table containing the true cost-to-goal value of each abstract state in the projection and a function that maps concrete states to abstract states. For a given concrete state, a PDB heuristic computes the corresponding abstract state and returns its cost-to-goal value. To compute a useful collection of PDBs, several construction techniques from classical planning have been adapted to the probabilistic setting [17], including counterexample-guided abstraction refinement (CEGAR, [24]), and local search using hill climbing [10].

The development of probability-aware PDB heuristics prompts the question of whether Cartesian abstractions [25, 27], a more general family of abstraction in classical planning, can be generalized in a similar way. We demonstrate how the classical construction method via CEGAR can be extended to the probabilistic setting and discuss various aspects of an efficient implementation for SSPs, including 1) the representation of the abstract state space, 2) the efficient discovery of optimal abstract policies within it, 3) the identification of flaws in such policies, and 4) the refinement of the abstraction to prevent re-occurrence of identified flaws. Moreover, we show how to construct multiple additive Cartesian abstraction heuristics using saturated cost partitioning [26, 28]. To the best of our knowledge, this method has not been previously considered in probabilistic planning.

In our experimental evaluation, we first consider single-abstraction approaches and demonstrate that probability-aware Cartesian abstraction is much more informative than its determinization-based counterpart. Our heuristics are also often more informed than PDB heuristics constructed via CEGAR, attributable to the more fine-grained possibilities for abstraction refinement offered by the framework. We also analyze the bottlenecks of the procedure, and discuss possible improvements. In the multi-abstraction setting, we show for various kinds of abstraction heuristics, that applying saturated cost

partitioning yields dramatic advantages over other previously considered admissible combination techniques. Overall, our results indicate that probability-aware Cartesian abstraction heuristics and saturated cost partitioning are state-of-the-art methods for deriving admissible heuristics in probabilistic planning.

2 Background

We write $\mathit{Dist}(X) := \{\delta : X \to [0,1] \mid \sum_{x \in X} \delta(x) = 1\}$ for the set of *probability distributions* over finite X. The *support* of $\delta \in \mathit{Dist}(X)$ is given by $\mathit{supp}(\delta) := \{x \in X \mid \delta(x) > 0\}$. For partially defined functions, we write $x \in f$ if f defines a value for x.

Probabilistic Transition Systems

A *probabilistic transition system* (PTS) is a tuple $\Theta = \langle S_\Theta, L_\Theta, C_\Theta, T_\Theta, G_\Theta \rangle$. S_Θ is a finite set of *states*, L_Θ is a finite set of *labels*, $C_\Theta : L_\Theta \to \mathbb{R}_0^+$ is a *cost function*, $T_\Theta \subseteq S_\Theta \times L_\Theta \times \mathit{Dist}(S_\Theta)$ is a finite *stochastic transition relation* and $G_\Theta \subseteq S_\Theta$ are the *goal states*. For a transition $\tau = \langle s, \ell, \delta \rangle \in T_\Theta$, we define $C_\Theta(\tau) := C_\Theta(\ell)$.

A *policy* for Θ is a partial function $\pi : S_\Theta \to T_\Theta$ with $\pi(s) = \langle s, \ell, \delta \rangle$, if $s \in \pi$. If $s \notin \pi$, we say that π terminates in s. The expected accumulated cost of π when starting in s is defined by $J^\pi_\Theta(s) := \mathbb{E}_{\pi,s}[\sum_{i=0}^{\mathsf{T}} C_\Theta(\pi(\mathsf{S}_i))]$, where S_i is the random variable for the state at time step i and $\mathsf{T} := \inf\{i \mid \mathsf{S}_{i+1} \notin \pi\}$. A policy *solves* s, iff the execution of π from s terminates in a goal state with certainty. The set of solutions for s is denoted $\mathit{Sols}_\Theta(s)$. The *optimal value function* of Θ is defined by $J^*_\Theta(s) := \inf_{\pi \in \mathit{Sols}_\Theta(s)} J^\pi_\Theta(s)$. A solution $\pi \in \mathit{Sols}_\Theta(s)$ is *optimal* for s if $J^\pi_\Theta(s) = J^*_\Theta(s)$. We consider a more general definition of J^*_Θ in Section 4.

Probabilistic Planning

We consider PTSs that are implicitly represented in an extended SAS$^+$ formalism [1]. A *variable space* is a finite, non-empty set $\mathcal{V}$ of state variables, where each $v \in \mathcal{V}$ is implicitly associated with its finite domain $\mathcal{D}(v)$. The sets $\mathcal{A}(V) := \bigtimes_{v \in V} \mathcal{D}(v)$ and $\mathcal{A}^p(V) := \bigcup_{W \subseteq V} \mathcal{A}(W)$ are the *complete and partial (variable) assignments* over the subset of variables $V \subseteq \mathcal{V}$. For $s \in \mathcal{A}^p(\mathcal{V})$, we use the notation $s[v]$ in place of $s(v)$. The set $\mathcal{C}(V) := \bigtimes_{v \in V} 2^{\mathcal{D}(v)}$ is the set of *Cartesian sets* of complete variable assignments over V. For $A \in \mathcal{C}(\mathcal{V})$, we write $\mathit{dom}(v, A)$ for the *abstract domain* $A(v) \subseteq \mathcal{D}(v)$ of variable v in A. Each assignment $s \in \mathcal{A}^p(\mathcal{V})$ can be seen as a Cartesian set $\mathit{Cart}(s) \in \mathcal{C}(\mathcal{V})$ with $\mathit{dom}(v, \mathit{Cart}(s)) := \{s[v]\}$ if $v \in s$ and $\mathit{dom}(v, \mathit{Cart}(s)) := \mathcal{D}(v)$ otherwise.

A *probabilistic planning task* is a tuple $\Pi = \langle \mathcal{V}_\Pi, \mathcal{O}_\Pi, C_\Pi, I_\Pi, G_\Pi \rangle$, where $\mathcal{V}_\Pi$ is a variable space and $\mathcal{O}_\Pi$ is a finite set of *operators*. Each $o \in \mathcal{O}_\Pi$ has a *precondition* $\mathit{pre}(o) \in \mathcal{A}^p(\mathcal{V}_\Pi)$, finitely many *effects* $\mathit{eff}_1(o), \ldots, \mathit{eff}_{\mathit{arity}(o)}(o) \in \mathcal{A}^p(\mathcal{V}_\Pi)$ and *effect probabilities* $\mathit{Pr}_1(o), \ldots, \mathit{Pr}_{\mathit{arity}(o)}(o) \in (0,1]$ (summing up to one). $C_\Pi : \mathcal{O}_\Pi \to \mathbb{R}_0^+$ is an operator cost function, $I_\Pi \in \mathcal{A}(\mathcal{V}_\Pi)$ is the initial state and $G_\Pi \in \mathcal{A}^p(\mathcal{V}_\Pi)$ is the goal.

A planning task Π induces the PTS $\Theta(\Pi) := \langle \mathcal{A}(\mathcal{V}_\Pi), \mathcal{O}_\Pi, C_\Pi, T', \mathit{Cart}(G_\Pi) \rangle$ with complete assignments as states and operators as labels. To this end, let the *updated assignment* $s[\![e]\!] \in \mathcal{A}^p(\mathcal{V})$ for $s, e \in \mathcal{A}^p(\mathcal{V})$ be defined as $s[\![e]\!][v] := e[v]$ if $v \in e$, $s[\![e]\!][v] := s[v]$ if $v \notin e$ and $v \in s$ and undefined otherwise. Applying an operator $o \in \mathcal{O}_\Pi$ in state $s \in \mathcal{A}(\mathcal{V})$ induces the successor distribution $s[\![o]\!][t] := \sum_{1 \le i \le \mathit{arity}(o). s[\![\mathit{eff}_i(o)]\!] = t} \mathit{Pr}_i(o)$. Finally, $T' := \{\langle s, o, s[\![o]\!] \rangle \mid o \in \mathcal{O}_\Pi, s \in \mathit{Cart}(\mathit{pre}(o))\}$. Our objective is to find a policy for $\Theta(\Pi)$ that is optimal for I_Π.

Algorithm 1 CEGAR refinement loop.

```
1: function CEGAR(Π)
2:     ⟨Θ_σ, σ⟩ ← TRIVIALABSTRACTION(Π)
3:     while not TERMINATIONCONDITION():
4:         π_σ ← FINDOPTIMALSOLUTION(Θ_σ, σ(I_Π))
5:         if not π_σ exists: return "Task unsolvable"
6:         φ ← FINDFLAW(Π, Θ_σ, π_σ)
7:         if not φ exists: return "Optimal solution found"
8:         ⟨Θ_σ, σ⟩ ← REFINE(Θ_σ, σ, φ)
9:     return Θ_σ
```

Heuristics & Probability-Aware Abstractions

A *heuristic* for Θ is a function $h : S_\Theta \to \mathbb{R}$ and h is *admissible* if $h(s) \le J^*_\Theta(s)$ for all states $s \in S_\Theta$. A widely deployed method to obtain an admissible heuristic for a task-induced PTS is applying classical planning heuristics on the *all-outcomes determinization* [12]. It compiles a probabilistic into a classical planning task by substituting every operator o with one deterministic operator o_e for each possible effect e, with the same precondition and cost as o, but the deterministic effect e. As this approach completely ignores the probabilities, the resulting heuristics are usually not very informative.

Probability-aware abstractions are a subclass of transformations on PTSs yielding admissible heuristics [18]. An abstraction of a PTS Θ is induced by a surjective abstraction mapping $\sigma : S_\Theta \to S'$, where S' is a set of *abstract states*. More precisely, let $\sigma[\delta] \in \mathit{Dist}(S')$ be the *abstract distribution* of $\delta \in \mathit{Dist}(S_\Theta)$ defined by $\sigma[\delta](s') := \sum_{s \in \sigma^{-1}(s')} \delta(s)$. Then the abstract PTS induced by σ is defined by $\sigma(\Theta) := \langle S', L_\Theta, C_\Theta, T', \sigma(G_\Theta) \rangle$, where $T' := \{\langle \sigma(s), \ell, \sigma[\delta] \rangle \mid \langle s, \ell, \delta \rangle \in T_\Theta\}$. An abstraction mapping σ for Θ induces the *abstraction heuristic* $h^\sigma(s) := J^*_{\sigma(\Theta)}(\sigma(s))$. As $\sigma(\Theta)$ forms a strong simulation of Θ, i.e., all solutions of Θ have an equivalent in $\sigma(\Theta)$, $h^\sigma(s)$ is guaranteed to be admissible.

Pattern database (PDB) heuristics [14] use this framework by choosing a subset of variables $P \subseteq \mathcal{V}$ of the given planning task, a so-called *pattern*, and define the abstraction function as $\sigma(s) := s|_P$, pruning all state variables not contained in P for a given state.

Cartesian Abstractions

Cartesian abstractions are more general than PDB heuristics. So far, they have only been considered for the classical setting within automated planning [27], but their definition can be straightforwardly extended to the probability-aware abstraction framework. Intuitively, an abstraction is Cartesian if each abstract state is a Cartesian set of complete variable assignments. Formally, let $A_1, A_2, \ldots, A_n \in \mathcal{C}(\mathcal{V})$ be Cartesian sets that form a partition of $\mathcal{A}(\mathcal{V})$, i.e., $\bigcup_{1 \le i \le n} A_i = \mathcal{A}(\mathcal{V})$. Such a partition induces the Cartesian abstraction with abstract states $S' = \{A_1, \ldots, A_n\}$ and corresponding abstraction mapping $\sigma(s) = A_i$ for each $s \in \mathcal{A}(\mathcal{V})$, where i is such that $s \in A_i$. As by assumption the Cartesian sets form a partition of the state set, this index i must exist and is unique.

The only method for constructing Cartesian abstractions suggested to date is counterexample-guided abstraction refinement (CEGAR) [7]. Algorithm 1 shows the high-level pseudo-code as per Seipp and Helmert [27]. While in their instantiation of Algorithm 1 solutions are *plans*, in our SSP variant of the algorithm solutions will be *policies*. In both cases, the construction loop starts with a coarse abstraction (TRIVIALABSTRACTION), typically the abstraction with the single abstract state $A_1 := \mathcal{A}(\mathcal{V})$. It then refines the abstraction

by extracting (FINDOPTIMALSOLUTION) and subsequently analyzing (FINDFLAW) an optimal *abstract* solution, looking for *flaws* that make the abstract solution fail for the original task. The found flaws are used to guide the refinement process (REFINE), *splitting* an abstract state A_i into two $A_i = A_{i,1} \cup A_{i,2}$ in a way ensuring that the same flaw cannot happen again in future iterations. After updating the abstract transition system, the process is started anew. The refinement loop stops when an optimal solution for the task is found, or a stopping criterion is met, such as hitting a time or memory limit.

3 Probability-Aware Cartesian CEGAR

We now generalize the classical CEGAR refinement loop sketched in Section 2 towards generating probability-aware Cartesian abstractions σ. Once built, h^σ can be precomputed by solving $J^*_{\Theta_\sigma}$ for all (reachable) abstract states, using e.g., topological value iteration [8]. In the remainder of this section, we discuss our adaptations of the three main procedures of the CEGAR algorithm: FINDOPTIMALSOLUTION, FINDFLAW and REFINE, as well as the representation of the maintained abstract PTS to enable an efficient implementation of these operations. In the refinement steps below, we always assume to start with representations of a valid Cartesian abstraction σ and its corresponding abstract PTS $\Theta_\sigma := \sigma(\Theta(\Pi))$.

3.1 Representing the Abstraction

To find flaws in a Cartesian abstraction, we need to distinguish between the individual outcomes of all probabilistic action effects. However, this information is not preserved in the PTS. We follow prior work [18] and *annotate* the transition relation, storing transitions $\langle A, o, \delta\rangle \in T_{\Theta_\sigma}$ as tuples $\langle A, o, \langle B_1, \ldots, B_{arity(o)}\rangle\rangle$ where B_i is the resulting abstract state under effect $\mathit{eff}_i(o)$. The set of annotated abstract transitions is given by $\tilde{T}_{\Theta_\sigma} := \{\langle \sigma(s), o, \langle \sigma(s[\![\mathit{eff}_1(o)]\!]), \ldots, \sigma(s[\![\mathit{eff}_{arity(o)}(o)]\!])\rangle\rangle \mid o \in \mathcal{O}_\Pi, s \in \mathit{Cart}(\mathit{pre}(o))\}$. Importantly, we also represent policies over this more fine-grained transition representation, i.e., from now on we consider policies $\pi_\sigma : S_{\Theta_\sigma} \to \tilde{T}_{\Theta_\sigma}$. Obviously, we can ignore the additional information at any point in time by aggregating the probabilities of the effects leading to the same abstract state.

3.2 Computing Optimal Abstract Policies

FINDOPTIMALSOLUTION can be implemented by any standard SSP algorithm that guarantees optimality. Doing this naively, however, is bound to generate a significant overhead. To reduce the overhead, we adopt an idea from the classical planning implementation, namely leveraging heuristic search while using the results from previous refinement iterations as heuristic guidance [27].

In our implementation, we compute optimal abstract policies using iLAO* [9], combined with FRET-π [31] to guarantee optimality also in the presence of zero cost operators. An important invariant of iLAO* is that the internal value function J maintained by the algorithm satisfies $h(s) \leq J(s) \leq J^*(s)$ at any point in time, where h is the admissible heuristic used by the search. Starting from the heuristic h_i of the CEGAR iteration i, we use this observation to construct from the corresponding J_i an admissible heuristic h_{i+1} for the next iteration $i+1$. As aforementioned, the corresponding abstract PTS $\Theta_{\sigma_{i+1}}$ differs from the previous Θ_{σ_i} only in having substituted some abstract state A by two new abstract states B_1 and B_2. As each refinement can only increase the J^* values, $h_{i+1}(B) := J_i(B)$ if $B \notin \{B_1, B_2\}$ and $h_{i+1}(B) := J_i(A)$ otherwise, is admissible. For the first iteration, we set $h_0(A) := 0$ which is trivially admissible.

Algorithm 2 Inspect an abstract policy for the concrete task.

```
1:  function FINDFLAW(Π, Θ_σ, π_σ)
2:      Q ← {I_Π}
3:      while Q ≠ ∅:
4:          s ← Q.POP()
5:          if σ(s) ∈ G_{Θ_σ}:
6:              if s ∈ Cart(G_Π): continue
7:              return ⟨s, σ(s) ∩ Cart(G_Π)⟩
8:          ⟨σ(s), o, ⟨B_1, ..., B_{arity(o)}⟩⟩ ← π_σ(σ(s))
9:          if s ∉ Cart(pre(o)): return ⟨s, σ(s) ∩ Cart(pre(o))⟩
10:         for all i ∈ {1, ..., arity(o)}:
11:             t_i ← s[[eff_i(o)]]
12:             if σ(t_i) ≠ B_i: return ⟨s, σ(s) ∩ regr_i(B_i, o)⟩
13:             Q.INSERTIFNEW(t_i)
14:     return "no flaw"
```

3.3 Finding Flaws in the Abstraction

Let σ be our current Cartesian abstraction, let Θ_σ be its (annotated) abstract PTS, and let π_σ be an optimal policy for the abstract initial state $\sigma(I_\Pi)$. We try to convert π_σ into an (implicit) concrete solution π for Π with the same expected cost $J^\pi_{\Theta(\Pi)}(I_\Pi) = J^{\pi_\sigma}_{\Theta_\sigma}(\sigma(I_\Pi))$ for the initial state. Algorithm 2 depicts the general procedure. We incrementally expand π by iteratively processing the concrete states s reachable by executing the current π from I_Π, and for each such s attempt to have π imitate the choice for the corresponding abstract state $\sigma(s)$ under the abstract policy π_σ. More precisely, if $\pi_\sigma(\sigma(s)) = \langle \sigma(s), o, \langle B_1, \ldots, B_n\rangle\rangle$, we try to select in $\pi(s)$ the (unique) concrete transition outgoing from s that is labeled by the same operator o. If $\sigma(s) \notin \pi_\sigma$, we also leave $\pi(s)$ undefined.

Akin to classical planning, this solution reconstruction may fail because of one of three reasons: (1) the chosen operator o is inapplicable in the concrete state s; (2) o is applicable, but the corresponding concrete transition *diverges* from the abstract one, i.e., there exists an operator effect $\mathit{eff}_i(o)$ such that $\sigma(s[\![\mathit{eff}_i(o)]\!]) \neq B_i$; and (3) the concrete policy terminates in a non-goal state s. In all three cases, the abstraction is too *coarse* in the sense that it does not sufficiently distinguish between concrete states. Algorithm 2 captures the reason of failure in terms of a *flaw*, i.e., a pair of concrete state s where the failure occurred and a non-empty Cartesian set $F \in \mathcal{C}(\mathcal{V}_\Pi)$ with $s \notin F \subseteq \sigma(s)$, representing the failure condition. This flaw is subsequently used to refine the abstraction (next subsection).

Flaw identification and extraction closely resemble the classical planning CEGAR variant. For (1), if the concrete state s does not satisfy the precondition of the chosen operator o, then $s \notin \mathit{Cart}(\mathit{pre}(o))$, so $\langle s, \sigma(s) \cap \mathit{Cart}(\mathit{pre}(o))\rangle$ is a flaw. For (3), to ensure that π reaches the goal with certainty, we generate the flaw $\langle s, \sigma(s) \cap \mathit{Cart}(G_\Pi)\rangle$ whenever $\sigma(s)$ is an abstract goal, i.e., $\sigma(s) \cap \mathit{Cart}(G_\Pi) \neq \emptyset$, but $s \notin \mathit{Cart}(G_\Pi)$. Lastly, for the *spurious transitions* (case 2), let $\mathit{regr}_i(A, o)$ denote the *Cartesian regression* of a Cartesian set $A \in \mathcal{C}(\mathcal{V}_\Pi)$ over the i-th effect of operator o:

$$\mathit{dom}(v, \mathit{regr}_i(A, o)) := \begin{cases} \{\mathit{pre}(o)[v]\} & \text{if } v \in \mathit{pre}(o), \\ \mathcal{D}(v) & \text{otherwise if } v \in \mathit{eff}_i(o), \\ \mathit{dom}(v, A) & \text{otherwise.} \end{cases}$$

Observe that if there is an (annotated) abstract transition $\langle A, o, \langle B_1, \ldots, B_n\rangle\rangle$, then $\mathit{regr}_i(B_i, o) \cap A \neq \emptyset$ for all i. In contrast, if a concrete state s has an o-labeled transition where $\sigma(s[\![\mathit{eff}_i(o)]\!]) \neq B_i$, then $s \notin \mathit{regr}_i(B_i, o)$. Hence, a flaw suitable

for case (2) is $\langle s, \sigma(s) \cap \mathit{regr}_i(B_i, o)\rangle$.

Obviously, if the abstract policy contains none of the errors (1)–(3), one has reconstructed a solution for the concrete initial state. The expected cost of this solution is the same as the expected cost of the abstract solution, by which we can conclude:

Theorem 1 *Algorithm 2 returns "no flaw" only if* $J^*_{\Theta(\Pi)}(I_\Pi) = J^*_{\Theta_\sigma}(\sigma(I_\Pi))$. *If it returns a flaw* $\langle s, F\rangle$, *then* $s \notin F \subseteq \sigma(s)$.

We note that a similar procedure has already been presented in the context of pattern generation for PDBs [17]. However, that context is simpler, as the abstract PTS is a projection, in which spurious transitions do not exist, which simplifies the flaw extraction.

3.4 Refining the Abstraction

Given a flaw $\phi = \langle s, F\rangle$, i.e., $F \subseteq \sigma(s)$ and $s \notin F$, found by the abstract policy inspection, in order to ensure progress in the next CEGAR iteration, we need to refine the abstraction in a way that separates F from the new abstract state that s is mapped to.

Splitting a Cartesian Abstract State

For the refinement, we split the abstract state $\sigma(s)$ into two new abstract states A_1, A_2 with $A_1 \cup A_2 = \sigma(s)$ such that $s \in A_1$ and $F \subseteq A_2$. To do so, we proceed as in classical planning. Since $s \notin F$ and F is a Cartesian set, there must be a *split variable* $v \in \mathcal{V}_\Pi$ such that $s[v] \notin \mathit{dom}(v, F)$. We partition $\sigma(s)$ into A_1 and A_2 as follows. For v, we choose $\mathit{dom}(v, A_1) := \mathit{dom}(v, \sigma(s)) \setminus \mathit{dom}(v, F)$ and $\mathit{dom}(v, A_2) := \mathit{dom}(v, F)$. For the remaining variables $v' \in \mathcal{V}_\Pi$, $v' \neq v$, the abstract domains are unchanged, i.e., $\mathit{dom}(v', A_1) := \mathit{dom}(v', A_2) := \mathit{dom}(v', \sigma(s))$. Obviously, the resulting split states satisfy $s \in A_1$ and $F \subseteq A_2$, as desired.

Incrementally Updating the Abstract PTS

After splitting $\sigma(s)$ into two new abstract states, yielding the refined Cartesian abstraction σ', we need to prepare the new abstract PTS $\Theta_{\sigma'}$ for the next CEGAR iteration. Constructing $\Theta_{\sigma'}$ from scratch is computationally expensive. Fortunately, the differences between $\Theta_{\sigma'}$ and the previous abstract PTS Θ_σ are limited to the newly introduced abstract states. We next show how to modify Θ_σ to obtain $\Theta_{\sigma'}$. This incremental update procedure is an essential part of the classical planning CEGAR algorithm [27], but the probabilistic setting introduces new unique challenges.

Let A_1 and A_2 be the new Cartesian abstract states, and let v be the corresponding split variable. Updating the abstract goal states is analogous to the classical planning procedure. Namely, observe that $A_i \in G_{\Theta_{\sigma'}}$ iff $\sigma(s) \in G_{\Theta_\sigma}$ and either $v \notin G_\Pi$ or $G_\Pi[v] \in \mathit{dom}(v, A_i)$, where $i \in \{1, 2\}$. The remaining abstract goal states are unaffected.

Refining the transitions is more complicated than in the classical planning setting, however. Obviously, transitions in Θ_σ not involving $\sigma(s)$ are not affected as the involved abstract states are still present in $\Theta_{\sigma'}$. Transitions involving $\sigma(s)$ must be rewired to the split states A_1 or A_2. At first glance, this rewiring step could in principle introduce exponentially many new transitions: consider an annotated transition $\langle B_0, o, \langle B_1, \ldots, B_{\mathit{arity}(o)}\rangle\rangle$ in Θ_σ. If $B_i = \sigma(s)$ for all i, this single transition potentially induces $2^{\mathit{arity}(o)+1}$ many different annotated transitions in $\Theta_{\sigma'}$, one for every combination of $B_i \in \{A_1, A_2\}$.

Luckily, only at most two of these transitions are actually possible, and we can efficiently test for their existence. To see this, let

Algorithm 3 Compute rewired transitions after splitting $\sigma(s)$ into A_1 and A_2 via split variable v.

```
1: function RewireTransition(⟨B_0, o, ⟨B_1, ..., B_arity(o)⟩⟩)
2:     B'_0, ..., B'_arity(o) ← B_0, ..., B_arity(o)   ▷ Initialize rewiring
3:     for all i ∈ {0, ..., arity(o)}:
4:         if B_i = σ(s) and v ∈ post_i(o):
5:             B'_i ← { A_1   post_i(o)[v] ∈ dom(v, A_1)
                      { A_2   else, i.e., post_i(o)[v] ∈ dom(v, A_2)
6:     RewireUniformly ← {i | B_i = σ(s) ∧ v ∉ post_i(o)}
7:     if RewireUniformly = ∅:
8:         return ⟨B'_0, o, ⟨B'_1, ..., B'_arity(o)⟩⟩
9:     LeftIntersection ← ⋂_{i:B_i≠σ(s)∧v∉post_i(o)} dom(v, B_i)
10:    for all A ∈ {A_1, A_2}:
11:        if LeftIntersection ∩ dom(v, A) ≠ ∅:
12:            for all i ∈ RewireUniformly: B'_i ← A
13:            output ⟨B'_0, o, ⟨B'_1, ..., B'_arity(o)⟩⟩
```

$\mathit{post}_i(o) := \mathit{pre}(o)[\![\mathit{eff}_i(o)]\!]$ be the i-th *post-condition* for $1 \leq i \leq \mathit{arity}(o)$, and let $\mathit{post}_0(o) := \mathit{pre}(o)$ to ease notation in the following. By definition of the abstract PTS and the induced PTS of a planning task, an abstract transition $\langle B_0, o, \langle B_1, \ldots, B_{\mathit{arity}(o)}\rangle\rangle$ exists if and only if there is a concrete state s_0 such that $s_0[\![\mathit{post}_i(o)]\!] \in B_i$ for $0 \leq i \leq \mathit{arity}(o)$. In more detail, the transition hence exists if and only if for $0 \leq i \leq \mathit{arity}(o)$ and for each variable $v \in \mathcal{V}_\Pi$:

$$v \in \mathit{post}_i(o) \text{ implies } \mathit{post}_i(o)[v] \in \mathit{dom}(v, B_i); \text{ and} \quad \text{(i)}$$

$$\bigcap_{i:v\notin \mathit{post}_i(o)} \mathit{dom}(v, B_i) \neq \emptyset. \quad \text{(ii)}$$

These observations lead to our rewiring method depicted in Algorithm 3. Since the split states A_1 and A_2 only differ from $\sigma(s)$ for the split variable v, it suffices to consider conditions (i) and (ii) of the transition check for v only. To rewire $B_i = \sigma(s)$ with $v \in \mathit{post}_i(o)$ (line 4), we check whether $\mathit{post}_i(o)[v] \in \mathit{dom}(v, A_1)$, in which case B_i must be rewired to A_1 as per condition (i); or vice versa $\mathit{post}_i(o)[v] \in \mathit{dom}(v, A_2) = \mathit{dom}(v, B_i) \setminus \mathit{dom}(v, A_1)$. Similarly, to rewire $B_i = \sigma(s)$ with $v \notin \mathit{post}_i(o)$, since $\mathit{dom}(v, A_1) \cap \mathit{dom}(v, A_2) = \emptyset$, (ii) can only hold if all such states are *uniformly* rewired to the same target A_1 or A_2. To check whether we can uniformly rewire these states to A_i, $i \in \{1, 2\}$, note that (ii) can be written as $\bigcap_{i:v\notin \mathit{post}_i(o)\wedge B_i\neq\sigma(s)} \mathit{dom}(v, B_i) \cap \mathit{dom}(v, A_i) \neq \emptyset$ after fixing a uniform rewire target. The left part of this intersection is computed in line 9, while line 11 finally intersects with $\mathit{dom}(v, A_i)$ and checks for emptiness. If the check succeeds, the respective states are uniformly rewired in line 12. All in all, this leads to at most two newly generated transitions. Repeating Algorithm 3 for all transitions in Θ_σ involving $\sigma(s)$ suffices to construct the transitions of $\Theta_{\sigma'}$.

3.5 Theoretical Properties

Without premature termination of the CEGAR loop, e.g., through a time limit, our algorithm converges to the optimal cost value of the initial state $J^*_{\Theta(\Pi)}(I_\Pi)$. This holds because every CEGAR iteration identifying a flaw results in a strictly finer abstraction going into the next iteration. This can however not happen indefinitely as eventually the abstraction mapping becomes the identity function, and then no more flaws can be found. The claim follows from Theorem 1.

Theorem 2 *If not terminated prematurely, Algorithm 1 terminates with* $J^*_{\Theta_\sigma}(\sigma(I_\Pi)) = J^*_{\Theta(\Pi)}(I_\Pi)$.

4 Multiple Cartesian Abstractions

A single abstraction usually faces the problem of diminishing returns, in the sense that the number of abstract states grows much more quickly than the quality of the abstraction heuristic [20]. In the following, we show how to overcome this issue by generating multiple diverse probability-aware Cartesian abstractions, and combining them admissibly via saturated cost partitioning [28].

4.1 Background on Cost Partitioning

Cost partitioning [13] is a technique from classical planning for additively combining several admissible heuristics $h_1, \dots, h_n$ while preserving admissibility. A theoretical analysis has recently shown that the same principles also apply in the SSP setting [15], even when allowing negative costs for the cost partitions, known as general cost partitioning [21]. The basic idea is to split the cost function C of the planning task into a *cost partition* $C_1, \dots, C_n$, guaranteeing that $\sum_{1 \leq i \leq n} C_i(o) \leq C(o)$ for every $o \in \mathcal{O}$. By evaluating each heuristic h_i under the cost function C_i, resulting in a new heuristic $h_i(\cdot, C_i)$, one obtains an admissible cost-partitioned heuristic $h(s) := h_1(s, C_1) + \cdots + h_n(s, C_n)$.

Saturated cost partitioning [28] is an efficient method to compute cost partitions. A *saturated cost function* (SCF) for heuristic h and cost function C is a cost function C' with $C' \leq C$ and $h(\cdot, C) = h(\cdot, C')$, i.e., using C' yields the same heuristic as C. Furthermore, C is a *minimal* SCF iff $C \leq C'$ for all SCFs C'. Given an ordering $h_1, \dots, h_n$ of the heuristics, saturated cost partitioning iterates through the heuristics in this order, computing the next heuristic h_{i+1} under the remaining costs $\mathit{remain}_{i+1}(\ell) := \mathit{remain}_i(\ell) - \mathit{scf}_i(\ell)$, where $\mathit{remain}_1 := C$ and scf_i is any SCF for h_i and remain_i. The resulting cost partition is $\mathit{scf}_1, \dots, \mathit{scf}_n$.

Ideally, scf_i is always minimal, so that the remaining costs are as high as possible. However, sometimes the cost of a label ℓ can be made arbitrarily low by scf_i without affecting the heuristic. It is convenient in classical planning to model this situation with a saturated cost of $\mathit{scf}_i(\ell) = -\infty$. In response, the remaining cost $\mathit{remain}_{i+1}(\ell) = +\infty$ is interpreted as arbitrarily high. In the context of a transition system, costs of $\pm\infty$ are handled using the usual arithmetic rules for extended real numbers $\overline{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$, including the *path-addition rule* $\infty + (-\infty) = \infty$. This rule intuitively treats transitions with cost $+\infty$ as being non-existent. A solution containing such a transition always has total cost $+\infty$, even if it contains a transition with cost $-\infty$. Hence, such solutions will be ignored.

4.2 Saturated Cost Partitioning for PTS

To keep our contribution as general as possible, we henceforth consider PTS Θ with cost functions $C_\Theta : S_\Theta \to \overline{\mathbb{R}}$, to mirror the classical framework outlined above. The optimal value function is defined differently in this setting, accounting for the possibility of negative cost cycles. J^*_Θ is now the greatest function $J : S_\Theta \to \overline{\mathbb{R}}$ satisfying the Bellman optimality equations [2]:

$$J(s) = \begin{cases} \infty & \text{if } J^\pi_\Theta(s) = \infty \; \forall \pi \in \mathit{Sols}(s), \\ \min\{0, (\mathbb{B}_\Theta J)(s)\} & \text{if } s \in G_\Theta, \\ (\mathbb{B}_\Theta J)(s) & \text{otherwise,} \end{cases} \tag{1}$$

$$\text{where} \quad (\mathbb{B}_\Theta J)(s) := \inf_{\langle s, \ell, \delta \rangle \in T_\Theta} C_\Theta(\ell) + \sum_{t \in S_\Theta} \delta(t) \cdot J(t).$$

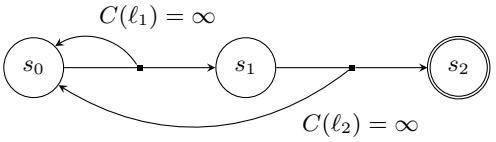

Figure 1: Counterexample used in Theorem 3.

To compute saturated cost partitions in our setting, we need to specify how to obtain an SCF for a probability-aware Cartesian abstraction heuristic. To this end, it suffices to show how to saturate J^*_Θ for the cost function C_Θ of some abstract PTS Θ. In the following, we denote with $\Theta[C]$ the PTS with the same components as Θ, except that the cost function is changed to C.

Abstractions in classical planning are known to have a unique minimal SCF [28]. Surprisingly, this property no longer holds for probability-aware abstraction heuristics.

Theorem 3 *The minimum saturated cost function for probability-aware abstraction heuristics is not unique in general.*

Proof. Consider the PTS Θ in Figure 1, where the transition probabilities are irrelevant. We have $J^*_\Theta(s_0) = J^*_\Theta(s_1) = \infty$ because the only solutions for s_0 and s_1 use infinite costs. Consider C^1_Θ and C^2_Θ, with $C^1_\Theta(\ell_1) = +\infty$, $C^1_\Theta(\ell_2) = -\infty$, and vice versa $C^2_\Theta(\ell_1) = -\infty$, and $C^2_\Theta(\ell_2) = +\infty$. Both cost functions are saturated by the path addition rule, because the only solution has a trace that encounters positively infinite costs. Any cost function C' that assigns both ℓ_1 and ℓ_2 a cost different from $+\infty$ is not saturated, because $J^*_\Theta \neq \infty$ under C'. In conclusion, C^1_Θ and C^2_Θ both are minimum SCFs, showing the claim. □

Recall that transitions with cost $+\infty$ can essentially be regarded as pruned due to the path addition rule. The non-uniqueness of the minimum SCF stems from the fact that introducing two new transitions to a PTS may introduce a new solution for a state, while adding them individually does not. This situation cannot occur in classical planning. Because of this, we employ the following SCF that preserves the infinite costs of the original cost function.

Theorem 4 *Let Θ be a PTS and define $T^{\mathit{fin}}_\Theta := \{\langle s, \ell, \delta \rangle \in T_\Theta \mid \forall t \in \mathit{supp}(\delta). J^*_\Theta(t) \neq \infty\}$. Then the function $\mathit{scf}(\ell) := \infty$ if $C_\Theta(\ell) = \infty$ and otherwise*

$$\mathit{scf}(\ell) := \sup_{\langle s, \ell, \delta \rangle \in T^{\mathit{fin}}_\Theta} J^*_\Theta(s) - \sum_{t \in S} \delta(t) \cdot J^*_\Theta(t)$$

is an SCF for J^*_Θ and C_Θ, i.e., $J^*_{\Theta[\mathit{scf}]} = J^*_\Theta$.

Proof (sketch). We have $J^*_\Theta(s) - \sum_{t \in S_\Theta} \delta(t) \cdot J^*_\Theta(t) \leq C_\Theta(\ell)$ for every transition $\langle s, \ell, \delta \rangle \in T^{\mathit{fin}}_\Theta$ because of (1). Therefore, $\mathit{scf} \leq C_\Theta$ by taking the supremum on the left hand side, implying $J^*_{\Theta[\mathit{scf}]} \leq J^*_\Theta$.

For the reverse inequality, recall that $J^*_{\Theta[\mathit{scf}]}$ is the greatest solution of (1) for $\Theta[\mathit{scf}]$, so it suffices to show that J^*_Θ is also a solution. The full details of this step can be found in our online appendix [16]. □

4.3 Computing Diverse Cartesian Abstractions

To generate multiple diverse Cartesian abstractions that focus on different aspect of the problem, we borrow the well-established ideas of *task decomposition via goals* and *task decomposition via landmarks*

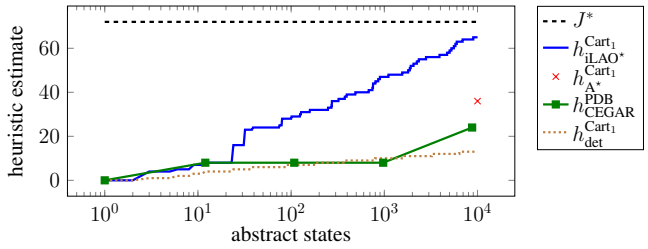

Figure 2: Initial state estimate over time in the problem P05-C3-P3-A3-S24056.PDDL of ZENOTRAVEL.

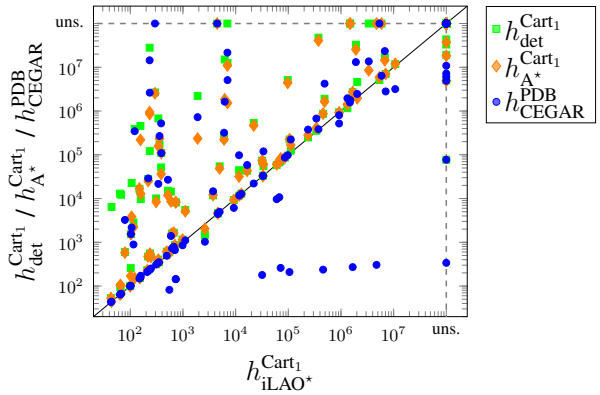

Figure 3: Number of evaluated states for each problem instance.

from classical planning [27]. The first method computes one abstraction for each goal fact, i.e., as input to the CEGAR procedure, we pass a modified task which contains exactly one goal fact and is otherwise identical to the original task. This task models a non-induced abstraction in which only one goal fact must be reached instead of multiple, and can only decrease the optimal cost-to-goal values, guaranteeing admissibility. The landmark decomposition precomputes a set of fact landmarks of the original task using $h^{\max}$ and computes one abstraction for each landmark found. Here, the modified task for CEGAR contains the fact landmark as the single goal fact. Additionally, every state which can be reached only after achieving the fact landmark is made a goal state, in order to guarantee admissibility (we refer to [27] for details). In our setting, we compute the fact landmarks by using $h^{\max}$ on the determinization of the task. A fact landmark in the determinization must be reached on every trace of an optimal policy, so this decomposition method still yields an admissible probability-aware abstraction heuristic.

5 Experiments

Our implementation is based on probabilistic Fast Downward [11, 31]. We provide the source code, our benchmark set and the detailed experimental data in an online appendix [16]. We compare probability-aware Cartesian abstraction heuristics against various other previously considered admissible heuristics for probabilistic planning. We use iLAO* combined with FRET-π throughout. The experiments were run using Downward Lab [29] on a cluster with Intel Xeon E5-2650 v3 processors @2.30 GHz CPUs, under memory and runtime limits of 4 GiB and 30 minutes.

As our benchmark set, we use nine PPDDL SSP domains, some of which contain zero-cost actions or avoidable dead ends. Eight of the domains stem from the IPPCs 2004, 2006 and 2008, subsuming the six domains used by Klößner and Hoffmann [14], but with 20 newly generated problem instances of more smoothly increasing difficulty. Additionally, we use the 20 largest instances from the stochastic version of the PARCPRINTER domain [32].

5.1 Single Abstractions

In our first experiment, we evaluate abstraction heuristics based on single abstractions constructed by running CEGAR with a time limit of 900s and a memory limit of 3.5 GB for the refinement. Note that the abstract PTSs are freed after construction, so most of the memory is reclaimed for the search. We experiment with three Cartesian abstraction variants: two probability-aware variants, where one is constructed with our proposed CEGAR algorithm ($h^{\text{Cart}_1}_{\text{iLAO}^\star}$) and one using classical CEGAR applied to the determinization of the probability-aware abstraction ($h^{\text{Cart}_1}_{\text{A}^\star}$); for comparison, we also experiment with a Cartesian abstraction heuristic entirely based on the determinization of the input task ($h^{\text{Cart}_1}_{\text{det}}$). In difference to $h^{\text{Cart}_1}_{\text{det}}$, $h^{\text{Cart}_1}_{\text{A}^\star}$ uses the determinization only for flaw finding while building and subsequently deriving the heuristic estimates from a probability-aware Cartesian abstraction. Finally, for reference, we include a probability-aware PDB heuristic comprised of just a single CEGAR-constructed projection ($h^{\text{PDB}}_{\text{CEGAR}}$), and the blind heuristic $h^{\text{blind}}(s) := 0$.

The left part of Table 1 provides the coverage results. The different Cartesian abstraction variants achieve a similar coverage, while $h^{\text{PDB}}_{\text{CEGAR}}$ performs slightly better in this regard. Overall, $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ spends over 90% of its time finding abstract solutions. In BLOCKSWORLD in particular, $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ spends increasingly large time portions on finding abstract solutions as the refinement loop progresses. It never reaches more than about 3 000 abstract states for any instance whereas, in comparison, $h^{\text{Cart}_1}_{\text{A}^\star}$ frequently reaches more than 10 000 and $h^{\text{PDB}}_{\text{CEGAR}}$ often reaches more than 100 000 abstract states. A phenomenon we observe in TTIREWORLD is that $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ expands a huge part of the concrete state space during the flaw analyses of the abstract policies. The overhead generated by the incremental abstract PTS updates is negligible. These observations clearly suggest that we need more efficient techniques for finding abstract solutions, for example, by adapting incremental search techniques as was done for classical Cartesian CEGAR [30]. Also, the flaw finding procedure may exhaust the memory for large policies, if not stopped prematurely.

While the raw coverage performance is somewhat limited by the overhead of the abstraction construction, in terms of heuristic accuracy, however, our new probability-aware Cartesian abstractions excel. Figure 3 compares the number of evaluated states incurred by the different heuristics. We can see that $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ expands much fewer states than its relatives $h^{\text{Cart}_1}_{\text{A}^\star}$ and $h^{\text{Cart}_1}_{\text{det}}$ in most problem instances. The differences are considerably larger compared to $h^{\text{Cart}_1}_{\text{det}}$ than to $h^{\text{Cart}_1}_{\text{A}^\star}$, indicating the importance of taking into account the probabilistic nature of the tasks for deriving the heuristic estimates. Taking into account the probabilities during CEGAR already is however the key advantage of $h^{\text{Cart}_1}_{\text{iLAO}^\star}$. In many problem instances, the construction in $h^{\text{Cart}_1}_{\text{A}^\star}$ and $h^{\text{Cart}_1}_{\text{det}}$ terminates very early due to finding an optimal plan in the determinization while $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ still finds flaws. Figure 2 visualizes one such example in ZENOTRAVEL, depicting how the initial state estimate changes over time as the refinement progresses. We see that $h^{\text{Cart}_1}_{\text{det}}$ improves slowly, exhibiting frequent plateaus; the heuristic estimate of $h^{\text{Cart}_1}_{\text{A}^\star}$ (which our implementation computes only at the end, as flaws are found in the determinization) is significantly improved, but the flaws considered for refinement are less useful for the heuristic quality compared to $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ as optimal plans and optimal policies not necessarily correlate.

	h^{blind} and single abstraction					multiple abstractions													
Domains	h^{blind}	$h^{\text{PDB}}_{\text{CEGAR}}$	$h^{\text{Cart}_1}_{\text{det}}$	$h^{\text{Cart}_1}_{\text{iLAO}^\star}$	$h^{\text{Cart}_1}_{\text{A}^\star}$	h^{roc}	$h^{\text{SYS-1}}_{\text{Can}}$	$h^{\text{SYS-2}}_{\text{Can}}$	$h^{\text{SYS-3}}_{\text{Can}}$	$h^{\text{HC}}_{\text{Can}}$	$h^{\text{DC}}_{\text{Max}}$	$h^{\text{SYS-1}}_{\text{SCP}}$	$h^{\text{SYS-2}}_{\text{SCP}}$	$h^{\text{SYS-3}}_{\text{SCP}}$	$h^{\text{HC}}_{\text{SCP}}$	$h^{\text{DC}}_{\text{SCP}}$	$h^{\text{Cart}_k}_{\text{det}}$	$h^{\text{Cart}_k}_{\text{iLAO}^\star}$	$h^{\text{Cart}_k}_{\text{A}^\star}$
BLOCKSWORLD	7	**8**	7	7	7	7	**9**	**9**	8	**9**	8	**9**	**9**	**9**	**9**	**9**	7	7	7
BOXWORLD	4	5	**7**	5	**7**	4	5	5	4	6	4	5	**7**	**7**	6	6	6	**7**	6
ELEVATORS	10	**11**	10	**11**	10	10	13	12	8	**17**	15	13	15	**17**	**17**	**17**	15	14	16
PARCPRINTER	**8**	**8**	**8**	**8**	**8**	**20**	13	8	8	19	11	14	**20**	6	**20**	18	12	14	12
RANDOM	14	17	**18**	17	**18**	14	16	17	13	**18**	17	**18**	**18**	13	**18**	16	**18**	16	**18**
SCHEDULE	12	**15**	13	13	12	11	12	13	12	14	13	14	12	12	13	13	13	12	12
SYSADMIN	**12**	**12**	**12**	11	11	**12**	**12**	**12**	8	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
TTIREWORLD	5	**9**	7	**9**	7	7	7	7	8	**9**	**9**	7	7	8	**9**	**9**	7	**9**	7
ZENOTRAVEL	5	10	8	**11**	10	7	9	10	9	10	9	9	9	9	10	10	8	10	10
Sum (of 180)	77	**95**	90	92	90	92	96	93	78	**114**	98	101	109	93	**114**	110	98	100	101

Table 1: Number of tasks solved by single-abstraction and multiple-abstractions configurations. Highest numbers within each group in boldface.

Comparing $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ to $h^{\text{PDB}}_{\text{CEGAR}}$, Figure 3 shows an advantage for $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ in terms of the number of evaluated states in many cases. While $h^{\text{Cart}_1}_{\text{iLAO}^\star}$ expands less states than $h^{\text{PDB}}_{\text{CEGAR}}$ in 42 tasks, the opposite is only true in 21 tasks, despite that $h^{\text{PDB}}_{\text{CEGAR}}$ is consistently building much larger abstractions than $h^{\text{Cart}_1}_{\text{iLAO}^\star}$. While Cartesian abstractions permit fine-grained refinements, the size of the PDB increases exponentially with each refinement (see Figure 2). Due to its limited representational power, $h^{\text{PDB}}_{\text{CEGAR}}$ often cannot improve significantly (for the initial state) before reaching the memory limit. The only exception is SCHEDULE in which $h^{\text{PDB}}_{\text{CEGAR}}$ excels.

5.2 Multiple Abstractions

In our second experiment, we evaluate combinations of multiple abstraction heuristics and provide comparisons to current state-of-the-art admissible SSP heuristics. We consider the same three Cartesian abstraction variants as in the previous experiment, but now constructing multiple abstractions using landmark and goal task decompositions. The individual abstraction heuristics are combined via saturated cost partitioning and denoted $h^{\text{Cart}_k}_{\text{iLAO}^\star}$, $h^{\text{Cart}_k}_{\text{A}^\star}$, and $h^{\text{Cart}_k}_{\text{det}}$ respectively. We compare these configurations to probability-aware PDB heuristics over multiple projections constructed using the disjoint CEGAR algorithm [24], hill-climbing based [10], and the systematic [22] pattern generation methods. As baselines, we use the same combination methods as in prior work [17], which is the additive canonical combination for hill climbing ($h^{\text{HC}}_{\text{Can}}$) and for the systematic PDBs ($h^{\text{SYS-k}}_{\text{Can}}$), and taking the maximum over the CEGAR-constructed projections ($h^{\text{DC}}_{\text{Max}}$) as these are typically not orthogonal. Furthermore, we also evaluate all PDB heuristics using saturated cost partitioning as combination method, denoted $h^{\text{DC}}_{\text{SCP}}$, $h^{\text{HC}}_{\text{SCP}}$, and $h^{\text{SYS-k}}_{\text{SCP}}$ respectively. Lastly, we include h^{roc} [32], computed using CPLEX 12.6.3 as the LP solver. $h^{\text{HC}}_{\text{Can}}$ and h^{roc} represent the current state of the art [17].

For the saturated cost partitioning algorithm, we follow a random abstraction order, except for the Cartesian abstraction heuristics which consider abstractions from the landmark decomposition before those generated from the goal decomposition, but otherwise still randomly. Across all the heuristics, we limit the abstraction construction time to 900 seconds. For $h^{\text{SYS-k}}_{\text{Can}}$ and $h^{\text{SYS-k}}_{\text{SCP}}$, we experiment with pattern size limits of $k \in \{1, 2, 3\}$. For all other heuristics, we enforce a limit on the number of abstract states to 50 000. We also experimented with a limit of 100 000, which yielded almost identical results. $h^{\text{Cart}_k}_{\text{iLAO}^\star}$, $h^{\text{Cart}_k}_{\text{A}^\star}$, $h^{\text{Cart}_k}_{\text{det}}$ and $h^{\text{SYS-k}}_{\text{SCP}}$ interleave the abstraction construction with saturated cost partitioning, distributing the time limit uniformly over all abstractions. All other (PDB) configurations do not allow for this, and so the resulting PDBs are recomputed under their saturated cost function after the construction algorithm finishes.

The right part of Table 1 shows the resulting coverage data. Regarding the Cartesian abstraction configurations, our probability-aware extensions solve more tasks than the fully-determinization-based variant, but fewer than the corresponding PDB configuration $h^{\text{DC}}_{\text{SCP}}$. As opposed to $h^{\text{DC}}_{\text{SCP}}$, $h^{\text{Cart}_k}_{\text{iLAO}^\star}$ again frequently runs into the construction time limit, often terminating with much fewer than 50 000 abstract states. The main bottleneck is once again the abstract solution computation. Whenever the time limit is not hit, both configurations evaluate a similar amount of states. We note that if the refinement is stopped early for a Cartesian abstraction, this may also decrease the remaining costs for succeeding abstractions, which further impedes the quality of the overall (combined) heuristic.

A clear message from Table 1 is that saturated cost partitioning is highly advantageous for SSP abstractions, significantly boosting the coverage of almost every configuration compared to the respective counterparts with the alternative combination methods. For example, $h^{\text{SYS-2}}_{\text{SCP}}$ solves 18 instances that $h^{\text{SYS-2}}_{\text{Can}}$ cannot solve, while the opposite is true for only 2 instances. The same holds true for $h^{\text{SYS-3}}_{\text{SCP}}$, $h^{\text{DC}}_{\text{SCP}}$, and for $h^{\text{SYS-1}}_{\text{SCP}}$ to a lesser extent. For $h^{\text{HC}}_{\text{SCP}}$, we see no overall change in coverage compared to $h^{\text{HC}}_{\text{Can}}$. However, one must note that the ranking function used by hill climbing to rank candidate patterns is based on the canonical PDB heuristic. The two best configurations, $h^{\text{HC}}_{\text{SCP}}$ and $h^{\text{HC}}_{\text{Can}}$, also have the least amount of evaluated states on average, with mild advantages over $h^{\text{DC}}_{\text{SCP}}$.

With respect to h^{roc}, among all tested configurations, only $h^{\text{SYS-3}}_{\text{Can}}$ solves fewer tasks overall. Although this observation is in line with observations from classical planning, where optimal cost partitioning over abstractions is inferior to other heuristic combination techniques in most cases, it disagrees with previous experiments in the SSP setting [14]. We attribute this to the benchmark collection. The original benchmarks from IPPCs 2004–2008 scale poorly in difficulty, making them less suitable for the evaluation of optimal offline planners.

6 Conclusions

We presented probability-aware Cartesian abstractions, which generalize PDB abstractions, as well as a construction algorithm based on CEGAR with convergence guarantees. Also, we adapted saturated cost partitioning for probability-aware abstraction heuristics. Our experiments show that probability-aware Cartesian abstractions are often more informed than their deterministic counterpart and than PDB heuristics constructed using CEGAR. Furthermore, saturated cost partitioning greatly improves over previously considered techniques for combining heuristics admissibly. In the future, we want to address the bottleneck in the CEGAR loop with incremental search, and compute saturated cost partitionings over multiple orders of abstraction heuristics.

Acknowledgements

This work was funded by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science) and partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by the Air Force Office of Scientific Research under award number FA9550-18-1-0245.

References

[1] Christer Bäckström and Bernhard Nebel, 'Complexity results for SAS^+ planning', *Computational Intelligence*, **11**(4), 625–655, (1995).

[2] Richard Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis, 'An analysis of stochastic shortest path problems', *Mathematics of Operations Research*, **16**, 580–595, (1991).

[4] Blai Bonet, 'An admissible heuristic for SAS+ planning obtained from the state equation', in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, ed., Francesca Rossi, pp. 2268–2274. AAAI Press/IJCAI, (2013).

[5] Blai Bonet and Hector Geffner, 'Labeled RTDP: Improving the convergence of real-time dynamic programming', in *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, eds., Enrico Giunchiglia, Nicola Muscettola, and Dana Nau, pp. 12–21, Trento, Italy, (2003). AAAI Press.

[6] Blai Bonet and Hector Geffner, 'mGPT: A probabilistic planner based on heuristic search', *Journal of Artificial Intelligence Research*, **24**, 933–944, (2005).

[7] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith, 'Counterexample-guided abstraction refinement for symbolic model checking', *Journal of the Association for Computing Machinery*, **50**(5), 752–794, (2003).

[8] Peng Dai, Mausam, Daniel S. Weld, and Judy Goldsmith, 'Topological value iteration algorithms', *Journal of Artificial Intelligence Research*, **42**, 181–209, (2011).

[9] Eric A. Hansen and Shlomo Zilberstein, 'LAO^*: a heuristic search algorithm that finds solutions with loops', *Artificial Intelligence*, **129**(1-2), 35–62, (2001).

[10] 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 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, eds., Adele Howe and Robert C. Holte, pp. 1007–1012, Vancouver, BC, Canada, (July 2007). AAAI Press.

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

[12] Sergio Jimenez, Andrew Coles, and Amanda Smith, 'Planning in probabilistic domains using a deterministic numeric planner', in *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig'06)*, (2006).

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

[14] Thorsten Klößner and Jörg Hoffmann, 'Pattern Databases for Stochastic Shortest Path Problems', in *Proceedings of the 14th Annual Symposium on Combinatorial Search (SOCS'21)*, pp. 131–135. AAAI Press, (2021).

[15] Thorsten Klößner, Florian Pommerening, Thomas Keller, and Gabrielle Röger, 'Cost Partitioning Heuristics for Stochastic Shortest Path Problems', in *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, pp. 193–202. AAAI Press, (2022).

[16] Thorsten Klößner, Jendrik Seipp, and Marcel Steinmetz. Supplementary Materials of the ECAI'23 submission "Cartesian Abstractions and Saturated Cost Partitioning in Probabilistic Planning". https://doi.org/10.5281/zenodo.8185720, 2023.

[17] Thorsten Klößner, Marcel Steinmetz, Àlvaro Torralba, and Jörg Hoffmann, 'Pattern Selection Strategies for Pattern Databases in Probabilistic Planning', in *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, p. 184–192. AAAI Press, (2022).

[18] Thorsten Klößner, Álvaro Torralba, Marcel Steinmetz, and Silvan Sievers, 'A Theory of Merge-and-Shrink for Stochastic Shortest Path Problems', in *Proceedings of the 33nd International Conference on Automated Planning and Scheduling (ICAPS'23)*, pp. 203–211. AAAI Press, (2023).

[19] Andrey Kolobov, Mausam, and Daniel S. Weld, 'LRTDP versus UCT for online probabilistic planning', in *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, eds., Jörg Hoffmann and Bart Selman, Toronto, ON, Canada, (July 2012). AAAI Press.

[20] Richard E. Korf, 'Finding optimal solutions to Rubik's Cube using pattern databases', in *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI'97)*, eds., Benjamin J. Kuipers and Bonnie Webber, pp. 700–705, Portland, OR, (July 1997). MIT Press.

[21] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp, 'From non-negative to general operator cost partitioning', in *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, eds., Blai Bonet and Sven Koenig, pp. 3335–3341. AAAI Press, (January 2015).

[22] Florian Pommerening, Gabriele Röger, and Malte Helmert, 'Getting the most out of pattern databases for classical planning', in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, ed., Francesca Rossi. AAAI Press/IJCAI, (2013).

[23] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet, 'LP-based heuristics for cost-optimal planning', in *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, eds., Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml, pp. 226–234. AAAI Press, (2014).

[24] Alexander Rovner, Silvan Sievers, and Malte Helmert, 'Counterexample-guided abstraction refinement for pattern selection in optimal classical planning', in *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*, pp. 362–367. AAAI Press, (2019).

[25] Jendrik Seipp and Malte Helmert, 'Counterexample-guided Cartesian abstraction refinement', in *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, eds., Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, pp. 347–351, Rome, Italy, (2013). AAAI Press.

[26] Jendrik Seipp and Malte Helmert, 'Diverse and additive Cartesian abstraction heuristics', in *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, eds., Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml, pp. 289–297. AAAI Press, (2014).

[27] Jendrik Seipp and Malte Helmert, 'Counterexample-guided Cartesian abstraction refinement for classical planning', *Journal of Artificial Intelligence Research*, **62**, 535–577, (2018).

[28] Jendrik Seipp, Thomas Keller, and Malte Helmert, 'Saturated cost partitioning for optimal classical planning', *Journal of Artificial Intelligence Research*, **67**, 129–167, (2020).

[29] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[30] Jendrik Seipp, Samuel von Allmen, and Malte Helmert, 'Incremental search for counterexample-guided Cartesian abstraction refinement', in *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, eds., J. Christopher Beck, Erez Karpas, and Shirin Sohrabi, pp. 244–248. AAAI Press, (2020).

[31] Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet, 'Goal probability analysis in MDP probabilistic planning: Exploring and enhancing the state of the art', *Journal of Artificial Intelligence Research*, **57**, 229–271, (2016).

[32] Felipe W. Trevizan, Sylvie Thiébaux, and Patrik Haslum, 'Occupation measure heuristics for probabilistic planning', in *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, pp. 306–315. AAAI Press, (2017).

[33] Felipe W. Trevizan, Sylvie Thiébaux, Pedro Henrique Santana, and Brian Williams, 'I-dual: Solving constrained SSPs via heuristic search in the dual space', in *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, ed., Carles Sierra, pp. 4954–4958. AAAI Press/IJCAI, (2017).