Merging Cartesian Abstractions for Classical Planning

Mauricio Salerno[1,*], **Raquel Fuentetaja**[1], **David Speck**[2] **and Jendrik Seipp**[3]

[1]Universidad Carlos III de Madrid, Leganés, Madrid, Spain
[2]University of Basel, Basel, Switzerland
[3]Linköping University, Linköping, Sweden

Abstract. Building a single Cartesian abstraction is usually not enough to obtain an informative heuristic for classical planning. Therefore, state-of-the-art methods decompose the original task into subtasks—for example, one per goal atom—and compute an abstraction for each individual subtask. However, building a single abstraction suffers from diminishing returns, while building multiple abstractions loses information about how to achieve the associated subtasks jointly. We interpolate between these two extremes by first considering subtasks individually and then merging some of the resulting abstractions. We introduce an efficient algorithm for merging pairs of Cartesian abstractions using their refinement hierarchies and show that it yields more informative abstractions in less time than a naive approach. Furthermore, we prove that adding merged abstractions can only improve a cost-partitioned heuristic based on saturated post-hoc optimization and that for maximal heuristic values, we need to keep the individual abstractions. Our experiments show that merging abstractions drastically improves the resulting heuristics.

1 Introduction

In optimal classical planning, the objective is to identify the cheapest sequence of actions that leads from an initial situation to a desired goal situation. One of the main solution methods is A* search [11] with an admissible heuristic [19], and a prominent approach to computing such an admissible heuristic involves determining goal distances in Cartesian abstractions [2, 28].

There are two primary paradigms for computing Cartesian abstraction heuristics in the literature. The first one focuses on refining a single abstraction until resource limits are reached, at which point the abstraction serves as the basis for a heuristic [30]. This approach converges to the perfect heuristic, but suffers from diminishing returns over time. The second paradigm involves identifying subtasks such as reaching individual goal atoms or landmarks [29], computing an abstraction for each subtask, and combining them using cost partitioning [16, 37, 21, 31, 5]. This approach yields state-of-the-art heuristics, but it does not converge to the perfect heuristic, since cost partitioning cannot recover all loss in heuristic accuracy that comes from splitting a task into subtasks.

The only approach in the literature that can converge to the perfect heuristic when starting from a set of Cartesian abstractions considers the *online* setting. Eifler and Fickert [8] create one Cartesian abstraction per goal atom and refine one of them during the A* search when the heuristic violates the Bellman equation (1957). If no refinement fixes the heuristic error, they merge two abstractions by computing the synchronized product. We discuss this approach in more detail in Section 7.

In contrast, for other classes of abstractions there are many approaches for generating increasingly more informative heuristics. For example, pattern database (PDB) heuristics [4, 6] can be obtained using genetic algorithms [7, 9], hill-climbing [12], systematic enumeration [20], or with cost partitioning [26, 22]. Furthermore, merge-and-shrink heuristics [33] operate on the factored task representation, gradually combining and coarsening abstractions using a variety of strategies for both operations.

Based on these observations, it is natural to move beyond using a fixed, predetermined set of Cartesian abstractions, and we make the following contributions: first, we introduce a systematic approach to generate Cartesian abstractions that gradually consider more goal atoms jointly; second, we introduce an efficient algorithm to compute the synchronized product of Cartesian abstractions using their refinement hierarchies; third, we analyze theoretically how adding product Cartesian abstractions affects three different cost partitioning techniques: optimal cost partitioning [17], saturated post-hoc optimization [20, 32] and saturated cost partitioning [31]; fourth, we empirically evaluate our new approach on a large set of benchmarks from the International Planning Competition (IPC) [36]. The results show that our new approach yield more informative heuristics, which solve many more tasks than state-of-the-art approaches for computing Cartesian abstractions.

2 Background

A SAS$^+$ planning task [1] with action costs is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. Each $v \in \mathcal{V}$ is a *finite-domain variable* with a finite *domain* $\mathcal{D}(v)$. A *partial state* s maps a subset of variables $\mathit{vars}(s) \subseteq \mathcal{V}$ to a value in their domain. We call s a *state* if $\mathit{vars}(s) = \mathcal{V}$. We denote the set of all states in Π by $S(\Pi)$. A pair $\langle v, d \rangle$ where $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$ is called an *atom*, written as $v \mapsto d$, and we often treat (partial) states as sets of atoms. Each *operator* $o \in \mathcal{O}$ is a tuple $\langle \mathit{pre}(o), \mathit{eff}(o), \mathit{cost}(o) \rangle$, where $\mathit{pre}(o)$ and $\mathit{eff}(o)$ are partial states defining the *preconditions* and *effects* of o, and $\mathit{cost}(o) \in \mathbb{R}_0^+$ is its *cost*. An operator o is applicable in state s if $\mathit{pre}(o) \subseteq s$, and the successor state resulting from applying o in s is $s[\![o]\!]$, with $s[\![o]\!][v] = \mathit{eff}(o)[v]$ if $v \in \mathit{vars}(\mathit{eff}(o))$, and $s[\![o]\!][v] = s[v]$ otherwise. State $\mathcal{I}$ is the *initial state* and $\mathcal{G}$ is a partial state describing the *goal*. An atom $v \mapsto d \in \mathcal{G}$ is a *goal atom*, and a state s such that $\mathcal{G} \subseteq s$ is a *goal state*.

A *transition system* $\mathcal{T} = \langle S, \mathcal{L}, T, s_0, S_\star \rangle$ is a directed labeled graph with a finite set of states S, labels $\mathcal{L}$, labeled transitions $T \subseteq$

[*] Corresponding Author. Email: msalerno@pa.uc3m.es

$S \times \mathcal{L} \times S$, an initial state $s_0 \in S$ and a set of goal states $S_\star \subseteq S$. A *path* is a sequence of transitions, and a *goal path* leads from the initial state to a goal state. The sequence of labels of a goal path is called a *plan*. When a transition system $\mathcal{T}$ is combined with a cost function $cost : \mathcal{L} \to \mathbb{R}$, we have a *weighted transition system* $\langle \mathcal{T}, cost \rangle$. For a weighted transition system, the cost of a plan is the sum of the costs of its labels, and a plan is optimal if there is no plan with a lower cost. A planning task Π induces a transition system $\mathcal{T}$, where the states S are the states $S(\Pi)$ in the planning task, the initial state is the same $s_0 = \mathcal{I}$, the set of labels $\mathcal{L}$ is the set of operators $\mathcal{O}$, the goal states are all states that contain the goal $S_\star = \{s \in S(\Pi) \mid \mathcal{G} \subseteq s\}$ and the transitions are defined by $T = \{\langle s, o, s' \rangle \mid s, s' \in S(\Pi), o \in \mathcal{O}, pre(o) \subseteq s, s' = s[\![o]\!]\}$. A plan is optimal for $\mathcal{T}$ iff it is optimal for Π.

An *abstraction* of a transition system $\mathcal{T}$ is a surjective function $\alpha : S \to S^\alpha$ mapping *concrete states* to *abstract states*. The *abstract state space* induced by α is the transition system $\mathcal{T}^\alpha = \langle S^\alpha, \mathcal{L}, T^\alpha, s_0^\alpha, S_\star^\alpha \rangle$, with $s_0^\alpha = \alpha(s_0)$, $S_\star^\alpha = \{\alpha(s) \mid s \in S_\star\}$ and $T^\alpha = \{\langle \alpha(s), \ell, \alpha(s') \rangle \mid (s, \ell, s') \in T\}$. Abstractions preserve paths in the original transition system. Thus, for a cost function *cost*, the *abstraction heuristic* $h^\alpha(cost, s) = h^*_{\mathcal{T}^\alpha}(cost, s^\alpha)$, which computes the cost of an optimal goal path from $s^\alpha = \alpha(s)$ in the abstract weighted transition system $\langle \mathcal{T}^\alpha, cost \rangle$, is admissible.

A *Cartesian abstraction* is an abstraction in which all abstract states are *Cartesian sets*. For a planning task with variables $\langle v_1, \dots, v_n \rangle$ a Cartesian set has the form $A_1 \times \cdots \times A_n$, where $A_i \subseteq \mathcal{D}(v_i)$ for all $1 \le i \le n$. The intersection of two Cartesian sets $A_1 \times \cdots \times A_n$ and $B_1 \times \cdots \times B_n$ is a Cartesian set defined by $(A_1 \cap B_1) \times \cdots \times (A_n \cap B_n)$ [30].

Counterexample-guided abstraction refinement (CEGAR) is a method that incrementally computes more fine-grained Cartesian abstractions of a transition system [30]. The process starts with a coarse abstraction where an abstract plan is computed. If said plan corresponds to a concrete plan, the procedure stops and no refinements are done to the abstraction. Otherwise, if the abstract plan does not correspond to a valid concrete plan, the reason is identified—this reason is called a *flaw*. There are three types of flaws: an operator is not applicable in the concrete plan; the concrete and abstract traces diverge (abstract state does not correspond to concrete state); and the final state is not a goal state in the concrete task. When a flaw is found, the abstraction is refined by splitting the abstract state that caused the flaw, guaranteeing that in future iterations the same flaw cannot occur again. This procedure is repeated until a concrete plan is found or a resource limit is reached.

Seipp and Helmert [30] note that building a single big Cartesian abstraction suffers from diminishing returns. To tackle this issue, they propose methods to create a set of diverse and small abstractions that capture relevant information about different aspects of the problem. One of these methods creates one Cartesian abstraction for each goal atom $\langle v \mapsto d \rangle \in \mathcal{G}$ and, after refining each *single-goal* abstraction, apply *cost partitioning* to get an admissible combination of the heuristic values of all abstractions. *Single-goal* abstractions are created from modified planning tasks, which are identical to the original one except that the considered goal is the only goal atom.

The *refinement hierarchy* [30] constitutes a fundamental data structure within the Cartesian CEGAR framework as it serves multiple purposes: it encodes the abstraction function; facilitates efficient on-demand computation of transitions between abstract states [27]; maintains the history of refinements starting from the trivial abstraction; and can be used to compute the set of abstract states. Formally, given a planning task with variables $\mathcal{V} = \langle v_1, \dots, v_m \rangle$, a *refinement hierarchy* $H = \langle N, E \rangle$ is a binary tree where each node $n \in N$ represents a Cartesian set $\mathcal{D}(v_1, n) \times \cdots \times \mathcal{D}(v_m, n)$. Leaf nodes of the tree represent states in an abstract transition system, while inner nodes represent splits (refinements). Each inner node $n \in N$ has exactly two children $a, b \in N$ with edges $\{\langle n, a \rangle, \langle n, b \rangle\} \subseteq E$. For each inner node n with children a and b, there exists a *split variable* $split\_var(n) = v_i \in \mathcal{V}$ (indicating the partitioning of $\mathcal{D}(v_i, n)$) such that $\mathcal{D}(v_i, a), \mathcal{D}(v_i, b) \subset \mathcal{D}(v_i, n)$, $\mathcal{D}(v_i, a) \cap \mathcal{D}(v_i, b) = \emptyset$, and $\mathcal{D}(v_i, a) \cup \mathcal{D}(v_i, b) = \mathcal{D}(v_i, n)$. For all other variables $v_j \in \mathcal{V} \setminus \{v_i\}$, it holds that $\mathcal{D}(v_j, a) = \mathcal{D}(v_j, b) = \mathcal{D}(v_j, n)$. We refer to the set of all splits as $Splits(H) = \{\langle n, a, b \rangle \mid \langle n, a \rangle, \langle n, b \rangle \in E\}$.

A *cost partitioning* over a sequence of admissible heuristics $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ for a weighted transition system $\langle \mathcal{T}, cost \rangle$ is a sequence of cost functions $C = \langle cost_1, \dots, cost_n \rangle$ such that $\sum_{i=1}^{n} cost_i(\ell) \le cost(\ell)$, for all $\ell \in \mathcal{L}$. The *cost partitioning heuristic* is $h^C(s) = \sum_{i=1}^{n} h_i(cost_i, s)$ [17]. Cost partitioning heuristics preserve admissibility.

For a heuristic, a *saturated cost function* (*scf*) is the minimum cost function that preserves all heuristic estimates. Formally, given an admissible heuristic h for a weighted transition system $\langle \mathcal{T}, cost \rangle$, a saturated cost function satisfies that: (1) $scf(\ell) \le cost(\ell)$ for all $\ell \in \mathcal{L}$, and (2) $h(scf, s) = h(cost, s)$ for all states $s \in S$. For abstraction heuristics there is a unique *minimum scf* defined for each label $\ell \in \mathcal{L}$ as $\max_{s \xrightarrow{\ell} s' \in T} (h^*_{\mathcal{T}}(cost, s) - h^*_{\mathcal{T}}(cost, s'))$ [31].

The *saturated post-hoc optimization* (SPhO) heuristic [20, 32] uses saturated cost functions scf_h for heuristics $h \in \mathcal{H}$ and solves the following linear program to obtain a heuristic value for state $s \in S$ in a transition system with labels $\mathcal{L}$:

$$\text{maximize} \sum_{h \in \mathcal{H}} h(cost, s) \cdot \omega_h \text{ subject to}$$

$$\sum_{h \in \mathcal{H}} scf_h(\ell) \cdot \omega_h \le cost(\ell) \text{ for all } \ell \in \mathcal{L} \tag{1}$$

$$\omega_h \ge 0 \text{ for all } h \in \mathcal{H} \tag{2}$$

3 Multi-Goal Cartesian Abstractions

Today's strongest heuristics based on Cartesian abstractions decompose the input task into subtasks and build one abstraction per subtask, dividing resources such as runtime among them equally. We focus on the basic version, where there is a subtask for each goal atom, and we call the resulting abstractions *single-goal* abstractions. Computing all single-goal abstractions is fast in practice, even for large problems [31]. This fast computation leaves room to further refine the individual Cartesian abstractions by adding more information about the concrete goal before combining them with cost partitioning. For this purpose, we consider Cartesian abstractions with n goal atoms instead of a single one [e.g., 31], and we call such abstractions *n-goal abstractions* or, if the number of considered goal atoms does not matter, simply *multi-goal abstractions*. A collection $\mathcal{A}$ of multi-goal Cartesian abstractions for a planning task Π is then defined as $\mathcal{A} = \{\alpha_{G_1}, \dots, \alpha_{G_n}\}$, where each α_{G_i} is a Cartesian abstraction computed for a non-empty set of goal atoms $G_i \subseteq \mathcal{G}$.

Moving from single-goal to multi-goal abstractions opens up a large design space for algorithms that generate collections of multi-goal abstractions, and two natural questions arise: (1) which multi-goal Cartesian abstractions to consider, and (2) how to efficiently generate and refine multi-goal Cartesian abstractions.

Regarding question (1), the number of possible multi-goal Cartesian abstractions for a planning task with n goal variables is $2^n - 1$, making it infeasible to always consider all of them. Therefore, we propose an incremental and systematic approach to include all multi-goal abstractions up to a certain number of goals m (or up to a certain time or memory limit). This is in line with previous work on systematic collections of pattern database heuristic [20]. More specifically, the idea is to start by creating and refining all 1-goal abstractions. Then we consider 2-goal abstractions by combining all pairs of 1-goal abstractions. In other words, for each pair of 1-goal abstractions $\langle \alpha_{G_i}, \alpha_{G_j} \rangle$, with $G_i \neq G_j$ and $|G_i| = |G_j| = 1$, we create and refine the 2-goal abstraction $\alpha_{G_i \cup G_j}$. Afterwards, to create 3-goal abstractions, combinations of 2-goal and 1-goal abstractions are considered. In general, to obtain i-goal abstractions, we consider $(i-1)$-goal abstractions with 1-goal abstractions, iteratively increasing $i \in \langle 1, \dots, m \rangle$ up to a given number of goal atoms $m \leq |\mathcal{G}|$ or until a resource limit is reached.

Regarding question (2), for two goal atoms $x \in \mathcal{G}$ and $y \in \mathcal{G}$, the most direct way to create a multi-goal abstraction for $\{x, y\}$ is to generate a planning task that is identical to the original except that the only goal atoms are x and y. It is easy to extend this to subsets of goal atoms: given two subsets $X \subseteq \mathcal{G}$ and $Y \subseteq \mathcal{G}$ of goal atoms, we denote the multi-goal Cartesian abstraction generated in this way by α_{XY}. However, refining an abstraction α_{XY} from scratch may require redoing much of the work already done to refine α_X and α_Y. As we will show next, this problem of redundant work can be alleviated by *merging* two already refined Cartesian abstractions, as opposed to refining an abstraction from scratch.

4 Merging Cartesian Abstractions

In this section, we introduce an efficient method for merging two Cartesian abstractions into a single Cartesian abstraction representing the synchronized product of the corresponding transition systems.[1]

Definition 1 (Product Cartesian Transition System). *Let Π be a planning task that induces the transition system $\mathcal{T} = \langle S, \mathcal{L}, T, s_0, S_\star \rangle$, and let α_A, α_B be two Cartesian abstractions that induce the abstract transition systems $\mathcal{T}^A = \langle S^A, \mathcal{L}, T^A, s_0^A, S_*^A \rangle$ and $\mathcal{T}^B = \langle S^B, \mathcal{L}, T^B, s_0^B, S_*^B \rangle$, respectively. We define the product of the two (Cartesian) transitions systems $\mathcal{T}^A \times \mathcal{T}^B$ as $\mathcal{T}^{A\times B} = \langle S^{A\times B}, \mathcal{L}, T^{A\times B}, s_0^{A\times B}, S_*^{A\times B} \rangle$ with the following components.*

- *The set of states $S^{A\times B}$ contains all non-empty intersections of all state pairs: $S^{A\times B} = \{a \cap b \mid a \in S^A, b \in S^B, a \cap b \neq \emptyset\}$.*
- *Labels $\mathcal{L}$ remain unchanged.*
- *There is a transition with label ℓ between two states $c = a \cap b$ and $c' = a' \cap b'$ if both are not empty and there is a transition labeled with ℓ in the corresponding transition systems between the considered states: $T^{A\times B} = \{(c, \ell, c') \mid c = a \cap b \neq \emptyset, c' = a' \cap b' \neq \emptyset, (a, \ell, a') \in T^A, (b, \ell, b') \in T^B\}$.*
- *The initial state s_0 is the intersection of the two individual initial states: $s_0^{A\times B} = s_0^A \cap s_0^B$.*
- *The set of goal states contains all non-empty intersection of all goal state pairs: $S_\star^{A\times B} = \{a \cap b \mid a \in S_\star^A, b \in S_\star^B, a \cap b \neq \emptyset\}$.*

The product transition system of two Cartesian abstractions α_A and α_B is induced by the *product Cartesian abstraction* $\alpha_{A\times B} : S \to S^{A\times B}$, where $\alpha_{A\times B}(s) = \alpha_A(s) \cap \alpha_B(s)$. This stems from the fact that, for any state $c = a \cap b \in S^{A\times B}$ and any concrete state $s \in S$ such that $\alpha_{A\times B}(s) = c$, we have that $\alpha_A(s) = a$ and $\alpha_B(s) = b$.

[1] Eifler and Fickert [8] mention such a product in their work on online abstraction refinement but do not define it nor show how to compute it.

4.1 Product Refinement Hierarchy

Since practical implementations of Cartesian CEGAR need to maintain the refinement hierarchy of each considered abstraction, we create a refinement hierarchy $H_{A\times B}$ from two given refinement hierarchies H_A and H_B, such that $H_{A\times B}$ represents the product abstraction of the two abstractions represented by H_A and H_B. Intuitively, we compute the product refinement hierarchy by applying all *relevant* splits done in one of the two hierarchies to the leaf nodes of the other. Let $H_A = \langle N_A, E_A \rangle$ and $H_B = \langle N_B, E_B \rangle$ be the refinement hierarchies of Cartesian abstractions α_A and α_B. We define the set of splits of H_B *affecting* a leaf a of H_A as:

$$\mathit{Splits}(H_B, a) = \{\langle b, b_l, b_r \rangle \in \mathit{Splits}(H_B) \mid a \cap b_l \neq \emptyset, a \cap b_r \neq \emptyset\}$$

Formally, we define the product of refinement hierarchies as follows.

Definition 2 (**Product Refinement Hierarchy**). *Let $H_A = \langle N_A, E_A \rangle$ and $H_B = \langle N_B, E_B \rangle$ be the refinement hierarchies of Cartesian abstractions α_A and α_B. Then, the* product refinement hierarchy *is $H_{A\times B} = \langle N_{A\times B}, E_{A\times B} \rangle$, where*

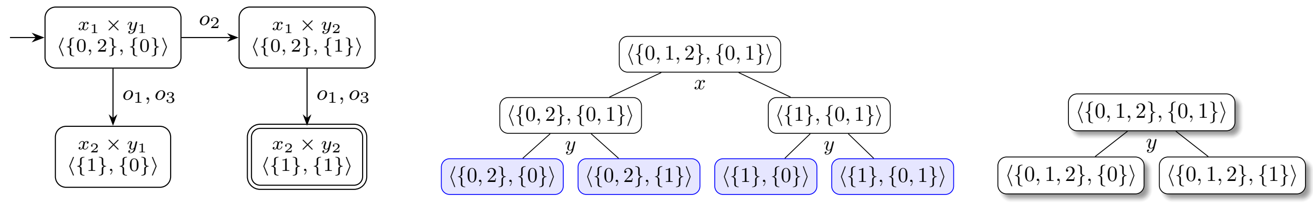
- *$N_{A\times B} = N_A \cup \{a \cap b_l, a \cap b_r \mid a \in \mathit{Leaves}(H_A), \langle b, b_l, b_r \rangle \in \mathit{Splits}(H_B, a)\}$, and*
- *$E_{A\times B} = E_A \cup \{\langle a \cap b, a \cap b_l \rangle, \langle a \cap b, a \cap b_r \rangle \mid a \in \mathit{Leaves}(H_A), \langle b, b_l, b_r \rangle \in \mathit{Splits}(H_B, a)\}$.*

Example 1 (**Product Abstraction and Refinement Hierarchy**). Consider the following task: $\mathcal{V} = \{x, y\}$, $\mathcal{D}(x) = \{0, 1, 2\}$, $\mathcal{D}(y) = \{0, 1\}$, $\mathcal{O} = \{o_1, o_2, o_3\}$ with $o_1 = \langle \{x \mapsto 0\}, \{x \mapsto 1\} \rangle$, $o_2 = \langle \{x \mapsto 0, y \mapsto 0\}, \{x \mapsto 2, y \mapsto 1\} \rangle$ and $o_3 = \langle \{x \mapsto 2\}, \{x \mapsto 1\} \rangle$, $s_0 = \{x \mapsto 0, y \mapsto 0\}$ and $s_\star = \{x \mapsto 1, y \mapsto 1\}$. Let α_X and α_Y be the single-goal abstractions for x and y, respectively. The transition system $\mathcal{T}^X$ induced by α_X has 2 states: $x_1 = \langle \{0, 2\}, \{0, 1\} \rangle$ and $x_2 = \langle \{1\}, \{0, 1\} \rangle$. The one induced by α_Y, $\mathcal{T}^Y$, has 2 states: $y_1 = \{0, 1, 2\} \times \{0\}$ and $y_2 = \{0, 1, 2\} \times \{1\}$. The product Cartesian transition system $\mathcal{T}^{X\times Y}$ induced by the product abstraction $\alpha_{X\times Y}$ is depicted in Figure 1a. The product refinement hierarchy $H_{X\times Y}$ is shown in Figure 1b. It is the result of applying the splits in H_Y (shown in Figure 1c) to the leaves of H_X (white part of Figure 1b).

Next, we prove that Definition 2 exactly captures the product of the underlying abstractions.

Theorem 1. *Let $H_A = \langle N_A, E_A \rangle$ and $H = \langle N_B, E_B \rangle$ be two refinement hierarchies of abstractions α_A and α_B inducing transition systems $\mathcal{T}^A$ and $\mathcal{T}^B$. Then, (i) $H_{A\times B}$ obtained with Definition 2 is a refinement hierarchy and (ii) its leaves are exactly the set of abstract states in $S^{A\times B}$.*

Proof. (i) $H_{A\times B}$ is a binary tree by definition, and for any split $s = \langle n, n_l, n_r \rangle \in \mathit{Splits}(H_{A\times B})$ with $\mathit{split\_var}(n) = v$, we have to show that $\mathcal{D}(v, n_l) \cap \mathcal{D}(v, n_r) = \emptyset$, $\mathcal{D}(v, n_l) \cup \mathcal{D}(v, n_r) = \mathcal{D}(v, n)$, and $\mathcal{D}(w, n_l) = \mathcal{D}(w, n_r) = \mathcal{D}(w, n)$ for all $w \neq v$. If $s \in \mathit{Splits}(H_A)$, then all conditions are satisfied since H_A is a refinement hierarchy. Otherwise, $s = \langle a \cap b, a \cap b_l, a \cap b_r \rangle$, with $a \in N_A$ and $\langle b, b_l, b_r \rangle \in \mathit{Splits}(H_B, a)$. Since $\langle b, b_l, b_r \rangle$ is a split of H_B, we have that $\mathcal{D}(v, b_l) \cap \mathcal{D}(v, b_r) = \emptyset$, $\mathcal{D}(v, b_l) \cup \mathcal{D}(v, b_r) = \mathcal{D}(v, b)$, and $\mathcal{D}(w, b_l) = \mathcal{D}(w, b_r) = \mathcal{D}(w, b)$ for all $w \neq v$. Then, it follows that $a \cap b_l$ and $a \cap b_r$ only differ in the domains of $\mathit{split\_var}(b)$, and all conditions on the split $s = \langle a \cap b, a \cap b_l, a \cap b_r \rangle$ are satisfied.
(ii) Let $a \in S^A$ and $b \in S^B$. Then $c = a \cap b$ is a leaf node of $H_{A\times B}$: $N_{A\times B}$ contains all non-empty intersections between nodes of H_A and H_B, and since a and b are leaves of H_A and H_B, respectively, then c has no children in $H_{A\times B}$ by definition. □



$x_1 \times y_1$
$\langle\{0,2\},\{0\}\rangle$
o_2
$x_1 \times y_2$
$\langle\{0,2\},\{1\}\rangle$
o_1, o_3
o_1, o_3
$x_2 \times y_1$
$\langle\{1\},\{0\}\rangle$
$x_2 \times y_2$
$\langle\{1\},\{1\}\rangle$
$\langle\{0,1,2\},\{0,1\}\rangle$
x
$\langle\{0,2\},\{0,1\}\rangle$
$\langle\{1\},\{0,1\}\rangle$
y
y
$\langle\{0,2\},\{0\}\rangle$
$\langle\{0,2\},\{1\}\rangle$
$\langle\{1\},\{0\}\rangle$
$\langle\{1\},\{0,1\}\rangle$
$\langle\{0,1,2\},\{0,1\}\rangle$
y
$\langle\{0,1,2\},\{0\}\rangle$
$\langle\{0,1,2\},\{1\}\rangle$

(a) Product Cartesian transition system $\mathcal{T}^{X\times Y}$ induced by $\alpha_{X\times Y}$.

(b) Refinement hierarchy for α_X in white. Changes for product refinement hierarchy $\alpha_{X\times Y}$ in blue.

(c) Refinement hierarchy for α_Y.

Figure 1: Product Cartesian transition system and refinement hierarchies for the single goal abstractions and their product for Example 1.

Algorithm 1 Compute the product refinement hierarchy for the refinement hierarchies H_A and H_B associated with two Cartesian abstractions.

```
 1: function ComputeHierarchy($H_A, H_B$)
 2:     for $a \in Leaves(H_A)$ do
 3:         AddSplits($a$, Root($H_B$))
        return $H_A$
 4: function AddSplits($a, b$)
 5:     if $b$ is not a leaf node then
 6:         $b_l, b_r \leftarrow children(b)$
 7:         $v \leftarrow split\_var(b)$
 8:         if $\mathcal{D}(v, a) \subseteq \mathcal{D}(v, b_l)$ then
 9:             AddSplits($a, b_l$)
10:         else if $\mathcal{D}(v, a) \subseteq \mathcal{D}(v, b_r)$ then
11:             AddSplits($a, b_r$)
12:         else
13:             $split\_var(a) \leftarrow v$
14:             $children(a) \leftarrow \langle a \cap b_l, a \cap b_r \rangle$
15:             AddSplits($a \cap b_l, b_l$)
16:             AddSplits($a \cap b_r, b_r$)
```

Theorem 1 guarantees that the product Cartesian abstraction $\alpha_{A\times B}$ inducing the product Cartesian transition system $\mathcal{T}^{A\times B}$ obtained by Definition 1 is in fact a Cartesian abstraction of $\mathcal{T}$, and that $\mathcal{T}^{A\times B}$ is equal to one of the factors or more fine-grained than both.

Algorithm 1 shows how to efficiently create the product refinement hierarchy from two given hierarchies according to Definition 2. It takes as input two refinement hierarchies H_A and H_B representing abstractions α_A and α_B with states S^A and S^B. For each leaf node a of H_A, i.e. for each state in S^A of the abstraction α_A, all refinements of the abstraction α_B are applied (lines 2–3). Function AddSplits(a,b) recursively applies to a all the splits affecting it from the subtree of H_B rooted at b. For that, it recursively calls AddSplits(a,b_l) and AddSplits(a,b_r), where b_l, b_r are the children of b. If b is a leaf node of H_B (i.e., a state in S^B), no further splits are performed, and it means that a leaf node (abstract state in the product abstraction) has been created. Otherwise, we get the left and right children of b, as well as the variable v were the split was done (lines 6–7). If $\mathcal{D}(v, a)$ is a subset of either $\mathcal{D}(v, b_l)$ or $\mathcal{D}(v, b_r)$, then the split done in b cannot be applied in a, and the refinement continues down the appropriate branch (lines 8–12). Otherwise, the split is applied, two new children are added to a and the refinement continues for these new nodes (lines 13–16).

4.2 Refinement From Scratch vs. From the Product

We call an abstraction *fully refined* if its induced transition system has an optimal concrete plan. The product of two Cartesian abstractions is not necessarily a fully refined abstraction. Thus, after computing the product, further refinements might be required to obtain a fully refined abstraction. This raises the question whether there exists a relation between multi-goal Cartesian abstractions obtained from the product of abstractions (and then refined) and those obtained purely from refinement (considering multiple goal atoms from the start).

Proposition 1. *Let α_A, α_B be two Cartesian abstractions that induce the abstract transition systems $\mathcal{T}^A$ and $\mathcal{T}^B$, respectively. The product $\mathcal{T}^{A\times B}$ can be obtained with Cartesian abstraction refinements starting from the trivial abstraction.*

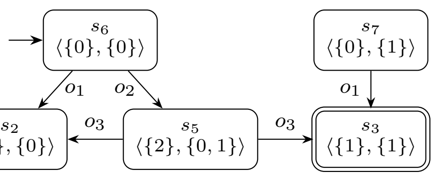
This follows directly from Theorem 1, since Definition 2 defines a refinement hierarchy whose leaf nodes are the states in $S^{A\times B}$.

Theorem 2. *Let $\alpha_{A\times B}$ be a fully refined Cartesian abstraction obtained from refining (if needed) the product abstraction of the Cartesian abstractions α_A and α_B. Then, there exists a sequence of refinements that, starting from the trivial abstraction, produces a fully refined abstraction α_{AB} such that $|S^{AB}| \leq |S^{A\times B}|$.*

Proof. This follows directly from Proposition 1: $\alpha_{A\times B}$ can be obtained with a sequence of refinements, and any other fully refined abstraction computed within the Cartesian CEGAR framework can be obtained via a sequence of refinements. □

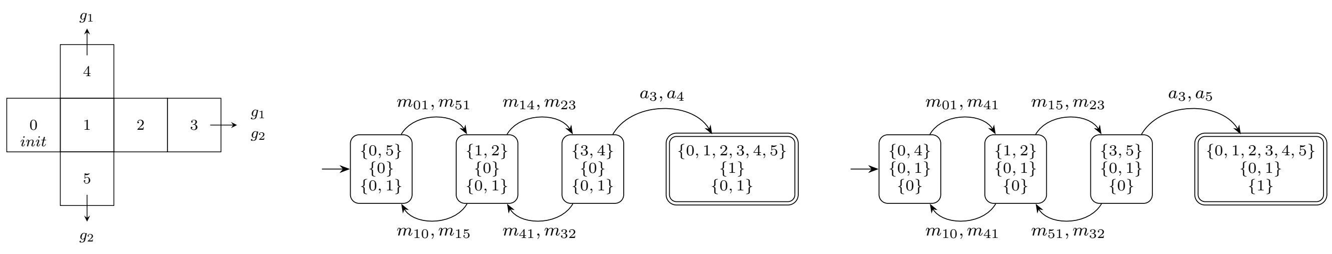
However, in practice, $\alpha_{A\times B}$ can be smaller or larger than α_{AB}, since the optimal refinement strategy for obtaining a minimally-sized abstraction is unknown. Our experiments show that $\alpha_{A\times B}$ is often smaller than α_{AB}, but we now show two examples where $\alpha_{A\times B}$ is smaller than α_{AB}, and vice versa.

Example 2 (**Product Abstraction is Smaller**)**.** Consider again Example 1. Assume the optimal abstract plan found for $\alpha_{X\times Y}$ (in Figure 1a) is o_2, o_3, which is a concrete plan. Then, there is no need for any refinement. The from-scratch abstraction α_{XY} is built by starting with a single state that contains all values for all variables. Assume that we start by splitting off the goal facts, generating 3 states: $s_1 = \langle\{0,2\},\{0,1\}\rangle$, $s_2 = \langle\{1\},\{0\}\rangle$, and $s_3 = \langle\{1\},\{1\}\rangle$. All optimal plans for this abstraction have length one (apply either o_1 or



s_6
$\langle\{0\},\{0\}\rangle$
s_7
$\langle\{0\},\{1\}\rangle$
o_1
o_2
o_1
s_2
$\langle\{1\},\{0\}\rangle$
o_3
s_5
$\langle\{2\},\{0,1\}\rangle$
o_3
s_3
$\langle\{1\},\{1\}\rangle$

Figure 2: Transition system induced by the abstraction α_{XY} resulting from refining from scratch for Example 2.



(a) Example task. (b) Transition system induced by α_{G_1}. (c) Transition system induced by α_{G_2}.

Figure 3: Visualization of task and transition systems induced by single goal abstractions for Example 3.

o_3 to go from s_1 to s_3). Neither of them is a concrete plan, so α_{XY} needs to be refined. The next refinement step splits s_1 for the values of one variable, and there are two alternatives: choose either x or y. This is decided by a *flaw selection strategy* [35, 23, 24]. If y is selected, then, after refinement, the next abstraction would be exactly the product abstraction. However, assume that x is selected, in which case the next abstraction contains 4 states: the two states resulting from the split, $s_4 = \langle\{0\}, \{0, 1\}\rangle$ and $s_5 = \langle\{2\}, \{0, 1\}\rangle$, and states s_2 and s_3, which remain unchanged. Again, the optimal abstract plan, which is now $\langle o_1\rangle$, is not a concrete plan, so another refinement step is required. Assume it splits the state s_4 for y, generating $s_6 = \langle\{0\}, \{0\}\rangle$ and $s_7 = \langle\{0\}, \{1\}\rangle$. By now, the refined abstraction has 5 states (depicted in Figure 2), with optimal plan $\langle o_2, o_3\rangle$, which is a concrete plan. Therefore, for this example, the refined-from-scratch abstraction α_{XY} has more states than the product abstraction $\alpha_{X\times Y}$.

Example 3 (**Refined-From-Scratch Abstraction is Smaller**). Consider now the example task shown in Figure 3a, where an agent situated at position 0 has to achieve g_1 and g_2. There are three variables representing the position of the agent, and whether each goal g_1 and g_2 has been achieved: $\mathcal{V} = \{pos, g_1, g_2\}$ with $\mathcal{D}(pos) = \{0, \dots, 5\}$ and $\mathcal{D}(g_i) = \{0, 1\}$. There are operators m_{ij} to move between adjacent cells horizontally and vertically, and three operators to achieve goals: $a_3 = \langle\{pos \mapsto 3\}, \{g_1 \mapsto 1\, g_2 \mapsto 1\}\rangle$, $a_4 = \langle\{pos \mapsto 4\}, \{g_1 \mapsto 1\}\rangle$, and $a_5 = \langle\{pos \mapsto 5\}, \{g_2 \mapsto 1\}\rangle$. The single goal abstractions, α_{G_1} and α_{G_2}, are depicted in Figure 3. They separate the states in the shortest path to each goal, paths 0-1-4-g_1 and 0-2-5-g_2, respectively. It is easy to see that the product abstraction has 12 states, since it separates the states on the two paths that achieve one goal before the other: 0-1-4-g_1-1-5-g_2 and 0-1-5-g_2-1-4-g_1. It also represents the path from 0 to 3 to achieve both goals, but positions 1 and 2 are mapped to the same abstract state ($\langle\{1, 2\}, \{0\}, \{0\}\rangle$). Thus, it needs to be refined, resulting in an abstraction with 13 states, which contains the optimal concrete path 0-1-2-3-g_1g_2. Figure 4 shows a possible refined abstraction for the task with both goals. It does not separate the states in the paths that achieve one goal before the other completely. Those paths have to be refined only to have at least length 4, the length of the optimal concrete path. Then, the minimum size of this abstraction is 9, which is smaller than the size of the refined product abstraction. (Note that, depending on the refinement strategy, it could also be larger.) If the corridors are enlarged while maintaining the proportion, the difference between the sizes of the two abstractions becomes larger.

5 Multi-Goal Abstractions and Cost Partitioning

After computing a set of Cartesian abstractions, we combine their estimates within a cost partitioning heuristic [e.g., 17]. Since a product

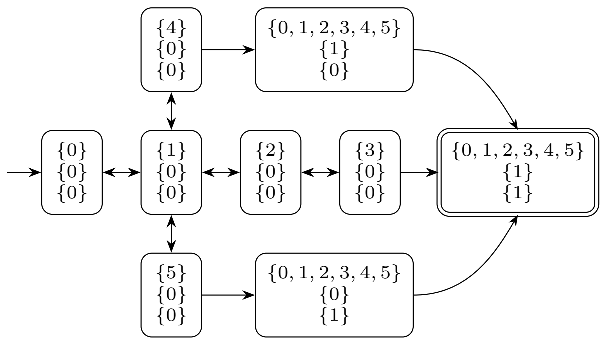


Figure 4: Transition system induced by the abstraction $\alpha_{G_1G_2}$ for the task in Example 3. We omit transition labels for simplicity.

abstraction is more fine-grained than either of the factors (i.e., the factors are abstractions of the product), one might wonder whether it is beneficial to keep abstractions that were used to compute a product, or if they can be dropped without affecting the heuristic values of a cost partitioning heuristic. Under optimal cost partitioning (OCP) [16], merged factors can be discarded.

Theorem 3. *Let $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ be a set of Cartesian abstractions. For any α_i and α_j in $\mathcal{A}$, let $\mathcal{B} = (\mathcal{A} \setminus \{\alpha_i, \alpha_j\}) \cup \alpha_{i\times j}$ be the set of abstractions where α_i and α_j have been replaced by their fully refined product abstraction $\alpha_{i\times j}$. Then $h^{OCP}_{\mathcal{A}} \leq h^{OCP}_{\mathcal{B}}$.*

Proof. This follows from Theorem 5 by Sievers et al. [34], who show that merging two factors in a merge-and-shrink abstraction and discarding the two merged factors can only benefit OCP, and the fact that any further refinements for $\alpha_{i\times j}$—until it is fully refined—can only increase the heuristic value computed by OCP. ☐

However, for saturated post-hoc optimization this does not hold and discarding merged factors can lead to a decrease in the heuristic value.

Theorem 4. *There are sets of Cartesian abstractions $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ is obtained by replacing two abstractions in $\mathcal{A}$ by their fully refined product abstraction, such that $h^{\text{SPhO}}_{\mathcal{B}} < h^{\text{SPhO}}_{\mathcal{A}}$.*

Proof. Consider the SPhO LP for the following example: $\mathcal{V} = \{x, y, z\}$, $\mathcal{O} = \{o_1, o_2, o_3\}$ with $o_1 = \langle\{x \mapsto 0\}, \{x \mapsto 1\}\rangle$, $o_2 = \langle\{y \mapsto 0\}, \{y \mapsto 1\}\rangle$ and $o_3 = \langle\{x \mapsto 0, y \mapsto 1\}, \{z \mapsto 1\}\rangle$. $s_0 = \{x \mapsto 0, y \mapsto 0, z \mapsto 0\}$ and $s_\star = \{x \mapsto 1, y \mapsto 1, z \mapsto 1\}$. Let α_X, α_Y and α_Z be the single-goal abstractions for x, y and z, respectively. Assume unit cost for all operators. The SPhO LP for the single-goal abstractions and the initial state is

$$\begin{aligned} \text{maximize} \quad & 1 \cdot w_X + 1 \cdot w_Y + 2 \cdot w_Z \quad \text{subject to} \\ & w_X \leq 1 \\ & w_Y + w_Z \leq 1 \\ & w_Z \leq 1 \\ & w \geq 0 \text{ for all } w \in \{w_X, w_Y, w_Z\}, \end{aligned}$$

and the optimal objective value is 3. Now consider a multi-goal abstraction α_{XY} for x and y and assume that we replace α_X by α_{XY}. The resulting LP will be the same, regardless of whether we obtain α_{XY} by merging α_X and α_Y (yielding an abstraction with 4 states) or by refining for the two goal atoms from scratch (yielding an abstraction with 3 states, regardless of the considered goal order). The SPhO LP for the initial state becomes

$$\begin{aligned} \text{maximize} \quad & 2 \cdot w_{XY} + 1 \cdot w_Y + 2 \cdot w_Z \quad \text{subject to} \\ & w_{XY} \leq 1 \\ & w_{XY} + w_Y + w_Z \leq 1 \\ & w_Z \leq 1 \\ & w \geq 0 \text{ for all } w \in \{w_{XY}, w_Y, w_Z\}, \end{aligned}$$

and the optimal objective value decreases to 2. This value remains the same if we also remove abstraction Y from the LP. □

The final cost partitioning algorithm that we consider for this analysis is saturated cost partitioning (SCP) [31]. SCP considers an ordered sequence ω of heuristics and greedily assigns to each heuristic as much cost as that heuristic needs to preserve its estimates, i.e., its *saturated cost function* (*scf*), and saves the remaining costs for subsequent heuristics, until all heuristics have been served this way. Due to its greedy nature, adding product abstractions to the set of heuristic considered by saturated cost partitioning can decrease its estimate, even when keeping all merged factors.

Theorem 5. *There are sets of Cartesian abstraction heuristics $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, $\mathcal{B} = \mathcal{A} \cup \{\alpha_{i \times j}\}$, and orders $\omega_\mathcal{A}$, $\omega_\mathcal{B}$ over $\mathcal{A}$ and $\mathcal{B}$, such that $\alpha_{i \times j}$ is the fully refined product abstraction of two abstractions α_i and α_j in $\mathcal{A}$, and $h^{\text{SCP}}_{\omega_\mathcal{B}} < h^{\text{SCP}}_{\omega_\mathcal{A}}$.*

Proof. Consider the example from the proof of Theorem 4, but with operators $o_1 = \langle \{x \mapsto 0\}, \{x \mapsto 1\} \rangle$, $o_2 = \langle \{x \mapsto 1\}, \{x \mapsto 0, y \mapsto 1\} \rangle$ and $o_3 = \langle \{x \mapsto 1, y \mapsto 1\}, \{z \mapsto 1\} \rangle$. The saturated cost functions ($scf(o_1)$, $scf(o_2)$, $scf(o_3)$) for the single goal abstractions are $\langle 1, 0, 0 \rangle$ for α_X, $\langle 1, 1, 0 \rangle$ for α_Y, and $\langle 1, 1, 1 \rangle$ for α_Z. For the order of single-goal abstractions $\omega = \langle \alpha_Z, \alpha_Y, \alpha_X \rangle$, abstraction α_Z uses all costs and the remaining costs for α_Y and α_X are all zero. The value of the SCP heuristic for s_0 is 4, the length of the optimal plan $\langle o_1, o_2, o_1, o_3 \rangle$, which is also the optimal plan for α_Z. Now add abstraction α_{ZX}, and consider the order $\omega = \langle \alpha_X, \alpha_Y, \alpha_Z, \alpha_{ZX} \rangle$. With this order, α_X uses the cost of o_1 and α_Y uses the cost of o_2. So, the only remaining cost for α_Z is that of o_3, and the remaining costs for α_{ZX} are all zero. Then, the value of the SCP heuristic for s_0 is 3, since plans for X only use o_1 once. □

Given this limitation of saturated cost partitioning for multi-goal abstractions, we exclusively use saturated post-hoc optimization in the experiments below and always keep all abstractions. We leave it as future work to investigate how to decide which abstractions to keep for SCP and how to guarantee that SCP can only benefit from adding multi-goal abstractions.

6 Experiments

We implemented multi-goal abstractions in the Scorpion planner [31], which extends Fast Downward [13]. As search algorithm, we use A* [11] with an offline saturated post-hoc optimization heuristic [15] computed over single- and multi-goal abstractions constructed in different ways.

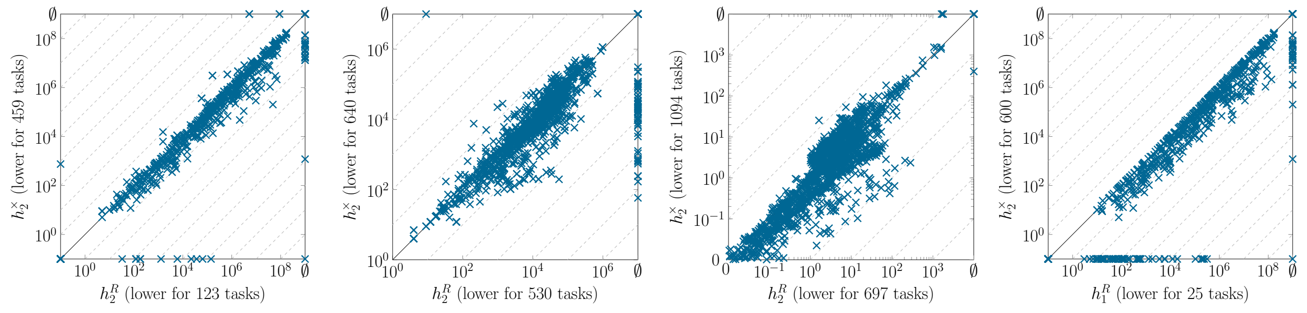
To construct the multi-goal abstractions, we use two different approaches to compare the effectiveness of merging abstractions versus building them from scratch: (1) $h^\times_n$: follows the process described in Section 3 to merge and refine abstractions of up to n goal atoms; (2) h^R_n: refines multi-goal abstractions ordered by the number of goal atoms, up to n goal atoms, starting from the trivial abstraction. As baselines, we consider single-goal Cartesian abstractions (h^R_1) and the variant that creates a single abstraction for all goal atoms ($h^\mathcal{G}$). Finally, to see how multi-goal abstractions can improve state-of-the-art abstraction heuristics, we test a configuration where we compute both Cartesian abstractions induced by goals and landmarks. Fact landmarks are atoms that must be true at some point in any plan [14], and we use the abstractions induced by landmarks as per Seipp and Helmert [30], who generate landmarks of the delete relaxation of the task [18]. h^L_1 uses single-goal abstractions and h^L_2 uses 2-goal abstractions (computed as for $h^\times_2$) along with landmark induced abstractions.

We evaluate the different configurations on all tasks of the optimal tracks from the International Planning Competitions (IPC) from 1998 to 2018. All experiments were run on machines with an Intel Xeon Gold 6130 processor at 2.10 GHz, with total time and memory limits for each run of 30 minutes and 8 GiB. As is common practice, we also limit the resources used to construct the abstractions by imposing a cumulative limit of 10 million transitions across all abstractions for each individual run. When we reach the transition limit, we compute the cost partitioning over the constructed abstractions and start the A* search with the resulting heuristic. All benchmarks, code and data are available online [25].

Table 1 shows overall coverage scores, as well as in how many domains one configuration solves more tasks than another. As reported in the literature [30], considering a single abstraction with all goal atoms ($h^\mathcal{G}$) performs worse than decomposing the original task, building multiple abstractions, and combining them with cost partitioning (e.g., h^R_1). However, there are some tasks that $h^\mathcal{G}$ solves that none of the other configurations do, implying that some information about the interaction between the goal atoms is lost when creating multiple abstractions, even when considering multi-goal abstractions.

Moving to the performance of our proposed multi-goal abstractions, we can observe that multi-goal abstractions are beneficial compared to single-goal abstractions both when they are refined from scratch and when they are merged using Algorithm 1. The effectiveness and importance of merging abstractions is evident when considering that $h^\times_n$ solves more tasks than h^R_n, for all n tested. This empirically confirms our conjecture that refining a multi-goal abstraction from scratch, which is the product of two abstractions, can introduce redundant work and overhead that can be eliminated by directly merging the two corresponding abstractions. To better understand the reason for the difference in performance between from-scratch refinement h^R_n and merge and refine $h^\times_n$, consider Figure 5, which compares the number of expanded nodes (5a), the size of the resulting abstractions (5b), and the time taken to start the search between $h^\times_2$ and h^R_2 (5c).

Figure 5a shows that $h^\times_2$ is more efficient in terms of the number of expanded nodes, meaning that the abstractions obtained from the product of abstractions (and then refined) are generally more informative than abstractions refined from scratch. This seems to indicate that



(a) Expansions until last f-layer between h_2^R and $h_2^\times$.

(b) States in multi-goal abstractions for h_2^R and $h_2^\times$.

(c) Time for building abstractions for h_2^R and $h_2^\times$.

(d) Expansions until last f-layer between $h_2^\times$ and h_1^R.

Figure 5: Comparison between different configurations of h_n^R and $h_n^\times$. Tasks unsolved within the resource limits are marked with $\emptyset$.

		Refine					Merge & Refine				+Landmarks	
		$h^\mathcal{G}$	h_1^R	h_2^R	h_3^R	h_4^R	$h_2^\times$	$h_3^\times$	$h_4^\times$	$h_\infty^\times$	h_1^L	h_2^L
Refine	$h^\mathcal{G}$	–	11	7	7	6	6	5	5	5	10	6
	h_1^R	**14**	–	0	1	0	0	0	0	0	1	3
	h_2^R	**20**	**12**	–	1	2	2	3	3	3	9	7
	h_3^R	**22**	**14**	**5**	–	**3**	6	6	6	6	10	10
	h_4^R	**21**	**12**	**6**	**3**	–	4	3	3	3	8	9
M & R	$h_2^\times$	**25**	**19**	**14**	**12**	**11**	–	3	**3**	**3**	**11**	6
	$h_3^\times$	**25**	**20**	**14**	**13**	**12**	**4**	–	**1**	**2**	**12**	8
	$h_4^\times$	**25**	**20**	**14**	**13**	**12**	**3**	0	–	**1**	**12**	8
	$h_\infty^\times$	**24**	**19**	**13**	**12**	**11**	2	0	0	–	**11**	7
+L	h_1^L	**22**	**13**	**11**	**11**	**11**	9	8	8	8	–	2
	h_2^L	**24**	**19**	**16**	**14**	**15**	**9**	**11**	**11**	**11**	**11**	–
	C	844	853	868	875	875	888	889	888	887	1009	1026

Table 1: Coverage comparison for different sets of abstractions. The cell (r, c) shows the number of domains in which r solves more tasks than c. For each pair of algorithms we highlight the maximum of entries (r, c) and (c, r) in bold. The last row shows the total number of solved tasks for each variation. h_n^R refers to configurations that refine multi-goal abstractions from scratch, while $h_n^\times$ refers to configurations that merge abstractions. $h^\mathcal{G}$ refers to the configuration that creates a single abstraction with all goal atoms. h_1^L and h_2^L refer to the configurations that use single-goal and two-goal abstractions, respectively, in conjunction with landmark induced abstractions.

information gained from refinements done for single-goal abstractions is not necessarily obtained when refining a multi-goal abstraction from scratch, where the goals are a superset of the single-goal abstraction, and that this information is useful during search. Nevertheless, both the size of the abstractions and the time needed to refine the abstractions are generally smaller for $h_2^\times$. This is surprising at first glance: intuitively, one expects larger abstractions to provide more information during search, but by considering the product of abstractions, we manage to find more informative and smaller abstractions under SPhO. However, the improvement from increasing the number of goals considered for both $h_n^\times$ and h_n^R is limited, as shown in Table 1. This is because the abstractions start to grow too fast, and we hit the transition limit of 10 million transitions, which limits the amount of information we can get from more abstractions. We experimented with raising this limit and found that it does not readily improve performance, because the computational cost of refining larger abstractions reduces the total number of tasks solved.

Figure 5d shows the number of expanded nodes for $h_2^\times$ and h_1^R, showing that considering the 2-goal abstraction can greatly reduce the number of expanded nodes, in some cases by several orders of magnitude. The few cases where h_1^R expands fewer nodes are due to the fact that we are using offline SPhO. Using online SPhO would guarantee that $h_2^\times$ dominates h_1^R, but preliminary experiments showed that the overhead of using online SPhO is too high, greatly reducing coverage for all configurations.

Finally, h_2^L solves more tasks than h_1^L (Table 1), demonstrating that the use of multi-goal abstractions is beneficial even when combined with abstractions not induced by goal atoms, thereby improving the current state of the art.

7 Online vs. Offline Refinement

Eifler and Fickert [8] improve Cartesian abstractions during search by merging abstractions. In the online setting, the merge can occur at any point. In contrast, we only consider merging abstractions that are fully refined from the initial state. The different settings make it difficult to empirically compare the approaches in a meaningful way. However, Eifler and Fickert [8] report that merging abstractions is computationally expensive, while our approach balances merging effort with heuristic accuracy. Furthermore, efficient merging of Cartesian abstractions is crucial, which we explicitly address by using refinement hierarchies as underlying data structures.

8 Conclusions

We introduced an efficient algorithm for computing multi-goal Cartesian abstractions. In addition, we showed that creating multi-goal abstractions starting from the product of abstractions can significantly speed up the refinement process compared to refining the multi-goal abstraction from scratch. We showed that having additional abstractions can only increase the heuristic value for saturated post-hoc optimization, but not for saturated cost partitioning. Our experiments show that using multi-goal abstractions generates more informative heuristics than using single-goal abstractions alone, which is reflected in solving many more tasks.

As future work, we plan to go beyond the systematic generation of multi-goal abstractions by exploring strategies for selecting which abstractions to merge, similar to merging strategies for merge-and-shrink heuristics [33]. In addition, we want to investigate ways to guarantee that the use of multi-goal abstractions under saturated cost partitioning can never decrease the heuristic value.

Acknowledgements

This work was partially supported by grant PID2021-127647NB-C21 funded by MICIU/AEI/10.13039/501100011033 and by "ERDF A way of making Europe". Additionally, this work was partially supported by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA) and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

[1] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[2] T. Ball, A. Podelski, and S. K. Rajamani. Boolean and Cartesian abstraction for model checking C programs. In T. Margaria and W. Yi, editors, *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *Lecture Notes in Computer Science*, pages 268–283. Springer-Verlag, 2001.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

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

[5] D. Drexler, J. Seipp, and D. Speck. Subset-saturated transition cost partitioning. In Goldman et al. [10], pages 131–139.

[6] S. Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 84–90. AAAI Press, 2001.

[7] S. Edelkamp. Automated creation of pattern database search heuristics. In S. Edelkamp and A. Lomuscio, editors, *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, pages 35–50, 2006.

[8] R. Eifler and M. Fickert. Online refinement of Cartesian abstraction heuristics. In V. Bulitko and S. Storandt, editors, *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, pages 46–54. AAAI Press, 2018.

[9] S. Franco, Á. Torralba, L. H. S. Lelis, and M. Barley. On creating complementary pattern databases. In C. Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pages 4302–4309. IJCAI, 2017.

[10] R. P. Goldman, S. Biundo, and M. Katz, editors. *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 2021. AAAI Press.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 1007–1012. AAAI Press, 2007.

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

[14] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[15] P. Höft, D. Speck, and J. Seipp. Sensitivity analysis for saturated post-hoc optimization in classical planning. In K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, and R. Rădulescu, editors, *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, pages 1044–1051. IOS Press, 2023.

[16] M. Katz and C. Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In J. Rintanen, B. Nebel, J. C. Beck, and E. Hansen, editors, *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 174–181. AAAI Press, 2008.

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

[18] E. Keyder, S. Richter, and M. Helmert. Sound and complete landmarks for and/or graphs. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 335–340. IOS Press, 2010.

[19] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[20] F. Pommerening, G. Röger, and M. Helmert. Getting the most out of pattern databases for classical planning. In F. Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2357–2364. AAAI Press, 2013.

[21] F. Pommerening, M. Helmert, G. Röger, and J. Seipp. From non-negative to general operator cost partitioning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pages 3335–3341. AAAI Press, 2015.

[22] F. Pommerening, T. Keller, V. Halasi, J. Seipp, S. Sievers, and M. Helmert. Dantzig-Wolfe decomposition for cost partitioning. In Goldman et al. [10], pages 271–280.

[23] M. Pozo, Á. Torralba, and C. Linares López. When CEGAR meets regression: A love story in optimal classical planning. In J. Dy and S. Natarajan, editors, *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, pages 20238–20246. AAAI Press, 2024.

[24] M. Pozo, Á. Torralba, and C. Linares López. Gotta catch 'em all! sequence flaws in CEGAR for classical planning. In U. Endriss and F. S. Melo, editors, *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI 2024)*, pages 4287–4294. IOS Press, 2024.

[25] M. Salerno, R. Fuentetaja, D. Speck, and J. Seipp. Code and data for the ECAI 2025 paper "Merging Cartesian Abstractions for Classical Planning". https://doi.org/10.5281/zenodo.16311148, 2025.

[26] J. Seipp. Pattern selection for optimal classical planning with saturated cost partitioning. In S. Kraus, editor, *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 5621–5627. IJCAI, 2019.

[27] J. Seipp. Efficiently computing transitions in Cartesian abstractions. In S. Bernardini and C. Muise, editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, pages 509–513. AAAI Press, 2024.

[28] J. Seipp and M. Helmert. Counterexample-guided Cartesian abstraction refinement. In D. Borrajo, S. Kambhampati, A. Oddi, and S. Fratini, editors, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 347–351. AAAI Press, 2013.

[29] J. Seipp and M. Helmert. Diverse and additive Cartesian abstraction heuristics. In S. Chien, A. Fern, W. Ruml, and M. Do, editors, *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pages 289–297. AAAI Press, 2014.

[30] J. Seipp and M. Helmert. Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, 62:535–577, 2018.

[31] J. Seipp, T. Keller, and M. Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67:129–167, 2020.

[32] J. Seipp, T. Keller, and M. Helmert. Saturated post-hoc optimization for classical planning. In K. Leyton-Brown and Mausam, editors, *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 11947–11953. AAAI Press, 2021.

[33] S. Sievers and M. Helmert. Merge-and-shrink: A compositional theory of transformations of factored transition systems. *Journal of Artificial Intelligence Research*, 71:781–883, 2021.

[34] S. Sievers, F. Pommerening, T. Keller, and M. Helmert. Cost-partitioned merge-and-shrink heuristics for optimal classical planning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, pages 4152–4160. IJCAI, 2020.

[35] D. Speck and J. Seipp. New refinement strategies for Cartesian abstractions. In S. Thiébaux and W. Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pages 348–352. AAAI Press, 2022.

[36] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, and J. Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024. doi: 10.1002/aaai.12169.

[37] F. Yang, J. Culberson, R. Holte, U. Zahavi, and A. Felner. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32:631–662, 2008.