Not Everything is Permitted: Constrained Cartesian Abstractions for Optimal Classical Planning

Martín Pozo[1], **Álvaro Torralba**[2], **Carlos Linares López**[1]

[1]Computer Science and Engineering Department, Universidad Carlos III de Madrid, Madrid, Spain
[2]Department of Computer Science, Aalborg University, Aalborg, Denmark
marcosmartin.pozo@uc3m.es, alto@cs.aau.dk, carlos.linares@uc3m.es

Abstract

Cartesian abstractions can flexibly approximate planning tasks to generate admissible heuristic functions. Constrained abstractions use state constraints, such as mutexes, to eliminate parts of the abstraction that cannot belong to solutions for the original problem. While this has been successfully applied to simple forms of abstraction, no previous work has explored how to do this for Cartesian abstractions.

We introduce constrained Cartesian abstractions, which leverage state constraints in multiple ways: to prune spurious transitions and to simplify or even remove abstract states. Moreover, we also use disambiguation to better guide the counterexample-guided process used to generate the abstractions. Our experimental results show that the resulting constrained Cartesian abstractions induce more informed heuristics than their non-constrained counterpart.

Introduction

Abstractions are popular techniques for optimal planning because the admissible heuristics that they induce are informed and flexible (Edelkamp 2001; Helmert et al. 2014; Sievers and Helmert 2021). Cartesian abstractions are very fine-grained and powerful, but to the best of our knowledge no work has applied state constraints to them yet, but only in abstractions for PDBs (Haslum, Bonet, and Geffner 2005) and in a different method to create abstractions that also removes spurious paths (Fan and Holte 2015). A common constraint are state invariants, such as mutexes, stating that a certain set of facts cannot be part of any reachable state (Bonet and Geffner 2001). This is very valuable information that has been used to improve a wide variety of planning algorithms, such as pruning of the unreachable or dead-end operators (Alcázar and Torralba 2015; Fiser and Komenda 2018; Fišer, Torralba, and Shleyfman 2019), pruning states in regression search (Alcázar et al. 2013; Torralba and Alcázar 2013; Torralba et al. 2017), or translation of STRIPS representation into SAS$^+$ (Helmert 2009; Fiser and Komenda 2018; Fišer 2020; Fiser 2023), and strengthening heuristics (Haslum, Bonet, and Geffner 2005; Torralba, Linares López, and Borrajo 2018; Fišer, Horčík, and Komenda 2020).

We revisit this idea in the context of Cartesian abstractions. Besides removing abstract states and transitions that violate state invariants, we analyse how to use disambiguation to restrict the Cartesian sets corresponding to states, while maintaining a representation based on Cartesian sets. Furthermore, instead of only focusing on identifying unreachable states, we use dead-pair constraints that identify pairs of facts that cannot be part of any state that is part of a plan, i.e., any state that is unreachable from the initial state or that is a dead-end (Alcázar and Torralba 2015).

We show that using disambiguation during the CEGAR construction process can lead to more informed heuristics. However, checking all transitions can become expensive, and this overhead could result in less refined abstractions within the same time constraints. Therefore, we consider different variants, such as disambiguating only a subset of transitions or disambiguating operators preconditions instead. We analyse their relative power both theoretically and experimentally. Our experiments show that all of these variants significantly improve the performance, achieving a good tradeoff between informativeness of the heuristic and the computational overhead while building the abstraction.

Background

We consider tasks in SAS$^+$ representation (Bäckström and Nebel 1995), where states are described in terms of a set of variables V, and each $v \in V$ has a finite domain, $\mathcal{D}_v$. A *partial state* p is a variable assignment over some variables $\mathrm{vars}(p) \subseteq V$. A (concrete) state s is a full assignment, i.e., $\mathrm{vars}(s) = V$. We write $p[v]$ for the value assigned to $v \in \mathrm{vars}(p)$ in p. Two partial states p_1 and p_2 are consistent if $p_1[v] = p_2[v]$ for all $v \in \mathrm{vars}(p_1) \cap \mathrm{vars}(p_2)$. We denote by $[\![p]\!] \subseteq S$ the set of states consistent with p. A fact $f = \{v \mapsto x\}$ is an assignment of a value $x \in \mathcal{D}_v$ to a variable $v \in V$, so (partial) states can be interpreted as sets of facts.

A SAS$^+$ task Π is a tuple $\langle V, O, I, G \rangle$ where I is the initial state, G is a partial state describing the goal, and O is a set of operators $o = \langle pre(o), \mathit{eff}(o), \mathsf{cost}(o) \rangle$ where preconditions $pre(o)$ and effects $\mathit{eff}(o)$ are partial states and $\mathsf{cost}(o) \in \mathbb{R}_0^+$. An operator o is applicable in s if $s \in [\![pre(o)]\!]$. The result of applying o in s is another state $s' = s[\![o]\!]$ where $s'[v] = \mathit{eff}(o)[v]$ if $v \in \mathrm{vars}(\mathit{eff}(o))$ and $s'[v] = s[v]$ otherwise. We write $s \xrightarrow{o} s'$ as a shorthand for $s' = s[\![o]\!]$.

A task $\Pi = \langle V, O, I, G \rangle$ induces the *transition system* $\Theta = \langle S, O, T, I, \mathcal{G} \rangle$, where S is the set of states, $\mathcal{G} = [\![G]\!]$ is

the set of goal states, and $T = \{(s, o, s') \mid s \xrightarrow{o} s'\}$ is the set of transitions. A *path* from s to s' is a sequence of operators $\langle o_1, o_2, \ldots, o_n \rangle$ s.t. $s \xrightarrow{o_1} s_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} s'$. A *plan* π for s is a path from s to a goal state $s' \in \mathcal{G}$. The cost of π is the summed up cost of its operators. A plan for Π is a plan for I.

A state s is *forward-reachable* if there is a path from I to s. The set of all forward-reachable states is denoted by $S_{\rightarrow}$. A state s is *backward-reachable* (solvable) if there is a plan for s, and the set of backward-reachable states is denoted by $S_{\leftarrow}$. A state is *alive* if it is both forward- and backward-reachable, and the set of alive states is $S_{\leftrightarrow}$. A state is *dead* if it is not alive (so it is unreachable or a dead-end), and the set of dead states is denoted by $S_{\not\leftrightarrow}$.(Sievers and Helmert 2021).

A* search with an admissible heuristic is a common approach to find optimal plans (Hart, Nilsson, and Raphael 1968). A *heuristic* is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$. The goal-distance $h^*(s)$ from s is the minimum cost of any plan for s, or ∞ if no plan exists. h is admissible if $h(s) \leq h^*(s)$ for all states $s \in S$, consistent if $h(s) \leq h(s') + \mathsf{cost}(o)$ for all transitions $s \xrightarrow{o} s'$, and goal-aware if $h(s) = 0$ for all states $s \in \mathcal{G}$. Consistent and goal-aware heuristics are admissible. h is *forward-admissible* if $h(s) \leq h^*(s)$ for all states $s \in S_{\rightarrow}$, *forward-consistent* if $h(s) \leq h(s') + \mathsf{cost}(o)$ for all transitions $s \xrightarrow{o} s'$ s.t. $s, s' \in S_{\rightarrow}$, and *forward-goal-aware* if $h(s) = 0$ for all states $s \in \mathcal{G} \cap S_{\rightarrow}$. Forward-consistent and forward-goal-aware heuristics are forward-admissible. As only forward-reachable states are generated during the search, A* with a forward-admissible heuristic is optimal.

A *mutex pair*, or simply *mutex*, is a pair of facts that are not part of any forward-reachable state, i.e., $\langle f_1, f_2 \rangle \not\subseteq s$ for all $s \in S_{\rightarrow}$. A known method to collect mutex pairs is the h^2 heuristic (Haslum, Bonet, and Geffner 2005). The same criterion can be used in regression to find pairs of facts that cannot simultaneously be present in alive states (Alcázar and Torralba 2015). We call *dead pairs* or *dead pair constraints* to a set of fact pairs that can only occur on dead states. The set of states that do not satisfy dead pair constraints is $[\![\mathcal{C}_{dead}]\!] = \{s \mid \exists f_1 \in s, f_2 \in s \text{ s.t. } \langle f_1, f_2 \rangle \in \mathcal{C}_{dead}\}$. Note that $[\![\mathcal{C}_{dead}]\!] \subseteq S_{\not\leftrightarrow}$, but they are not necessarily equal since dead pairs do not encapsulate all dead constraints. A *disambiguation* of a partial state p for a variable $v \in V$ and a set of dead pairs $\mathcal{C}_{dead}$ is a set of values $d_v \subseteq \mathcal{D}_v$ s.t. $s[v] \in d_v$ for all $s \in S_{\leftrightarrow} \cap [\![p]\!]$ (Fišer, Horčík, and Komenda 2020).

An abstraction $\alpha = \langle f^\alpha, \Theta^\alpha \rangle$ consists of a surjective function $f^\alpha : S \mapsto S^\alpha$ and an abstract transition system $\Theta^\alpha = \langle S^\alpha, O, T^\alpha, I^\alpha, \mathcal{G}^\alpha \rangle$ where S^α is a finite set of abstract states and $[\![s^\alpha]\!] = \{s \mid s \in S, f^\alpha(s) = s^\alpha\}$ for all $s^\alpha \in S^\alpha$. Θ^α is a homomorphism of Θ, i.e., $I^\alpha = f^\alpha(I)$, $\mathcal{G}^\alpha = \{f^\alpha(s) \mid s \in \mathcal{G}\}$ and there is a transition $f^\alpha(s) \xrightarrow{o} f^\alpha(t)$ for all $s \xrightarrow{o} t \in T$. A *conservative* abstraction (homomorphism) may contain additional arbitrary transitions, while in an *induced* abstraction (*strict* homomorphism) each abstract transition corresponds to a concrete transition: $T^\alpha = \{(f^\alpha(s) \xrightarrow{o} f^\alpha(t) \mid s \xrightarrow{o} t \in T)\}$. Each abstraction induces a heuristic h^α where $h^\alpha(s)$ is the distance from $f^\alpha(s)$ to the closest goal abstract state in Θ^α.

In Cartesian abstractions the set of states $[\![s^\alpha]\!]$ is Cartesian for all $s^\alpha \in S^\alpha$, i.e. it is of the form $A_1 \times A_2 \times \ldots \times A_n$, where $A_i \subseteq \mathcal{D}_{v_i}$ for all $v_i \in V$ (Seipp and Helmert 2018).

A Cartesian state a is a tuple $\langle A_1, A_2, \ldots, A_n \rangle$, where $A_i \subseteq \mathcal{D}_{v_i}$ for all $v_i \in V$, representing the Cartesian set of states $[\![a]\!] = A_1 \times A_2 \times \ldots \times A_n$. Thus, a Cartesian state compactly represents a set of states with space linear in the number of variables and their domains. Given a Cartesian state a, we denote by $a[v_i]$ the set of values that v_i can take in a, i.e., $a[v_i] = A_i \subseteq \mathcal{D}_{v_i}$. The intersection of two Cartesian states $a' = a_1 \cap a_2$ is also a Cartesian state, where $a'[v] = a_1[v] \cap a_2[v]$ for all $v \in V$. Also, for any (partial) state p, we can build a Cartesian state $a = \mathcal{X}(p)$ such that $[\![a]\!] = [\![p]\!]$, by making $a[v] = \{p[v]\}$ if $v \in \mathrm{vars}(p)$ and $a[v] = \mathcal{D}_v$ otherwise. An operator o is applicable in a Cartesian state a if $pre(o) \cap a \neq \varnothing$, and $a[\![o]\!][v] = \{post(o)[v]\}$ if $v \in \mathrm{vars}(post(o))$ and $a[\![o]\!][v] = a[v]$ otherwise.

The only known technique to obtain informative Cartesian abstractions is CEGAR (Seipp and Helmert 2013, 2018). It starts with the trivial abstraction, which consists of a single abstract state a s.t. $a[v] = \mathcal{D}_v$ for all $v \in V$. Then, it is iteratively refined until reaching a termination condition or finding a feasible plan. In the refinement loop, an optimal abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} a_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} a_n$ is executed in the concrete space, resulting in a trace $s_0 \xrightarrow{o_1} s_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} s_n$. If this execution succeeds and $s_n \in \mathcal{G}$, then it is an optimal plan for the task. If the plan execution fails at some step, a flaw is reported and the abstraction is refined by splitting an abstract state along the plan into two, in such a way that the same flaw cannot happen again.

Constrained Abstractions

Abstractions induce admissible heuristics for all states, but they can be constrained to guarantee the admissibility only for states that satisfy a set of constraints, $\mathcal{C}$, which correspond to a set of states (e.g., dead states $[\![\mathcal{C}_{dead}]\!]$) that can be ignored by the abstraction. Thus, abstractions must preserve the behaviour over the set of permitted states $P_{\mathcal{C}} = S \setminus [\![\mathcal{C}]\!]$.

Definition 1 (Constrained Abstraction)**.** *Let $\Theta = \langle S, O, T, I, \mathcal{G} \rangle$ be a transition system, $\mathcal{C}$ a set of state constraints, and D any set of states s.t. $P_{\mathcal{C}} \subseteq D \subseteq S$ and $I \in D$. A constrained abstraction for Θ wrt. $\mathcal{C}$ with domain D is a pair $\alpha_{\mathcal{C}} = \langle f^{\alpha_{\mathcal{C}}}, \Theta^{\alpha_{\mathcal{C}}} \rangle$ where $f^{\alpha_{\mathcal{C}}} : D \mapsto S^{\alpha_{\mathcal{C}}}$ is a surjective function and $\Theta^{\alpha_{\mathcal{C}}} = \langle S^{\alpha_{\mathcal{C}}}, O^{\alpha_{\mathcal{C}}}, T^{\alpha_{\mathcal{C}}}, I^{\alpha_{\mathcal{C}}}, \mathcal{G}^{\alpha_{\mathcal{C}}} \rangle$ is a transition system s.t. $O^{\alpha_{\mathcal{C}}} \subseteq O$, $I^{\alpha_{\mathcal{C}}} = f^{\alpha_{\mathcal{C}}}(I)$, $T^{\alpha_{\mathcal{C}}}_{P_{\mathcal{C}}} \subseteq T^{\alpha_{\mathcal{C}}} \subseteq T^{\alpha_{\mathcal{C}}}_{D}$, $\mathcal{G}^{\alpha_{\mathcal{C}}}_{P_{\mathcal{C}}} \subseteq \mathcal{G}^{\alpha_{\mathcal{C}}} \subseteq \mathcal{G}^{\alpha_{\mathcal{C}}}_{D}$, $T^{\alpha_{\mathcal{C}}}_{X} = \{f^{\alpha_{\mathcal{C}}}(s_i) \xrightarrow{o} f^{\alpha_{\mathcal{C}}}(s_j) \mid s_i, s_j \in X$ s.t. $s_i \xrightarrow{o} s_j\}$ and $\mathcal{G}^{\alpha_{\mathcal{C}}}_{X} = \{f^{\alpha_{\mathcal{C}}}(s) \mid s \in \mathcal{G} \cap X\}$ for $X \in \{P_{\mathcal{C}}, D\}$.*

A constrained abstraction $\alpha_{\mathcal{C}} = \langle f^{\alpha_{\mathcal{C}}}, \Theta^{\alpha_{\mathcal{C}}} \rangle$ induces a heuristic function $h^{\alpha_{\mathcal{C}}}$ where $h^{\alpha_{\mathcal{C}}}(s)$ is the distance from $f^{\alpha_{\mathcal{C}}}(s)$ to the closest goal abstract state if $s \in D$ and ∞ otherwise. It is induced if $T^{\alpha_{\mathcal{C}}} = T^{\alpha_{\mathcal{C}}}_{P_{\mathcal{C}}}$ and $\mathcal{G}^{\alpha_{\mathcal{C}}} = \mathcal{G}^{\alpha_{\mathcal{C}}}_{P_{\mathcal{C}}}$. Often, abstractions are assumed to be induced as this leads to better heuristics. But computing an induced constrained abstraction may be hard in general, especially if we do not have any assumption on the set of constraints $\mathcal{C}$. Thus, we approximate it by a superset of states $D \subseteq P_{\mathcal{C}}$, but still allow for better approximations when considering the set of transitions and goal states.

This definition is similar to the one by Haslum, Bonet, and Geffner (2005), but more general in three aspects. First, they approximate the set of abstract states as the ones in which non-abstracted variables do not contain a mutex set. That is a specific over-approximation of the permitted states. Stronger approximations could be used, even in the case of Pattern Databases where variables are either fully ignored, or fully considered. For example, there could be a variable that has been abstracted away, but whose values are all incompatible with different values of the non-abstracted variables. Thus, we leave the definition of constrained abstraction more generic to allow for multiple over-approximations of the set of permitted states. Second, we also restrict the set of goal states in the constrained abstraction because, even if an abstract state is not removed, it may be incompatible with the goal. Finally, our definition allows any type of constraints (not only mutexes that cannot appear in reachable states), and we consider dead pairs throughout the paper (that discard both unreachable and dead-end states).

Theorem 1. *Let Θ be a transition system and $\alpha_{\mathcal{C}} = \langle f^{\alpha_{\mathcal{C}}}, \Theta^{\alpha_{\mathcal{C}}} \rangle$ a constrained abstraction for Θ wrt. $\mathcal{C}$. If $S_{\rightarrow} \subseteq P_{\mathcal{C}}$, $h^{\alpha_{\mathcal{C}}}$ is forward-consistent, forward-goal-aware, and so forward-admissible.*

Proof. h is forward-consistent if $h(s) \leq h(s') + \mathsf{cost}(o)$ for all $s \xrightarrow{o} s'$ s.t. $s, s' \in S_{\rightarrow}$. A transition $f^{\alpha_{\mathcal{C}}}(s) \xrightarrow{o} f^{\alpha_{\mathcal{C}}}(s')$ exists in $\Theta^{\alpha_{\mathcal{C}}}$ if $s \xrightarrow{o} s'$ for all $s, s' \in P_{\mathcal{C}}$. Thus, the maximum cost between s and s' is $\mathsf{cost}(o)$ because such an abstract transition exists, and maybe lower if non-induced transitions between s and s' with lower cost exist.

h is forward-goal-aware if $h(s) = 0$ for all $s \in \mathcal{G} \cap S_{\rightarrow}$. As $\mathcal{G}^{\alpha_{\mathcal{C}}} = \{f^{\alpha_{\mathcal{C}}}(s) | s \in \mathcal{G} \cap X\}$, where $P_{\mathcal{C}} \subseteq X$, and $S_{\rightarrow} \subseteq P_{\mathcal{C}}$, $h^{\alpha_{\mathcal{C}}}$ is forward-goal-aware. Hence, h is forward-admissible. □

Theorem 2. *Let Θ be a transition system, $\alpha = \langle f^{\alpha}, \Theta^{\alpha} \rangle$ an abstraction for Θ and $\alpha_{\mathcal{C}} = \langle f^{\alpha_{\mathcal{C}}}, \Theta^{\alpha_{\mathcal{C}}} \rangle$ a constrained abstraction for Θ wrt. $\mathcal{C}$ with domain D, s.t. $f^{\alpha_{\mathcal{C}}}(s) = f^{\alpha}(s)$ for all $s \in D$. Then, $h^{\alpha}(s) \leq h^{\alpha_{\mathcal{C}}}(s)$ for all states $s \in S$.*

Proof. $h^{\alpha_{\mathcal{C}}} = \infty \geq h^{\alpha}$ for all $s \notin D$. Also, $h^{\alpha}(s) \leq h^{\alpha_{\mathcal{C}}}(s)$ for all $s \in D$ because $T^{\alpha_{\mathcal{C}}} \subseteq T^{\alpha}$ and $\mathcal{G}^{\alpha_{\mathcal{C}}} \subseteq \mathcal{G}^{\alpha}$, and removing transitions and/or goal states can only maintain or increase the optimal-goal distance. □

Constrained Cartesian Abstractions

Given a Cartesian state a and a set of binary constraints $\mathcal{C}$ such that the set of permitted states is $P_{\mathcal{C}} = S \setminus [\![\mathcal{C}]\!]$, determining if $P_{\mathcal{C}} \cap a = \varnothing$ is NP-Complete. This corresponds to a Constrained Satisfaction Problem with finite-domain variables (with domain $|a[v]|$ for all $v \in V$) and a set of binary constraints ($\mathcal{C}$). It is decidable in polynomial time for binary variables or acyclic constraints, but this is not the case in general (Schaefer 1978; Jonsson, Lagerkvist, and Osipov 2021). However, incomplete methods with a lower complexity can be used as a tradeoff.

Definition 2 (Disambiguation Method)**.** *A disambiguation method $\lambda^{\mathcal{C}_{dead}}$ is a function that receives a Cartesian state c and a set of dead pairs $\mathcal{C}_{dead}$ and returns another Cartesian state c' such that $c \setminus [\![\mathcal{C}_{dead}]\!] \subseteq c' \subseteq c$.*

Algorithm 1: AC-3 Disambiguation

Data: Cartesian state c, $\mathcal{C}_{dead}$
Result: Disambiguated Cartesian state c'

$c' \leftarrow c$
$work \leftarrow \{\langle x, y \rangle \mid \exists x_i \in \mathcal{D}_x, y_j \in \mathcal{D}_y : \langle x_i, y_j \rangle \in \mathcal{C}_{dead}\}$
while not $work.\mathit{empty}()$ **do**
  $\langle x, y \rangle \leftarrow work.\mathit{pop}()$
  foreach $x_i \in c'[x]$ **do**
    if $\langle x_i, y_j \rangle \in \mathcal{C}_{dead}$ *for all* $y_j \in c'[y]$
      $c'[x] \leftarrow c'[x] \setminus x_i$
  if c' *has changed*
    if $c'[x] = \varnothing$
      return $\varnothing$
    $work.\mathit{push}(\{\langle z, x \rangle \mid \exists \langle z_i, x_j \rangle \in \mathcal{C}_{dead} :$
     $z_i \in \mathcal{D}_z \wedge x_j \in \mathcal{D}_x \wedge z \neq y \wedge \langle z, x \rangle \notin work\})$
return c'

The disambiguation method is an essential component of this work, as we need a fast but strong method to disambiguate as many facts as possible. We use a variant for Cartesian states of the constraints satisfaction AC-3 algorithm (Mackworth 1977). Its time complexity is $O(ed^3)$, where e is the number of dead pairs and d is the size of the largest variable's domain. Algorithm 1 introduces in a work set all the pairs of variables with any constraint between them and loops until it is empty. In this loop, a pair of variables $\langle x, y \rangle$ is extracted from the work set, and then it removes from the Cartesian state the values of x that do not satisfy the dead pair constraints for all the values of y. If any value is removed, all pairs of variables $\langle z \neq y, x \rangle$ with some constraint with x are added to the work set. A similar algorithm was used by Fišer, Horčík, and Komenda (2020), but using a work set reduces the number of checks in most cases.

We consider multiple ways to constrain a Cartesian abstraction for a set of dead pairs, called *constrainers*.

Definition 3 (Constrainer)**.** *A constrainer Γ is a function that receives a (constrained) Cartesian abstraction $\alpha_{\mathcal{C}} = \langle f^{\alpha_{\mathcal{C}}}, \Theta^{\alpha}_{\mathcal{C}} \rangle$,a set of dead pairs $\mathcal{C}_{dead}$, and a disambiguation method $\lambda^{\mathcal{C}_{dead}}$ and returns a constrained Cartesian abstraction $\alpha'_{\mathcal{C}} = \langle f^{\alpha'_{\mathcal{C}}}, \Theta^{\alpha'_{\mathcal{C}}} \rangle$ with domain $D \supseteq P_{\mathcal{C}}$ such that $f^{\alpha'_{\mathcal{C}}}(s) = f^{\alpha_{\mathcal{C}}}(s)$ for all $s \in D$.*

Our constrainers remove dead transitions and apply disambiguation on abstract states and operators. A Cartesian state a can be disambiguated by applying $\lambda^{\mathcal{C}_{dead}}(a)$, so $a[v]$ may be reduced for some $v \in V$. In that case, we remove the corresponding states from the domain of $f^{\alpha'_{\mathcal{C}}}$, all transitions from or into a that are no longer induced, and a is removed from $\mathcal{G}^{\alpha'_{\mathcal{C}}}$ if no goal state is mapped to it any more.

We also remove as many dead transitions as possible. A transition $a_i \xrightarrow{o} a_j$ is dead for constraints $\mathcal{C}$ if $\nexists s_i, s_j \in P_{\mathcal{C}}$ s.t. $s_i \in [\![a_i]\!]$, $s_j \in [\![a_j]\!]$, and $s_i \xrightarrow{o} s_j$. Algorithm 2 detects dead transitions by applying $\lambda^{\mathcal{C}_{dead}}$ on the intersection of a_i and $pre(o)$ to check if o is applicable in some state $s \in P_{\mathcal{C}}$. Then, it applies $\lambda^{\mathcal{C}_{dead}}$ on the intersection of $c[\![o]\!]$ and a_j to check if o reaches some state $s_j \in P_{\mathcal{C}}$ from some state $s_i \in [\![a_i]\!] \cap P_{\mathcal{C}}$.

Algorithm 2: isDead

Data: Transition $a_i \xrightarrow{o} a_j$, $\lambda^{\mathcal{C}_{dead}}$
Result: $\top$ if the transition is dead and $\bot$ otherwise
1 $aliveSrc \leftarrow \lambda^{\mathcal{C}_{dead}}(\mathcal{X}(pre(o)) \cap a_i)$
2 **if** $aliveSrc = \varnothing$
3 | **return** $\top$
4 $succ \leftarrow aliveSrc[\![o]\!]$
5 **return** $\lambda^{\mathcal{C}_{dead}}(succ \cap a_j) = \varnothing$

Algorithm 3: disambiguateOp

Data: Operator o, $\lambda^{\mathcal{C}_{dead}}$, $\mathcal{C}_{dead}$
Result: Disambiguated operator
1 $cpre \leftarrow \mathcal{X}(pre(o))$
2 **foreach** $v_p \mapsto x_p \in cpre[v_p]$ *s.t.* $v_p \notin \text{vars}(\mathit{eff}(o))$ **do**
3 | **if** $\exists v_e \mapsto x_e \in \mathit{eff}(o)$ *s.t.* $\langle v_p \mapsto x_p, v_e \mapsto x_e\rangle \in \mathcal{C}_{dead}$
4 | | $cpre[v_p] \leftarrow cpre[v_p] \setminus \{x_p\}$
5 $cpre \leftarrow \lambda^{\mathcal{C}_{dead}}(cpre)$
6 **return** *Operator*$(cpre, \mathit{eff}(o), \mathsf{cost}(o))$

Proposition 1. *Algorithm 2 is sound (it returns $\top$ only if a transition does not exist between a pair of permitted states).*

Proof. Let $a_i \xrightarrow{o} a_j$ be a non-dead transition for which $\lambda^{\mathcal{C}_{dead}}(succ \cap a_j) = \varnothing$. Then, alive states $s_i \in [\![a_i]\!]$, $s_j \in [\![a_j]\!]$ such that $s_i \xrightarrow{o} s_j$ exist. *aliveSrc* is empty iff no concrete state s_i s.t. $s_i \in [\![a_i]\!]$ from which o is applicable does not exist, and so $s_i \notin P_{\mathcal{C}}$, $s_i \notin [\![a_i]\!]$ or $s_i \notin [\![pre(o)]\!]$, a contradiction. *aliveSrc* is not empty but $\lambda^{\mathcal{C}_{dead}}(succ \cap a_j) = \varnothing$, then o is applicable in s_i but $s_j \notin P_{\mathcal{C}}$ or $s_j \notin [\![a_j]\!]$, a contradiction. □

Disambiguating operators reduces the set of dead transitions induced by it. This is weaker than detecting dead transitions with the aforementioned method, but faster since operators are disambiguated only once rather than checking each transition individually. The idea is to represent preconditions as a Cartesian state that indicates the subset of facts for each variable in which it is applicable. Algorithm 3 restricts the domain of variables by removing the facts not consistent with the preconditions (and so with any state $s_i \in P_{\mathcal{C}}$ in which the operator is applicable) and reaches another state $s_j \in P_{\mathcal{C}}$ (so if the operator does not have an effect on a variable, all facts mutex with some effect can also be removed).

Proposition 2. *Algorithm 3 is sound ($s \in [\![cpre]\!]$ for all $s, s' \in P_{\mathcal{C}}$ s.t. $s \xrightarrow{o} s'$).*

Proof. Let $s \notin [\![cpre]\!]$ be a state s.t. $s, s' \in P_{\mathcal{C}}$ and $s \xrightarrow{o} s'$. As o is applicable in s, $s \in [\![pre(o)]\!]$, so s is removed from $[\![cpre]\!]$ in line 4 or 5.

Case 1 (line 4). $\{v_p \mapsto x_p\} \in s$. Then, $\{v_p \mapsto x_p\} \in s'$ because $v_p \notin \text{vars}(\mathit{eff}(o))$. As s' is the result of applying o, $\{v_e \mapsto x_e\} \in s'$. Therefore, $s' \in [\![\mathcal{C}_{dead}]\!]$, contradicting $s' \in P_{\mathcal{C}}$.

Case 2 (line 5). $s \notin P_{\mathcal{C}}$ (a contradiction) because by Definition 2 $\lambda^{\mathcal{C}_{dead}}(c)$ returns a Cartesian state c' s.t. $c \setminus [\![\mathcal{C}_{dead}]\!] \subseteq c' \subseteq c$. □

Proposition 3. *Algorithm 2 returns $\top$ for some transitions on an operator o disambiguated by Algorithm 3.*

Proof. Let Π be a task with $V = \{v_1, v_2, v_3\}$, $O = \{o = \langle\{v_1 \mapsto 0\}, \{v_1 \mapsto 1\}, 1\rangle\}$, and $\mathcal{C}_{dead} = \{\langle v_2 \mapsto 0, v_3 \mapsto 1\rangle\}$. Algorithm 3 does not alter o and given a Cartesian state $a = \langle\{0,1\}, \{0\}, \{1\}\rangle$, it is applicable in no $s \in [\![a]\!] \cap P_{\mathcal{C}}$ as $[\![pre(o) \cap a]\!] = \{v_1 \mapsto 0, v_2 \mapsto 0, v_3 \mapsto 1\} \in [\![\mathcal{C}_{dead}]\!]$. □

We define the following constrainers:

All Transitions Disambiguation ($\Gamma_\forall$) removes all non-looping transitions detected as dead, so that $T^{\alpha_\mathcal{C}} \subseteq T^\alpha$.

Optimal Transitions Disambiguation (Γ_*) is like $\Gamma_\forall$ but checking only dead optimal transitions, i.e., $a_i \xrightarrow{o} a_j$ s.t. $h^*_\alpha(a_i) = h^*_\alpha(a_j) + \mathsf{cost}(o)$. If any transition is removed in all optimal plans, the optimal cost changes and new optimal transitions detected as dead are removed until some optimal plan is preserved. Therefore, $T^{\alpha_\mathcal{C}} \subseteq T^\alpha$.

Operators Disambiguation (Γ_o) disambiguates all operators by Algorithm 3 and then all non-looping transitions that are not induced by the new definition of operators are removed. This constrainer removes an operator o if $pre(o)$ is empty, thus $O^{\alpha_\mathcal{C}} \subseteq O^\alpha$ and $T^{\alpha_\mathcal{C}} \subseteq T^\alpha$.

Abstract States Disambiguation (Γ_s) applies $\lambda^{\mathcal{C}_{dead}}$ to all abstract states and then it removes all non-induced non-looping transitions. It reduces the domain of $\alpha_\mathcal{C}$ and also abstract goal states may become non-goal if $\lambda^{\mathcal{C}_{dead}}(a \cap G) = \varnothing$, so that $S^{\alpha_\mathcal{C}} \subseteq S^\alpha$, $\mathcal{G}^{\alpha_\mathcal{C}} \subseteq \mathcal{G}^\alpha$, and $T^{\alpha_\mathcal{C}} \subseteq T^\alpha$.

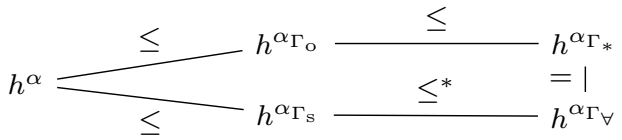
Only non-looping transitions are removed because loops have no effect in $h^{\alpha_\mathcal{C}}$. Constrainers are composable, so they can be applied in any order to produce another constrained abstraction $\alpha_{\mathcal{C}'}$ where $h^{\alpha_{\mathcal{C}'}}(s) \geq h^{\alpha_\mathcal{C}}(s)$ for all $s \in S$. We use the notation $\Gamma_{\mathrm{c_1 c_2 \ldots c_n}}$ for the combination of constrainers $c_1, c_2, \ldots, c_n$ and α_{c_x} for the abstraction resulted of applying Γ_{c_x}. Given a constrained abstraction $\alpha_\mathcal{C}$, $\mathcal{C}_{dead}$, and a $\lambda^{\mathcal{C}_{dead}}$, the following theorems state the relationships between the constrainers and the resulting abstraction $\alpha'_\mathcal{C}$.

Theorem 3. *$\alpha'_\mathcal{C} = \Gamma(\alpha_\mathcal{C}, \mathcal{C}_{dead}, \lambda^{\mathcal{C}_{dead}})$ is a constrained Cartesian abstraction for $\Gamma \in \{\Gamma_\forall, \Gamma_*, \Gamma_\mathrm{o}, \Gamma_\mathrm{s}\}$, and any combination of them.*

Proof Sketch. Algorithm 2 is sound, thus $\Gamma_\forall$ and Γ_* only remove dead transitions resulting in a constrained abstraction. Algorithm 3 is sound, and so Γ_o only removes transitions not induced by permitted states, resulting in a constrained abstraction. Γ_s applies $\lambda^{\mathcal{C}_{dead}}$ to states, so that only transitions not induced by two permitted states are removed. Finally, any combination of them applies their transformations without side effects, resulting in a constrained abstraction. □

Theorem 4. $\forall s \in S, h^\alpha(s) \leq h^{\alpha_{\Gamma_\mathrm{o}}}(s) \leq h^{\alpha_{\Gamma_\forall}}(s) = h^{\alpha_{\Gamma_*}}(s)$.

Proof Sketch. As $\alpha_{\Gamma_\mathrm{o}}$ is a constrained abstraction, $h^\alpha(s) \leq h^{\alpha_{\Gamma_\mathrm{o}}}(s)$ for all states $s \in S$. Algorithm 2 detects as dead some transitions on operators disambiguated by Algorithm 3, thus Γ_o removes up to the same transitions as $\Gamma_\forall$. Removing a non-optimal transition $a_i \xrightarrow{o} a_j$ in a transition system Θ^α does not increase $h^*_\alpha(s_i)$ because $\exists a_i \xrightarrow{o'} a'$ s.t. $\mathsf{cost}(o') < \mathsf{cost}(o)$, and so $h^{\alpha_{\Gamma_\forall}}(s) = h^{\alpha_{\Gamma_*}}(s)$ for all states $s \in S$. □



*Only guaranteed if $S(a) \subseteq \mathcal{G}$ for all $a \in \mathcal{G}^\alpha$

Figure 1: Relationships among all atomic constrainers.

Theorem 5. *There are cases where $h^{\alpha_{\Gamma_s}}(s) > h^{\alpha_{\Gamma_\forall}}(s)$ for some $s \in S$ but only if $\exists s \notin \mathcal{G}$ s.t. $f^\alpha(s) \in \mathcal{G}^\alpha$.*

Proof. Let $\mathcal{G}^{\alpha_{\Gamma_s}}$ be the goal states of α_{Γ_s} and $\mathcal{G}^{\alpha_{\Gamma_\forall}}$ the goal states of $\alpha_{\Gamma_\forall}$, such that $\mathcal{G}^{\alpha_{\Gamma_s}} \subseteq \mathcal{G}^{\alpha_{\Gamma_\forall}}$. Consider two abstractions $\alpha_{\Gamma_s} = \langle f^\alpha, \Theta^{\alpha_{\Gamma_s}} \rangle$, $\alpha_{\Gamma_\forall} = \langle f^\alpha, \Theta^{\alpha_{\Gamma_\forall}} \rangle$ such that $\Theta^{\alpha_{\Gamma_s}} = \Theta^{\alpha_{\Gamma_\forall}}$ except for $\mathcal{G}^{\alpha_{\Gamma_s}} \subset \mathcal{G}^{\alpha_{\Gamma_\forall}}$, and some $a = f^\alpha(s)$ whose only plan is $a \xrightarrow{o} b$ with $b \in \mathcal{G}^{\alpha_{\Gamma_\forall}}$ and $b \notin \mathcal{G}^{\alpha_{\Gamma_s}}$. Then $h^{\alpha_{\Gamma_s}}(s) = \infty > h^{\alpha_{\Gamma_\forall}}(s) = \mathsf{cost}(o)$.

On the other hand, let $T^{\alpha_{\Gamma_s}}$ be the transitions of α_{Γ_s} and $T^{\alpha_{\Gamma_\forall}}$ the transitions of $\alpha_{\Gamma_\forall}$. $T^{\alpha_{\Gamma_\forall}} \subseteq T^{\alpha_{\Gamma_s}}$ because all non-induced transitions after disambiguating states are dead. So, if $\mathcal{G}^{\alpha_{\Gamma_s}} = \mathcal{G}^{\alpha_{\Gamma_\forall}}$, $h^{\alpha_{\Gamma_s}}(s) \leq h^{\alpha_{\Gamma_\forall}}(s)$ □

About Γ_s and Γ_o, each transforms the abstraction in a non-comparable way, so that depending on the abstraction each of them may generate a stronger heuristic than the other. Figure 1 summarises the relationships among all constrainers.

CEGAR$_d$: Disambiguation in CEGAR

Instead of first generating a Cartesian abstraction and then transforming it with constrainers, we apply disambiguation during CEGAR. This avoids the generation of a larger abstraction before constraining it, and guides CEGAR to only focus on alive states. The resulting CEGAR$_d$ (Algorithm 4) applies transformations similar to constrainers, with an additional transformation that considers the transitions of the optimal abstract plan to be refined.

All Transitions Disambiguation ($\mathfrak{D}_\forall$) does not add any non-looping transition detected as dead to any child obtained after a refinement.

Optimal Transitions Disambiguation ($\mathfrak{D}_*$) is like $\mathfrak{D}_\forall$ but checking only optimal transitions and optimal backward transitions ($a_i \xrightarrow{o} a_j$ s.t. $g_\alpha(a_j) = g_\alpha(a_i) + \mathsf{cost}(o)$), the latter to reduce the number of regression flaws (Pozo, Torralba, and Linares López 2024b). At this point, the children states are not part of the abstraction yet, and so the optimality is checked for the parent state. For performance reasons, if some transition is removed in all optimal plans, new optimal transitions are not checked.

Plan Transitions Disambiguation ($\mathfrak{D}_\tau$) removes transitions detected as dead from the optimal abstract plan where flaws are searched at each iteration.

Operators Disambiguation ($\mathfrak{D}_o$) applies $\lambda^{\mathcal{C}_{dead}}$ to all operators via Algorithm 3 before the refinement loop, which reduces the number of induced transitions. It is done only once even for additive abstractions, so its overhead is low.

Algorithm 4: CEGAR$_d$ main algorithm

Data: Task $\Pi = \langle V, O, I, G \rangle$, $\lambda^{\mathcal{C}_{dead}}, \mathcal{C}_{dead}$
Result: "unsolvable", concrete plan or abstraction

$O' \leftarrow \forall o \in O$ *disambiguateOp*$(o, \lambda^{\mathcal{C}_{dead}}, \mathcal{C}_{dead})$; // $\mathfrak{D}_o$
$\Theta^\alpha = \langle S^\alpha, O', T^\alpha, I^\alpha, \mathcal{G}^\alpha \rangle \leftarrow$ *trivialAbstraction*()
$I^\alpha \leftarrow \lambda^{\mathcal{C}_{dead}}(I^\alpha)$
while not *terminationCondition*() **do**
  $\tau^\alpha \leftarrow$ *findOptimalTrace*(Θ^α)
  if τ^α = *"no trace"*
    return *"unsolvable"*
  if $\exists a_i \xrightarrow{o} a_{i+1} \in \tau^\alpha$ *s.t.* *isDead*$(a_i \xrightarrow{o} a_{i+1}, \lambda^{\mathcal{C}_{dead}})$
    remove $a_i \xrightarrow{o} a_{i+1}$ from T^α ; // $\mathfrak{D}_\tau$
    update abstract states distances
    continue
  $\varphi \leftarrow$ *findFlaw*(Π, τ^α)
  if φ = *"no flaw"*
    return *plan extracted from* τ^α
  $d, e, v_s \leftarrow$ *split*(φ) // $v_s = v \in V$ in which $d[v_s] \neq e[v_s]$
  $d \leftarrow \lambda^{\mathcal{C}_{dead}}(d, v_s)$, $e \leftarrow \lambda^{\mathcal{C}_{dead}}(e, v_s)$; // $\mathfrak{D}_s$
  $T^\alpha, I^\alpha, \mathcal{G}^\alpha \leftarrow$ *rewire*$(\Theta^\alpha, d, e, \lambda^{\mathcal{C}_{dead}})$; // $\mathfrak{D}_\forall, \mathfrak{D}_*$
return Θ^α

Abstract States Disambiguation ($\mathfrak{D}_s$) applies $\lambda^{\mathcal{C}_{dead}}$ to the trivial abstraction and each child obtained after refinements, which reduces the number of transitions and goal abstract states (if $\lambda^{\mathcal{C}_{dead}}(a \cap G) = \varnothing$). The cost of AC-3 is reduced by initialising the work set only with variables with dead pairs for the variable by which the state is split.

CEGAR$_d$ applies these transformations independently, so any number of them can be applied, although combining $\mathfrak{D}_\forall$, $\mathfrak{D}_*$ and $\mathfrak{D}_\tau$ is useless because the stricter transformation makes the others helpless. $\mathfrak{D}_{d_1 d_2 \ldots d_n}$ denote the combination of transformations $d_1, d_2, \ldots, d_n$. Algorithm 4 applies any of these transformations during the CEGAR process.

Proposition 4. *The CEGAR$_d$ algorithm induces a forward-admissible heuristic.*

Proof Sketch. Transformations applied by CEGAR$_d$ produce a constrained Cartesian abstraction with $S_\rightarrow \subseteq P_\mathcal{C}$. □

Experiments

We implemented CEGAR$_d$ within the Scorpion planner (Seipp, Keller, and Helmert 2020), based on Fast Downward (Helmert 2006). We run experiments using Downward Lab (Seipp et al. 2017) with limits to 30 minutes and 8 GiB of RAM. We use the 30 domains (out of 42) of Autoscale 21.11 benchmark (Torralba, Seipp, and Sievers 2021) for which the h^2-preprocessor finds dead pairs (Alcázar and Torralba 2015). Code and experimental data is available in Zenodo (Pozo, Torralba, and Linares López 2026). We use the best strategies from previous works: progression flaws, regression flaws (Pozo, Torralba, and Linares López 2024b) and sequence (it) flaws (Pozo, Torralba, and Linares López 2024a). Also, all experiments prune dead-end states detected by the constraints during search.

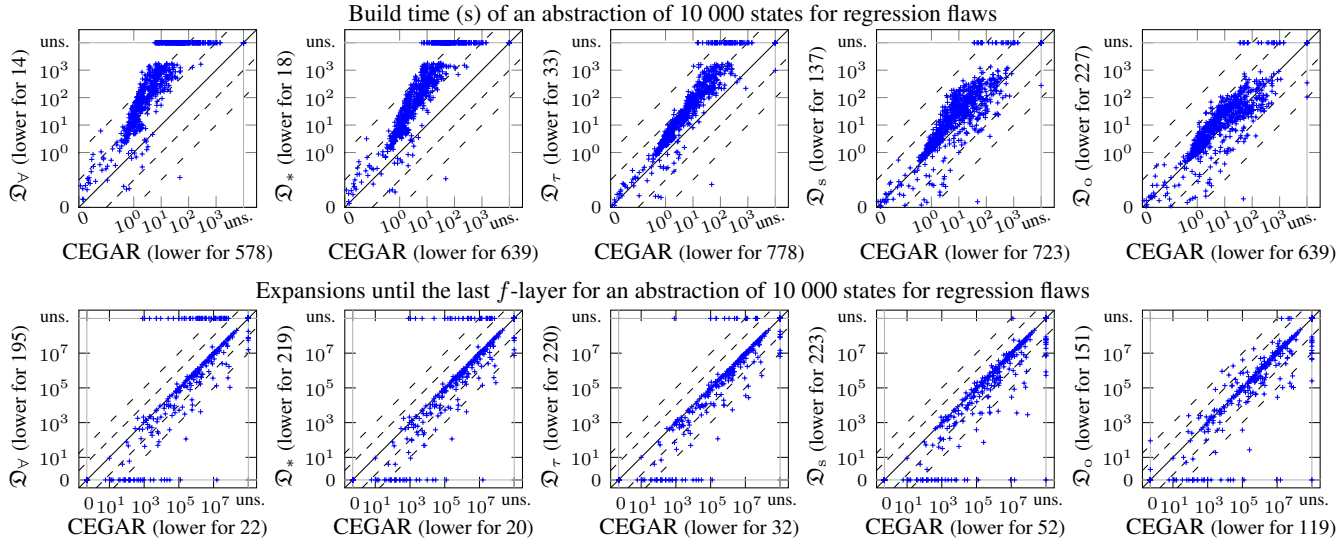


Figure 2: Build time and expansions of CEGAR against atomic transformations (numbers compare tasks solved by both).

Single Abstraction

First, we analyse the behaviour of disambiguation for a single abstraction with 10 000 states (except a concrete solution is found earlier) for a fair comparison of our disambiguation transformations in abstractions of the same size. Figure 2 compares the build time and expansions until the last f-layer of the original CEGAR algorithm against single disambiguation transformations for regression flaws, where more disambiguations are applied. Removing all optimal dead transitions is too expensive, while removing plan dead transitions needs more time than no disambiguating, but it is still reasonable. Disambiguating states reduces the time spent in the transitions rewiring, and disambiguation of operators is even faster because it is applied only once. Regarding expansions, removing only optimal dead transitions is almost as effective as removing all, while removing only plan dead transitions is a bit weaker but a good faster proxy. Disambiguating operators is the worst approach in terms of expansions, while disambiguating states is a good tradeoff.

Next, we analyse the number of tasks solved for a single abstraction with 10 million non-looping transitions and 15 minutes as bounds for each transformation. These bounds try to get a big enough abstraction without neither surpassing memory nor taking too much time, as the number of transitions varies greatly for different transformations and domains. Table 1 shows the number of domains in which the largest number of tasks is solved and the coverage during the refinement loop and during search for each transformation. The iterative sequence strategy solves more tasks than regression flaws and the latter solve more tasks than progression flaws for all transformations. Disambiguation is more useful for regression and sequence flaws, up to 29 more solved tasks for the disambiguation of states than original CEGAR. Disambiguating operators and plan transitions is also useful, but they usually do not pay off. The disambiguation of operators is much more useful for sequence flaws.

	Progr. flaws			Progr. it flaws			Regr. flaws		
Transf.	Dom	$\pi{=}\pi^\alpha$	Cov	Dom	$\pi{=}\pi^\alpha$	Cov	Dom	$\pi{=}\pi^\alpha$	Cov
CEGAR	7	138	332	15	135	367	13	112	355
$\mathfrak{D}_\forall$	7	149	329	13	155	358	13	142	363
$\mathfrak{D}_{\mathrm{so}\forall}$	7	156	331	12	160	363	12	141	355
$\mathfrak{D}_*$	8	153	330	13	158	365	13	151	363
$\mathfrak{D}_{\mathrm{so}*}$	8	167	334	12	169	369	15	151	362
$\mathfrak{D}_\tau$	8	148	333	16	162	373	13	143	366
$\mathfrak{D}_\mathrm{o}$	6	172	318	16	175	387	14	146	363
$\mathfrak{D}_{\mathrm{so}\tau}$	8	**190**	343	16	187	391	15	166	380
$\mathfrak{D}_\mathrm{so}$	7	**190**	339	16	187	**393**	17	167	379
$\mathfrak{D}_\mathrm{s}$	6	181	340	**18**	177	**393**	**18**	166	384

Table 1: Number of domains in which the largest number of tasks is solved (Dom), coverage during CEGAR ($\pi{=}\pi^\alpha$) and coverage during search (Cov) for the most interesting disambiguation transformations.

Table 2 shows the coverage of different strategies for CEGAR and the best CEGAR$_d$ transformations: $\mathfrak{D}_\mathrm{s}$ and $\mathfrak{D}_\mathrm{o}$. Disambiguation is more useful for regression and sequence flaws, where states during the search of flaws are Cartesian, and reducing their size also reduces the number of flaws. It is especially powerful in agricola (up to 9 more tasks solved) and tidybot (up to 11 more tasks solved), but useful in almost all domains. A novel strategy that refines the flaw that disambiguates the maximum number of children states (dis) is good in progression but less than the iterative strategy (it).

Multiple Abstractions

CEGAR is usually applied over additive abstractions, whose heuristics are combined by using saturated cost partitioning (Seipp and Helmert 2018). Forward-admissibility is not lost with additive constrained abstractions because the heuristic induced by each abstraction is also forward-admissible.

		Single abstraction											Multiple abstractions										
		Progression flaws							Regression flaws				Progression flaws							Regression flaws			
Domain	$\mathcal{C}_{dead}$ (k)	C	$\mathfrak{D}_\mathrm{s}$	$\mathfrak{D}_\mathrm{o}$	C^it	$\mathfrak{D}_\mathrm{s}^\mathrm{it}$	$\mathfrak{D}_\mathrm{o}^\mathrm{it}$	$\mathfrak{D}_\mathrm{s}^\mathrm{dis}$	C	$\mathfrak{D}_\mathrm{s}$	$\mathfrak{D}_\mathrm{o}$	$\mathfrak{D}_\mathrm{s}^\mathrm{dis}$	C	$\mathfrak{D}_\mathrm{s}$	$\mathfrak{D}_\mathrm{o}$	C^it	$\mathfrak{D}_\mathrm{s}^\mathrm{it}$	$\mathfrak{D}_\mathrm{o}^\mathrm{it}$	$\mathfrak{D}_\mathrm{s}^\mathrm{dis}$	C	$\mathfrak{D}_\mathrm{s}$	$\mathfrak{D}_\mathrm{o}$	$\mathfrak{D}_\mathrm{s}^\mathrm{dis}$
agricola	273.3	5	8	8	10	15	16	10	8	**17**	**17**	**17**	6	5	6	10	10	10	5	9	**11**	**11**	**11**
airport	36166.2	4	6	2	**7**	**7**	**7**	**7**	4	**7**	4	**7**	22	21	23	19	18	23	15	22	18	**25**	14
barman	74.6	12	12	12	13	14	**15**	13	12	14	14	13	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
blocksworld	5.2	12	13	13	14	15	15	15	13	15	15	**16**	15	13	**17**	15	13	16	15	**17**	15	16	13
depots	27.4	11	11	11	**13**	**13**	**13**	12	11	**13**	**13**	12	15	13	17	19	18	19	13	18	18	**21**	13
driverlog	3.1	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**20**	**20**	**20**	**20**	**20**	**20**	19	17	18	18	18
floortile	40.6	17	17	17	17	**18**	**18**	17	17	**18**	17	16	14	14	**16**	15	**16**	15	**16**	**16**	14	**16**	15
freecell	415.8	6	6	6	6	8	6	9	6	6	6	**12**	11	12	13	11	12	**17**	13	11	11	13	16
ged	49.0	22	22	22	**24**	**24**	**24**	22	**24**	**24**	**24**	22	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**
grid	19.8	15	16	15	22	**25**	24	11	19	20	16	13	19	18	19	17	18	**20**	14	19	17	19	15
mprime	5.7	6	6	6	**8**	**8**	**8**	6	6	6	6	7	17	16	17	22	**23**	22	17	22	21	21	**23**
nomystery	258.8	**7**	6	6	**7**	6	6	**7**	6	**7**	**7**	6	**16**	15	**16**	14	13	13	11	15	13	15	12
openstacks	46.9	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
organic-ss	8897.0	12	13	11	12	13	12	12	12	**14**	12	13	**14**	12	13	**14**	**14**	13	12	**14**	13	13	12
parcprinter	224.7	16	16	16	16	**17**	**17**	**17**	**17**	**17**	**17**	**17**	20	19	27	20	26	27	22	20	19	**30**	17
parking	366.8	8	8	8	9	10	10	9	9	**11**	10	10	**17**	16	15	**17**	16	15	**17**	**17**	16	**17**	16
pathways	11.7	9	9	9	9	10	**11**	9	10	10	**11**	9	11	11	10	11	11	11	11	12	**14**	11	**14**
pegsol	13.3	28	28	28	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
pipesw-not	232.9	18	19	18	20	20	20	18	19	**21**	18	20	23	22	23	24	**25**	23	24	23	23	22	23
pipesw-t	184.9	14	14	14	**15**	**15**	14	14	**15**	**15**	**15**	**15**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
rovers	1.5	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	4	4	4	4	4	4	4	4	4	4	**5**
snake	91.1	18	15	10	**20**	19	19	16	**20**	19	19	18	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
sokoban	78.7	17	16	12	17	17	17	17	**18**	15	12	**18**	21	20	**23**	20	20	21	21	20	20	**23**	21
storage	285.4	5	6	5	6	6	6	**7**	6	6	5	**7**	**15**	14	**15**	**15**	14	**15**	14	**15**	**15**	**15**	14
tetris	2759.0	19	18	18	**20**	**20**	19	18	**20**	**20**	**20**	**20**	**19**	18	**19**	**19**	18	**19**	18	18	18	**19**	18
thoughtful	2157.3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	6	5	**8**	5	5	6	5	6	5	7	5
tidybot	905.9	11	15	11	11	**22**	19	21	12	18	15	20	18	21	21	19	**23**	21	21	21	22	21	21
tpp	139.4	**3**	**3**	**3**	**3**	**3**	**3**	2	**3**	**3**	2	2	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
visitall	2.2	9	9	9	**11**	**11**	**11**	9	**11**	**11**	**11**	10	**14**	**14**	13	11	13	11	9	9	9	9	9
woodworking	73.2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	7	**8**	**8**
Total		332	340	318	367	**393**	387	355	355	384	363	377	450	436	468	454	463	474	429	458	446	**479**	438

Table 2: Coverage of CEGAR (C), $\mathfrak{D}_\mathrm{s}$ and $\mathfrak{D}_\mathrm{o}$ for some strategies. The largest number for single/multiple abstractions is highlighted. The $\mathcal{C}_{dead}$ column shows the total number of thousands of dead pairs for the 30 tasks of each domain. The superscript indicates the flaw selection strategy used: first flaw (no superscript), iterative flaws (it) (Pozo, Torralba, and Linares López 2024a), or refining the split that disambiguates the maximum number of children states (dis).

We combine landmark-based and goal-based subtasks by online Saturated Cost Partitioning with a bound of 900 seconds and 1 million of non-looping transitions as total for all subtasks, like previous works, since subtasks are smaller and need fewer transitions (Seipp and Helmert 2018). The right side of Table 2 shows the coverage of the top-performing transformations for this setting. Landmark-based subtasks usually abstract the domain of variables when individual values are irrelevant to achieve the landmark, but this produces weaker heuristics for CEGAR$_d$ because it reduces the number of dead pairs, so these results do not abstract them. Disambiguating states is too time-consuming for subtasks abstractions, as the time limit is small for each subtask, and so the best transformation for this setting is $\mathfrak{D}_\mathrm{o}$. Multiple abstractions get lower coverage in domains with tight interaction between landmarks as goals as agricola, but they also achieve impressive results in other domains like airport, depots, driverlog, storage, and parcprinter, even solving all tasks for the last one. Regression flaws achieve the best heuristics, solving 21 more tasks than the original CEGAR algorithm and 86 more tasks than the best single-abstraction setting. The novel dis strategy is especially worse in this setting because it increases the time spent on disambiguation and so the resulting abstractions are too small.

Conclusions

We have theoretically analysed constrained Cartesian abstractions, showing that different transformations can be applied to remove dead states and transitions. Applying these transformations during CEGAR reduces the overhead and guides the refinement process, resulting in more informed heuristics. Empirical results show that constrained Cartesian abstractions obtain a significant increment of tasks solved, and that disambiguating states is the most effective method for a single abstraction, whereas disambiguating operators is the best one for multiple abstractions.

Acknowledgments

This work was partially funded by grant PID2021-127647NB-C21 from MICIU/AEI/10.13039/501100011033, by the ERDF "A way of making Europe", and by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting Regression in Planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2254–2260. AAAI Press.

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

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

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

Edelkamp, S. 2001. Planning with Pattern Databases. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Fan, G.; and Holte, R. 2015. The Spurious Path Problem in Abstraction. In Lelis, L.; and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 18–27. AAAI Press.

Fišer, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9835–9842. AAAI Press.

Fiser, D. 2023. Operator Pruning Using Lifted Mutex Groups via Compilation on Lifted Level. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023*, 118–127. AAAI Press.

Fišer, D.; Horčík, R.; and Komenda, A. 2020. Strengthening Potential Heuristics with Mutexes and Disambiguations. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 124–133. AAAI Press.

Fiser, D.; and Komenda, A. 2018. Fact-Alternating Mutex Groups for Classical Planning. *Journal of Artificial Intelligence Research*, 61: 475–521.

Fišer, D.; Torralba, Á.; and Shleyfman, A. 2019. Operator Mutexes and Symmetries for Simplifying Planning Tasks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7586–7593. AAAI Press.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 1163–1168. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

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

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

Jonsson, P.; Lagerkvist, V.; and Osipov, G. 2021. Acyclic orders, partition schemes and CSPs: Unified hardness proofs and improved algorithms. *Artificial Intelligence*, 296: 103505.

Mackworth, A. K. 1977. Consistency in Networks of Relations. *Artif. Intell.*, 8(1): 99–118.

Pozo, M.; Torralba, Á.; and Linares López, C. 2024a. Gotta Catch 'Em All! Sequence Flaws in CEGAR for Classical Planning. In Endriss, U.; Melo, F. S.; Bach, K.; Diz, A. J. B.; Alonso-Moral, J. M.; Barro, S.; and Heintz, F., eds., *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI 2024)*, TODO: 20238–20246. IOS Press.

Pozo, M.; Torralba, Á.; and Linares López, C. 2024b. When CEGAR Meets Regression: A Love Story in Optimal Classical Planning. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20238–20246. AAAI Press.

Pozo, M.; Torralba, Á.; and Linares López, C. 2026. Supplementary Material, Code and Data for the AAAI 2026 Paper "Not Everything is Permitted: Constrained Cartesian Abstractions for Optimal Classical Planning". 10.5281/zenodo.17515416.

Schaefer, T. J. 1978. The Complexity of Satisfiability Problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC '78)*, 216–226. New York: ACM Press.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian Abstraction Refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

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

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

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

Sievers, S.; and Helmert, M. 2021. Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems. *Journal of Artificial Intelligence Research*, 71: 781–883.

Torralba, Á.; and Alcázar, V. 2013. Constrained Symbolic Search: On Mutexes, BDD Minimization and More. In Helmert, M.; and Röger, G., eds., *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SoCS 2013)*, 175–183. AAAI Press.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence*, 242: 52–79.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2018. Symbolic perimeter abstraction heuristics for cost-optimal planning. *Artificial Intelligence*, 259: 1–31.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 376–384. AAAI Press.