When CEGAR Meets Regression: A Love Story in Optimal Classical Planning

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

[1]Universidad Carlos III de Madrid, Madrid, Spain
[2]Aalborg University, Aalborg, Denmark
marcosmartin.pozo@alumnos.uc3m.es, alto@cs.aau.dk, carlos.linares@uc3m.es

Abstract

Counterexample-Guided Abstraction Refinement (CEGAR) is a prominent technique to generate Cartesian abstractions for guiding search in cost-optimal planning. The core idea is to iteratively refine the abstraction, finding a flaw of the current optimal abstract plan. All existing approaches find these flaws by executing the abstract plan using progression in the original state space.

Instead, we propose to do backward refinements by using regression from the goals. This results in a new type of flaw, that can identify invalid plan suffixes. The resulting abstractions are less focused on the initial state, but more informative on average, significantly improving the performance of current CEGAR-based techniques. Furthermore, they can be combined with forward refinements in several bidirectional strategies that provide the benefits of both methods.

Introduction

Counterexample-guided abstraction refinement (CEGAR) is a method that originated in model-checking (Clarke et al. 2000), where it is widely used to prove that error states are unreachable. The key idea is to iteratively refine an abstraction by finding an optimal abstract plan, understanding why it is not an actual plan by using interpolation to find a logic formula that explains the difference, and changing the abstraction to reflect the difference. The idea was brought to planning by Seipp and Helmert (2013b; 2018), using CEGAR to generate Cartesian abstractions that result in informative admissible heuristics for A* search (Hart, Nilsson, and Raphael 1968). Since then, CEGAR has become one of the dominant approaches for generating all kinds of abstraction heuristics (Rovner, Sievers, and Helmert 2019; Kreft et al. 2023). This is best exemplified by the state-of-the-art Scorpion planner (Seipp 2018), which uses this method to compute a diverse set of Cartesian abstraction heuristics and combine them additively (Seipp and Helmert 2013a, 2014) using saturated cost-partitioning (Katz and Domshlak 2010; Seipp, Keller, and Helmert 2017, 2020).

But the question of how to guide the refinement process of CEGAR to generate informative abstractions has comparatively received much less attention. One possibility is to perform batch refinement (Speck and Seipp 2022), where at each iteration the abstraction is refined based on multiple plans instead of only one. In this paper, we focus on the notion of *flaw*, i.e., identifying the reason why the abstract plan is not an actual plan. In planning, this is always done by executing the abstract plan on the original task, until some step in the plan cannot be executed or results in an unexpected effect. While this indeed is a valid interpolation, it is not necessarily the only one, i.e., there could also be other reasons why the same abstract plan is not an actual plan.

We introduce *regression flaws*. Instead of executing the abstract plan on the initial state, we use regression to identify sets of states that are solved by the suffix of the abstract plan. This results in a different type of flaw, which focuses on why the plan suffix fails to solve the task instead of why the plan prefix fails. Refining the abstraction based on regression flaws maintains all important properties of the overall CEGAR process, such as converging to an optimal plan.

Backward interpolation has also been successfully used in the context of model-checking (Hajdu and Micskei 2020). We show that the idea of backward refinements makes even more sense in the context of planning, where abstractions are used to estimate goal distance from any reachable state. In our experiments we notice abstractions obtained with backward refinements are less focused on the initial state, so they typically converge slower towards the optimal plan and get lower values when evaluating the initial state. However, the resulting heuristic is overall more informative, significantly improving the quality of the heuristic to guide the search.

In fact, replacing the standard forward refinements with backward ones leads to a significant improvement in the performance in most domains not only when deriving a single abstraction heuristic, but also when combining multiple abstractions via cost partitioning on the Scorpion planner.

We also experiment with strategies that combine both forward and backward refinements. The results show that they inherit the strengths of both methods: generating heuristics that are as informative as with backward refinement and, at the same time, obtaining heuristics even more informative for the initial state than the original forward refinements.

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

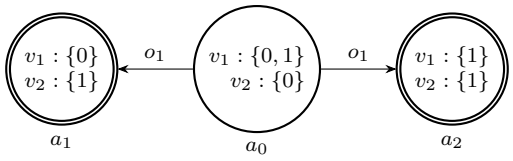

Figure 1: Example of a Cartesian abstraction for a task with two binary variables, $s_0 = \{v_1 \mapsto 0, v_2 \mapsto 0\}$, $G = \{v_2 \mapsto 1\}$, and a single operator o_1 with $pre(o_1) = \{v_2 \mapsto 0\}$, and $\mathit{eff}(o_1) = \{v_2 \mapsto 1\}$.

partial state p is a partial 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 the variable $v \in \mathrm{vars}(p)$ in the partial state p. Two partial states p and c are consistent if $p[v] = c[v]$ for all $v \in \mathrm{vars}(p) \cap \mathrm{vars}(c)$. We denote by $S(p) \subseteq S$ the set of states consistent with p.

A SAS$^+$ task Π is a tuple $\langle V, O, s_0, G\rangle$ where s_0 is the initial state, G is a partial state that describes the goals, and O is a set of operators. An operator $o \in O$ has preconditions $pre(o)$ and effects $\mathit{eff}(o)$, both of which are partial states, and a non-negative cost $\mathsf{cost}(o) \in \mathbb{R}_0^+$. An operator o is applicable in progression in a state s if s is consistent with $pre(o)$. The result of applying o to s is another state $s[\![o]\!]$ where $s[\![o]\!][v] = \mathit{eff}(o)[v]$ if $v \in \mathrm{vars}(\mathit{eff}(o))$ and $s[\![o]\!][v] = s[v]$ otherwise. $s \xrightarrow{o} s'$ is a shorthand for the application of o on s when $s' = s[\![o]\!]$. The *state space* of a task Π is a transition system, $\Theta = \langle S, O, T, s_0, S_G\rangle$, where S is the set of all states, $S_G = \{s \in S \mid s \text{ is consistent with } G\}$ is the set of goal states, and $T = \{(s, o, s') \mid s \xrightarrow{o} s'\}$ is the set of transitions. A *plan* π for s is a sequence of operators $\langle o_1, o_2, \ldots, o_n\rangle$, s.t. the trace $s \xrightarrow{o_1} s_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} s_n$ reaches a goal state $s_n \in S_G$. The cost of π is the summed up cost of its operators. The goal distance from s to the goal, $h^*(s)$, is the minimum cost of any plan for s, or ∞ if no plan exists. A plan for Π is a plan for the initial state, s_0.

Regression reasons backwards, starting from a partial state p and deriving from which states we can reach some state in $S(p)$ by applying some operator (Rintanen 2008; Alcázar et al. 2013). An operator o is applicable in regression in p if p is consistent with $pre^r(o)$ where $pre^r(o) = \mathit{eff}(o) \cup \{pre(o)[v] \mid v \notin \mathrm{vars}(\mathit{eff}(o))\}$.[1] After applying o in regression to p, the resulting partial state p' is defined for $(\mathrm{vars}(p) \setminus \mathrm{vars}(\mathit{eff}(o))) \cup \mathrm{vars}(pre(o))$ and $regr(p, o)[v] = pre(o)[v]$ for $v \in \mathrm{vars}(pre(o))$ and $regr(p, o)[v] = p[v]$ otherwise. We write $p \xleftarrow{o} p'$ as a shorthand.

A common approach to find optimal plans is to use A* search with an admissible heuristic. A *heuristic* is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$. The heuristic is admissible if $h(s) \leq h^*(s)$ for all $s \in S$.

An abstraction is a function $\alpha : S \mapsto S^\alpha$, where S^α is a finite set of abstract states. The abstract state space $\Theta^\alpha = \langle S^\alpha, O, T^\alpha, s_0^\alpha, S_G^\alpha\rangle$ is a homomorphism of the state space, i.e., $T^\alpha = \{(\alpha(s) \xrightarrow{o} \alpha(t) \mid s \xrightarrow{o} t \in T)\}$, $s_0^\alpha = \alpha(s_0)$, $S_G^\alpha = \{\alpha(s) \mid s \in S_G\}$. Each abstraction induces a heuristic function where $h^\alpha(s)$ is the distance from $\alpha(s)$ to the goal in Θ^α. Each abstract state $s^\alpha \in S^\alpha$ is identified with the set of states mapped to it, $S(s^\alpha) = \{s \mid s \in S, \alpha(s) = s^\alpha\}$.

Cartesian abstractions are a type of abstractions where the set of states $S(s^\alpha)$ is Cartesian for all $s^\alpha \in S^\alpha$ (Seipp and Helmert 2018). A set of states is Cartesian if it is of the form $A_1 \times A_2 \times \cdots \times A_n$, where $A_i \subseteq \mathcal{D}_{v_i}$ for all $v_i \in V$. Given a Cartesian set 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 sets $a' = a_1 \cap a_2$ is also a Cartesian set, where $a'[v] = a_1[v] \cap a_2[v]$ for all $v \in V$. Figure 1 shows an example of a Cartesian abstraction, where each abstract state is a Cartesian set, e.g., a_0 has two concrete states $\{v_1 \mapsto 0, v_2 \mapsto 0\}$ and $\{v_1 \mapsto 1, v_2 \mapsto 0\}$.

Note that Cartesian sets generalize partial states. For any partial state p, we can build a Cartesian set $C(p)$ such that $C(p) = S(p)$, by making $C(p)[v] = \{p[v]\}$ if $v \in \mathrm{vars}(p)$ and $C(p)[v] = \mathcal{D}_v$ otherwise. Slightly abusing notation, we will keep this conversion implicit and define the intersection of a Cartesian set a and a partial state p as $a \cap p = a \cap C(p)$.

The most successful technique to obtain informative Cartesian abstractions is CEGAR (Counterexample-Guided Abstraction Refinement) (Seipp and Helmert 2013b, 2018). CEGAR starts with the trivial abstraction with a single abstract state a s.t. $a[v] = \mathcal{D}_v \ \forall v \in \mathrm{vars}(v)$. Then, it is iteratively refined until reaching a termination condition or finding a concrete plan. The refinement loop is shown in Algorithm 1: 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 $s_n \in S_G$, 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. A *flaw* is a tuple $\langle s_i, c\rangle$ consisting of a concrete state $s_i \in S$ and a Cartesian set c. There are three types of flaws, which correspond to different reasons that can cause the execution of τ^α to fail at step i: (1) s_i is the first state in which o_{i+1} is not applicable and c is the set of concrete states in a_i in which o_{i+1} is applicable, i.e. $c = a_i \cap pre(o_{i+1})$. (2) s_i is the first state where o_{i+1} is applicable but its successor is not mapped to a_{i+1}. Then, c is the set of concrete states in a_i from which a_{i+1} is reached when applying o_i. (3) the sequence can be executed but s_n is not a goal state. This results in the flaw $\langle s_n, G\rangle$.

A flaw $\langle s, c\rangle$ is repaired by splitting $\alpha(s)$ into two abstract states d and e with $s \in S(d)$ and $S(c) \subseteq S(e)$. Usually, multiple possible splits exist in different variables to fix the flaw. A split selection strategy is a criterion to choose a split among the ones that fix the flaw (Seipp and Helmert 2018).

The process refines the abstraction until solving the problem either by finding an optimal plan (line 9) or proving the task unsolvable (line 4). The process can be stopped by some termination condition (typically a time or memory limit), resulting in a Cartesian abstraction that induces a heuristic.

[1] Previous work considered as additional precondition in regression that $\mathrm{vars}(p) \cap \mathrm{vars}(\mathit{eff}(o)) \neq \varnothing$, as otherwise the resulting partial state is subsumed.

Algorithm 1: CEGAR main algorithm

Data: $\Pi = \langle V, O, s_0, G\rangle, S^\alpha$; // task, initial abstraction
Result: unsolvable, concrete_plan, final_abstraction

while *not terminationCondition()* **do**
  $\tau^\alpha \leftarrow$ findOptimalTrace(S^α)
  if τ^α = *"no trace"* **then**
    return *task is unsolvable*
  $\varphi \leftarrow$ findFlaw(Π, τ^α)
  if φ = *"no flaw"* **then**
    return *plan extracted from* τ^α
  $S^\alpha \leftarrow$ refine(S^α, φ);
return S^α

Backward Refinement

Our main contribution is to perform the refinement of the abstraction in the backward direction, based on regressing the abstract plan from the goal.

Regression Flaw

Given an abstract plan, we define a regression flaw as an explanation of why a given suffix of the plan is not a solution for the planning task. The definition is similar to that of flaw used in previous work (Seipp and Helmert 2018). However, as the goal is not a fully specified state, regression should be done on partial states, in an attempt to cover all possible ways of reaching the goal with such an abstract plan.

Definition 1 (Regression flaw)**.** *Let Π be a planning task, α a Cartesian abstraction of Π, and $\tau^\alpha = a_0 \xrightarrow{o_1} a_1 \xrightarrow{o_2} \dots \xrightarrow{o_n} a_n$ an abstract plan trace. Let $p_n = G$ and $\langle p_n \xleftarrow{o_n} \dots \xleftarrow{o_1} p_0\rangle$ be the result of performing regression using the reversed sequence of operators. A regression flaw of τ^α is a pair $\langle p_i, c\rangle$ where i is the largest index for which:*

1. *o_i is not backward applicable on p_i (p_i is not consistent with $pre^r(o_i)$). In this case, $c = a_i \cap pre^r(o_i)$ is the set of concrete states in a_i where o_i is backward applicable;*
2. *o_i is backward applicable on p_i but $S(p_{i-1}) \cap S(a_{i-1}) = \varnothing$, i.e., no state in the predecessor is mapped to a_{i-1}. In this case, c is the Cartesian set within a_i that can be reached from some state in a_{i-1} by applying o_i; or*
3. *the entire sequence is well defined but p_0 is incompatible with the initial state s_0, and c is the Cartesian set of s_0.*

Algorithm 2 shows how to find the first regression flaw by generating the sequence of partial states in regression along the plan, checking the conditions above. An important property for the correctness of the approach is that returning "no flaw" means the abstract plan is a plan for the original task. In forward refinements this property is guaranteed, as any plan trace τ^α without forward flaws is always mappable and therefore is a plan for the original task.

Definition 2 (Mappable abstract plan)**.** *An abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} a_1 \xrightarrow{o_2} \dots \xrightarrow{o_n} a_n$ is mappable if the sequence of operators corresponds to an actual plan trace $s_0 \xrightarrow{o_1} s_1 \xrightarrow{o_2} \dots \xrightarrow{o_n} s_n$, and $s_i \in S(a_i)$ for all $i \in [0, n]$.*

Algorithm 2: FindRegressionFlaw

Data: $\Pi = \langle V, O, s_0, G\rangle, \tau^\alpha$; // task, abstract plan trace
Result: Regression Flaw $\langle$partial state, Cartesian set$\rangle$

$p \leftarrow G$
forall $b \xrightarrow{o} a \in$ *reverse(τ^α)* **do**
  if *o is not backward applicable to p* **then**
    return $\langle p, a \cap pre^r(o)\rangle$
  $p' \leftarrow regr(p, o)$
  if $S(p') \cap b \neq \varnothing$ **then**
    return $\langle p, a \cap progr(b, o)\rangle$
  $p \leftarrow p'$
if $p \cap s_0 \neq s_0$ **then**
  return $\langle p, s_0\rangle$
return "no flaw"

However, partial states generated during regression may correspond to multiple abstract states. Therefore, in case (2.), we check whether the intersection is empty instead of requiring the partial state to be included in the abstract state. Due to this, it might happen that abstract plans with no regression flaw are not directly mappable to actual plans.

Proposition 1. *An abstract plan without regression flaws may be not mappable.*

Proof. In the Cartesian abstraction of Figure 1, the optimal abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} a_2$ is not mappable because the state $s_1 = s_0[\![o_1]\!]$ is mapped to a_1 instead of the expected a_2. However, τ^α has no regression flaws because the goal partial state $\{v_2 \mapsto 1\}$ has a non-empty intersection with a_2, the operator o_1 can be applied in the goal partial state $\{v_2 \mapsto 1\}$ reaching the state $\{v_2 \mapsto 0\}$ which has a non-empty intersection with both, a_0 and s_0. □

Nevertheless, regression flaws do keep the most important property, which is that any abstract plan without a flaw corresponds to a plan for the original task.

Theorem 1. *Let Π be a planning task, and α a Cartesian abstraction of Π. If an abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} a_1 \xrightarrow{o_2} \dots \xrightarrow{o_n} a_n$ has no regression flaws, then the sequence of operators $\langle o_1, o_2, \dots, o_n\rangle$ is a plan for the original task.*

Proof. If there is no regression flaw, then the whole regression sequence $\langle p_n \xleftarrow{o_n} p_{n-1} \xleftarrow{o_{n-1}} \dots \xleftarrow{o_1} p_0\rangle$ is well-defined, i.e., each o_i is consistent with $pre^r(o_i)$ and $p_{i-1} = regr(p_i, o_i)$. We show by induction that there exist $s_0, \dots, s_n$ such that $s_i \in S(p_i)$ for all $i \in [0, n]$, and $s_i[\![o_{i+1}]\!] = s_{i+1}$. The base case, $s_0 \in S(p_0)$, holds due to the absence of a type (3.) flaw. For the inductive case, if $s_i \in S(p_i)$ and $p_i = regr(p_{i+1}, o_{i+1})$, then by the definition of regression there must exist $s_{i+1} \in S(p_{i+1})$ such that $s_i[\![o_i]\!] = s_{i+1}$. Finally, $s_n \in S(p_n) = S_G$, so the whole sequence is a plan for Π. □

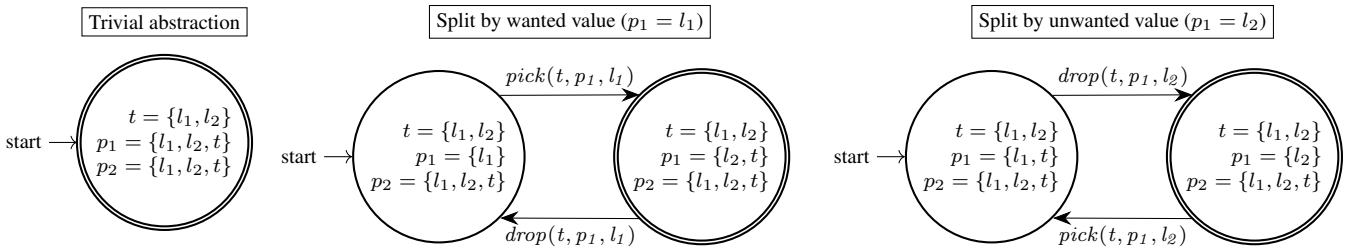

Figure 2: Backward refinement of the trivial abstraction. The regression flaw identifies that the goal disagrees with the initial state and chooses to refine based on $v = p_1$. The value $p_1 = t$ can be assigned to either side, but "unwanted" is preferable.

Splitting Strategy

After finding a regression flaw, we need to refine the abstraction by splitting a Cartesian state into two states. In progression flaws, $\langle s, c\rangle$, the refinement is performed over $\alpha(s)$. In regression flaws, $\langle p, c\rangle$, this is not uniquely defined however, as the states within $S(p)$ may correspond to different abstract states. For a regression flaw, $\langle p, c\rangle$, we will perform the refinement on the abstract state a such that $S(c) \subseteq S(a)$. Note that this is uniquely defined, as in all three types of flaws c corresponds to a subset of the states within a Cartesian state. To guarantee that a can be refined, it is enough to show that it contains more than one concrete state.

Theorem 2. *If $\langle p_i, c\rangle$ is a regression flaw for τ^α, there exists an abstract state a, such that $S(c) \subseteq S(a)$ and $|S(a)| > 1$.*

Proof. Let a be the state used to define c in any of the three types of flaw. $S(c) \subseteq S(a)$ because c is the intersection of a and another Cartesian set.

We show that $|S(a)| > 1$ by contradiction. Assume that $|S(a)| = 1$, then $\{s\} = S(a)$. We consider the three cases corresponding to each type of flaw. Flaw (1) can happen if some o_i is not applicable on p_i in regression. However, since $a_{i-1} \xrightarrow{o_i} a_i$ is part of the abstract plan, by the definition of the abstract state space there must exist some $s_{i-1} \xrightarrow{o_i} s_i$ such that $s_{i-1} \in S(a_{i-1})$ and $s_i \in S(a_i)$. As $a_i = \{s_i\}$, and $S(a_i) \cup S(p_i) \neq \varnothing$ then $s_i \in S(p_i)$. Thus, o is applicable in regression on p_i, reaching a contradiction.

Flaw (2) can happen if some o_i is applicable in regression on p_i but $p_{i-1} \cap a_{i-1} = \varnothing$. Since $a_i = \{s_i\}$, then by the definition of abstract state space there exists $s_{i-1} \in S(a_{i-1})$ s.t. $s_{i-1}[\![o_i]\!] = s_i$. By the definition of regression, $s_{i-1} \in S(p_{i-1})$, reaching a contradiction as $s_{i-1} \in S(p_{i-1} \cap a_{i-1})$.

Flaw (3) happens if $s_0 \notin p_0$. Because no flaw of type (2) exists, $p_0 \cap a_0 \neq \varnothing$. τ^α is an abstract plan trace, so $s_0 \in a_0$. Since $p_0 \cap a$, so $s_0 = s$, so $s_0 \in p_0$, reaching a contradiction. □

To split a into two Cartesian sets d and e, we pick a variable $v \in \text{vars}(p)$ such that $p[v] \notin c[v]$, but $p[v] \in a[v]$. The choice of v can be based on the same strategies defined for forward flaws (Speck and Seipp 2022). Then, a is split by choosing d and e such that $d[v] \cup e[v] = a[v]$, $d[v] \cap e[v] = \varnothing$, and $d[v'] = e[v'] = a[v']$ for all $v' \neq v$. The split is done in such a way that $p[v] \in d[v]$ and $c[v] \subseteq e[v]$, so d is consistent with p and e is consistent with c. All other values of $a[v]$ can be assigned either to d or e.

Figure 2 illustrates this, by showing two possible splits. The original CEGAR algorithm splits the wanted values (the values in $c[v]$) from the rest. This achieves the desirable behavior, of having as many values as far from the goal as possible. In backward refinements, we need to reverse this, by splitting away the unwanted value of the partial state ($p[v]$). This is more desirable because that way all states taking the remaining values ($p_1 = t$ in our example) will get a larger heuristic value. This is most clear in the trivial abstraction, where splitting the wanted values splits only the initial state fact, which provide low improvements to the heuristic.

Corollary 1. *The CEGAR algorithm using regression flaws without termination condition returns an optimal plan if the task is solvable, and otherwise returns "task is unsolvable".*

Proof. At each iteration of CEGAR, three things can happen: (1) there is no abstract plan. As α is an abstraction, this can only happen on unsolvable tasks. (2) There is no flaw and a plan is returned. The plan is valid due to Theorem 1 and optimal, as it is optimal in the abstraction. (3) The abstraction is refined. By Theorem 2, this is always successful. As the number of concrete states in some abstract state is always reduced, eventually the abstraction cannot be refined any longer and either (1) or (2) hold. □

Relation to Forward Refinement

After showing the correctness of CEGAR with regression flaws, the question of whether this is a significant change arises, compared to CEGAR with forward flaws. First, we consider the relationship between both types of flaws.

Theorem 3. *If an abstract plan has a regression flaw, then it has a progression flaw, but not necessarily vice-versa.*

Proof Sketch. ($\Rightarrow$) If the abstract plan has a regression flaw, then it is not executable in regression or $p_i \cap a_i = \varnothing$ at some step i. Therefore, it cannot be executed in progression either. The full proof can be found in the supplementary material (Pozo, Torralba, and Linares López 2023).

($\not\Leftarrow$) The example from Figure 1 shows an abstract plan with a progression flaw but no regression flaws. Any plan without a progression flaw is mappable to a plan trace $\tau = s_0 \xrightarrow{o_1} s_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} s_n$ s.t. $\alpha(s_i) = a_i \; \forall i \in [0, n]$. So, the regression over $p_n \xleftarrow{o_n} p_{n-1}$ is possible and $s_i \in S(p_i) \cap S(a_i)$, so the intersection is not empty at any step. □

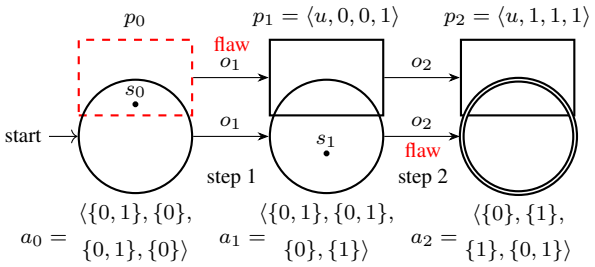

Figure 3: States of an abstract optimal plan (other states of the abstraction and self-loops omitted) for the example of Theorem 4. Circumferences are abstract states, rectangles are partial states and dots are concrete states. The portions of partial states outside their mapped abstract state may be part of any other abstract state. u means undefined variable. If the operator o_1 were applicable, s_0 would be inside p_0.

An initial motivation for using backward refinements is that, intuitively, doing regression over the abstract plan will find flaws on abstract states closer to the goal.[2] This is expected to be beneficial, as a refinement on an abstract state can never increase the goal distance of abstract states closer to the goal. So refining on states closer to the goal has higher chances of propagating the increase of goal distance to other abstract states. Indeed, Speck and Seipp (2022) experimentally showed that strategies refining states closer to the goal provide better heuristics. However, regression flaws are not guaranteed to occur closer to the goal.

Theorem 4. *Let $\tau^\alpha = a_0 \xrightarrow{o_1} \ldots \xrightarrow{o_n} a_n$ be an abstract trace with a progression flaw at step $a_i \xrightarrow{o_i} a_{i+1}$ and a regression flaw at step $a_{j+1} \xleftarrow{o_j} a_j$. Cases where $i > j$ exist.*

Proof. Consider a task with binary variables $V = \{v_1, v_2, v_3, v_4\}$, $s_0 = \{v \mapsto 0 \mid v \in V\}$, $G = \{v_2 \mapsto 1, v_3 \mapsto 1, v_4 \mapsto 1\}$, $O = \{o_1, o_2\}$ with $pre(o_1) = \{v_2 \mapsto 0\}$, $\mathit{eff}(o_1) = \{v_2 \mapsto 1, v_4 \mapsto 1\}$, $pre(o_2) = \{v_2 \mapsto 0, v_3 \mapsto 0\}$, and $\mathit{eff}(o_2) = \{v_3 \mapsto 1\}$.

Figure 3 shows the abstract states along an optimal abstract plan. The forward flaw happens at step 2 because o_2 is not applicable on s_1. The regression flaw happens at step 1 because o_1 is not backward applicable on p_1. □

Note that this happens when concrete states reached in progression are not inside partial states reached in regression, as happens with s_1 in the figure. In this example, both refinements happen on a_1, but it could easily be extended so that the regression flaw happens on some a_i and the progression flaw happens on some a_j with $i < j$. Another difference is that progression flaws focus on plan prefixes, refining an abstract state on which there is at least one reachable state. In contrast, regression flaws are found on abstract states where the heuristic value is correct for at least one of their states, to discriminate the ones where this is not the case.

[2]We confirmed this experimentally. In single abstractions, regression flaws are closer to the goal in $\approx 99\%$ of the cases, though this decreases for additive abstractions (Pozo et al. 2023).

Experiments

We implemented our new refinement algorithms within the Scorpion planner (Seipp, Keller, and Helmert 2020). Our experiments run on the Autoscale 21.11 benchmark set (Torralba, Seipp, and Sievers 2021), which contains 42 domains with 30 tasks each, for a total of 1260 instances. All experiments are limited to 30 minutes and 8 GB of RAM and run in an Ubuntu Linux 20.04 server with an Intel Xeon X3470 processor at 2.93 GHz, 16 GB of RAM and a 1 TB HDD.

Unless explicitly said otherwise, we use Scorpion's default parameters for CEGAR. Specifically, we use incremental search for optimal abstract plans instead of A* (Seipp, von Allmen, and Helmert 2020); set the termination condition as 1 million of non-looping transitions; and choose splits by maximizing the amount of covered flaws ($COVER$), breaking ties by the most refined split ($MAX\_REFINED$), i.e., the one with minimal $\mathit{remaining\ values}/\mathit{domain\ size}$ (Speck and Seipp 2022).

In the Scorpion planner, goals are refined before starting the main CEGAR loop as an optimization. We keep this for the forward refinements, as it improves the results. This is disabled on all configurations using backward and bidirectional refinements, where this was detrimental. Another optimization used on all configurations is, on tasks with a single goal variable, we refine all facts unreachable before the goal on the relaxed planning graph (Blum and Furst 1997).

All experiments use the "wanted" splitting strategy for progression flaws and the "unwanted" strategy for regression flaws. Using "wanted" with backward refinements is still better than forward refinements, but "unwanted" is always better. Details are shown in the supplementary material, available in Zenodo along with code and experimental data (Pozo, Torralba, and Linares López 2023).

Single Abstraction

First, we study the effect of backward refinements by using CEGAR to construct a single Cartesian abstraction. Apart from configurations that only do forward (C_f), and backward (C_b) refinements, we also experiment with three bidirectional strategies that combine both types of refinements:

- C_bd: Interleaves backward and forward refinements, alternating between them in each iteration of the loop.
- $C_\mathrm{b\text{-}f}$: Uses backward refinements with half of the limits (in terms of the time spent and/or number of transitions) and then switches to forward refinements for the rest.
- $C_\mathrm{f\text{-}b}$: Like the previous one but refining first in the forward direction and afterwards in the backward direction.

Table 1 compares all refinement methods in terms of coverage. We distinguish those tasks solved directly with the CEGAR algorithm while building the abstraction, from those solved by A* search using the resulting abstraction to compute the heuristic. Interestingly, the comparison in terms of tasks solved directly by the CEGAR algorithm is completely opposite to the tasks solved by search informed with the resulting abstractions. Our C_b refinements are significantly worse than the baseline C_f in terms of tasks solved by the CEGAR algorithm. And yet, they result in far better

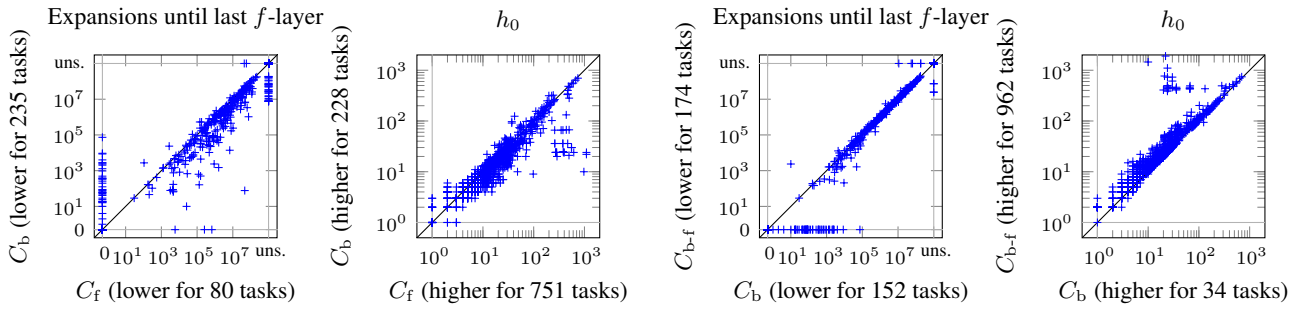

Figure 4: Expansions until last f-layer and initial state value (h_0) of single abstractions: C_f vs. C_b (left) and C_b vs. $C_{b\text{-}f}$ (right).

	CEGAR						A* + CEGAR					
	C_f	C_b	C_{bd}	$C_{b\text{-}f}$	$C_{f\text{-}b}$	Cov	C_f	C_b	C_{bd}	$C_{b\text{-}f}$	$C_{f\text{-}b}$	Cov
C_f	–	**18**	3	0	**7**	112	–	2	1	1	1	397
C_b	2	–	0	0	0	78	**17**	–	**10**	**5**	**14**	**427**
C_{bd}	**6**	**21**	–	0	**9**	123	**9**	2	–	1	**4**	414
$C_{b\text{-}f}$	**10**	**24**	**8**	–	**13**	**131**	**14**	2	**6**	–	**9**	426
$C_{f\text{-}b}$	**6**	**17**	3	1	–	116	**8**	2	2	1	–	411

Table 1: Coverage of different refinement methods when considering only the tasks solved within the CEGAR algorithm (left side), and search with the resulting abstraction (right side). The cell in row x and column y shows the number of domains where method x solved more tasks than method y. "Cov" indicates the total number of tasks solved.

heuristics, solving more tasks. This can be observed in terms of total coverage, where C_f solves 34 more instances within the CEGAR process but 30 instances less when combined with search. And the same happens on a per-domain basis, where in terms of tasks solved by CEGAR, C_f is superior to C_b in 18 domains and only worse in 2 of them. However, when combined with search, C_b is better in 17 domains and only worse in 2 instances from DataNetwork and Driverlog.

Figure 4 shows additional insights on the comparison of C_f and C_b. This confirms that C_f's abstractions obtain a higher heuristic value for the initial state (h_0) in most of the tasks,[3] and solve more tasks with no search (0 expansions). However, C_b's abstractions induce more informed heuristics in almost all cases, leading to fewer expansions.

Combining both types of flaws in any of our bidirectional approaches produces abstractions with the strengths of both. So, despite solving 1 fewer problem, $C_{b\text{-}f}$ forges a more robust heuristic with higher h_0 value and more problems solved without search. All bidirectional strategies are similar in performance, but $C_{b\text{-}f}$ gets the best coverage while $C_{f\text{-}b}$ is more focused on h_0. C_{bd} has a mixed behavior but doing the forward refinements at the end gets better results.

To understand the counter-intuitive observation that the variants leading to a notably higher heuristic value for the initial state are overall less informed (A* performs more expansions), we take a closer look at the distribution of heuristic values on selected instances. Table 4 shows the average h value, h_0 and search time of some representative instances. DataNetwork is one of the few domains where C_b performs worse than C_f, and the average heuristic value is also lower. However, in most instances the average heuristic value of C_b is higher despite having a smaller h_0, which explains why it has a better coverage and lower search times. h_0 is more influential in problems with few expanded nodes, like the Logistics example. But in larger instances of the same and other domains, where more expansions are required, the average h value is paramount. Another factor is the distribution of the heuristic value (Figure 5), since having fewer states with low h values may be more helpful in the search than having a very high value on some states (Holte et al. 2006). Indeed, we observe that this happens in some domains like Transport, where despite reaching a high h_0 value, C_f has many states with very low h values.

[3] In all h_0 plots the maximum value has been bounded to 2000. This excludes the ParcPrinter domain, with a similar behavior but orders of magnitude larger values, making visualization difficult.

	C_f^{add}	C_b^{add}	C_{bd}^{add}	$C_{b\text{-}f}^{add}$	$C_{f\text{-}b}^{add}$	Cov
C_f^{add}	–	2	1	3	1	480
C_b^{add}	**7**	–	3	**4**	**6**	490
C_{bd}^{add}	**7**	**6**	–	**6**	**5**	**493**
$C_{b\text{-}f}^{add}$	**6**	1	3	–	**4**	484
$C_{f\text{-}b}^{add}$	**2**	2	1	3	–	483

Table 2: Coverage of CEGAR with additive abstractions.

	S_f	S_f^{batch}	S_b	S_{bd}	$S_{b\text{-}f}$	$Compl^2$	S^{2018}	Cov
S_f	–	2	4	3	4	**22**	**20**	727
S_f^{batch}	**5**	–	4	4	5	**22**	**20**	730
S_b	**11**	**8**	–	**7**	**4**	**22**	**21**	**743**
S_{bd}	**7**	5	2	–	2	**23**	**20**	737
$S_{b\text{-}f}$	**9**	**6**	1	**5**	–	**22**	**22**	740
$Compl^2$	17	16	15	16	15	–	**19**	653
S^{2018}	10	9	8	9	8	**19**	–	687

Table 3: Coverage of Scorpion variants.

Additive Abstractions

CEGAR is usually used in a saturated cost partitioning scheme, generating abstractions that focus on the cost of

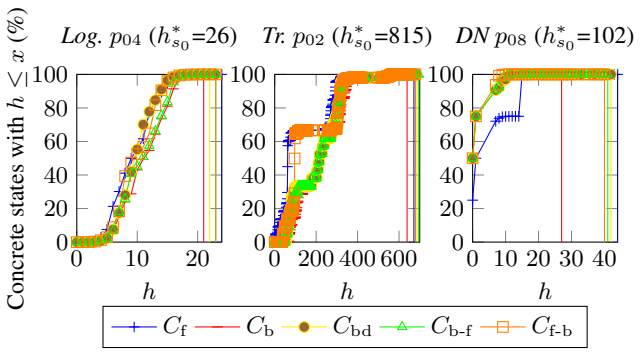

Figure 5: Accumulated h distribution in selected instances.

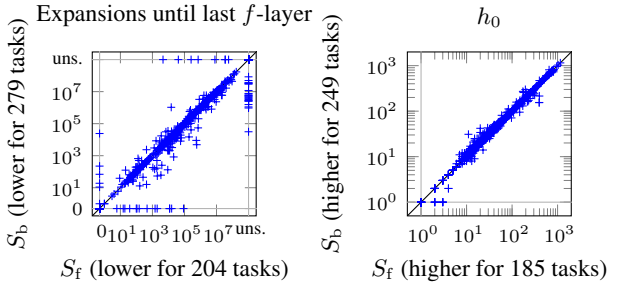

Figure 6: Expansions until last f-layer and h_0 of S_f vs S_b.

reaching a single landmark or goal. This helps because CEGAR's abstractions do not tend to focus in different aspects of the problem (Seipp and Helmert 2014).

Table 2 shows the coverage using additive abstractions. In this case, no problem is solved by the CEGAR loop directly, because each abstraction is only computed on a subproblem. Similar trends can be observed, though using additive abstractions slightly reduces the impact of the refinement method. The coverage is higher than using a single abstraction in all variants, with 58-83 more problems solved. C_b^{add} solves 10 more problems than C_f^{add}, improving coverage in 7 domains and being detrimental in only 2 domains. C_{bd}^{add} solves 3 more problems than C_b^{add}, being the best heuristic and worse than C_f^{add} in only 1 domain. $C_{b\text{-}f}^{add}$ and $C_{f\text{-}b}^{add}$ solve fewer problems but still more than C_f^{add}.

Scorpion Configuration

The last experiments use the Scorpion configuration, the CEGAR-based state-of-the-art planner (Seipp 2018, 2023). This configuration combines additive CEGAR abstractions and Pattern Databases (PDBs) (Culberson and Schaeffer 1998; Edelkamp 2001) in an online saturated cost partitioning scheme (Seipp 2021). Furthermore, it uses the h^2 preprocessor (Alcázar and Torralba 2015) and atom-centric stubborn sets pruning (Alkhazraji et al. 2012; Wehrle et al. 2013; Wehrle and Helmert 2014; Röger et al. 2020). A minor difference with the previous experiments is that Scorpion configuration uses the *MAX_REFINED* splitting strategy, breaking ties in favor of the minimum position in the partial ordering of the causal graph.

Logistics p_{04}	C_f	C_b	C_{bd}	$C_{b\text{-}f}$	$C_{f\text{-}b}$
h_0	**24**	21	23	23	23
Avg. h	9.9	**11.7**	10.3	**11.3**	10.4
Search t. (s)	**1.1**	3.8	2.0	2.1	1.7
Transport p_{02}	C_f	C_b	C_{bd}	$C_{b\text{-}f}$	$C_{f\text{-}b}$
h_0	672	637	683	**696**	679
Avg. h	133	**221**	206	208	165
Search t. (s)	1.5	**0.6**	0.8	0.7	1.0
DataNet p_{08}	C_f	C_b	C_{bd}	$C_{b\text{-}f}$	$C_{f\text{-}b}$
h_0	**44**	27	42	41	40
Avg. h	**5.8**	2.2	2.2	2.2	2.1
Search t. (s)	**101**	163	163	165	175

Table 4: h statistics in some selected instances.

Table 3 shows the coverage of all Scorpion configurations, including S^{batch}, Scorpion with batch forward refinements (Speck and Seipp 2022), and the best non-portfolio planners from the International Planning Competition (IPC) 2018: Complementary2 (Franco et al. 2017; Franco, Lelis, and Barley 2018) ($Compl^2$) and Scorpion 2018 (S^{2018}).

Backward refinements can once again improve the overall performance in the Scorpion configuration, solving 16 more problems improving in 11 domains and only being detrimental on 4. $S_{b\text{-}f}$ is again the best bidirectional strategy but close to S_{bd}. Our enhancement is orthogonal to batch refinements, not improving in exactly the same domains. Total coverage is around 250 higher than using additive abstractions without Scorpion's improvements, so online ordering of partitions, preprocessing and pruning have a huge impact in coverage.

Another finding is that, contrary to the experiments on single abstractions, h_0 is slightly higher for S_b than S_f, as shown in Figure 6. So, this shortcoming of backward refinements is overcome when combined with Scorpion. As a final remark, S_b and $S_{b\text{-}f}$ are the best alternatives, solving more problems than the non-portfolio planners of the last IPCs: Scorpion 2023 (S_f) and Complementary2. Interestingly, Complementary2 solves 90 fewer problems, though it solves more problems in around 15 domains.

Conclusions

CEGAR is a state-of-the-art technique to generate abstraction heuristics, iteratively refining the abstraction by finding flaws in abstract plans. However, the mechanism to obtain these flaws has barely been explored. In this paper we propose backward refinements, which find flaws using regression from the goal. We show that the resulting abstractions induce better heuristics, improving the overall planner performance. In addition, we introduce bidirectional strategies that use refinements in both directions. They obtain a coverage similar to backward refinements and are also more robust for obtaining higher heuristic values on the initial state.

This opens a new research avenue on how to identify flaws in abstract plans, and how to combine this with other orthogonal enhancements, such as refining multiple plans at once (Speck and Seipp 2022) or online refinements (Eifler and Fickert 2018; Seipp 2021).

Acknowledgments

This work was partially funded by grant PID2021-127647NB-C21 from MCIN/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.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A Stubborn Set Algorithm for Optimal Planning. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

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

Blum, A.; and Furst, M. L. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1–2): 281–300.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In Emerson, E. A.; and Sistla, A. P., eds., *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, 154–169.

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

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.

Eifler, R.; and Fickert, M. 2018. Online Refinement of Cartesian Abstraction Heuristics. In Bulitko, V.; and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 46–54. AAAI Press.

Franco, S.; Lelis, L. H. S.; and Barley, M. 2018. The Complementary2 Planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 32–36.

Franco, S.; Torralba, Á.; Lelis, L. H. S.; and Barley, M. 2017. On Creating Complementary Pattern Databases. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 4302–4309. IJCAI.

Hajdu, Á.; and Micskei, Z. 2020. Efficient Strategies for CEGAR-Based Model Checking. *Journal of Automated Reasoning*, 64(6): 1051–1091.

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.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over Multiple Pattern Databases Speeds up Heuristic Search. *Artificial Intelligence*, 170(16–17): 1123–1136.

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

Kreft, R.; Büchner, C.; Sievers, S.; and Helmert, M. 2023. Computing Domain Abstractions for Optimal Classical Planning with Counterexample-Guided Abstraction Refinement. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press.

Pozo, M.; Torralba, Á.; and Linares López, C. 2023. When CEGAR Meets Regression: A Love Story in Optimal Classical Planning. Supplementary Material, Code, Experimental Results and Scripts. 10.5281/zenodo.10405579.

Rintanen, J. 2008. Regression for Classical and Nondeterministic Planning. In Ghallab, M.; Spyropoulos, C. D.; Fakotakis, N.; and Avouris, N., eds., *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, 568–572. IOS Press.

Röger, G.; Helmert, M.; Seipp, J.; and Sievers, S. 2020. An Atom-Centric Perspective on Stubborn Sets. In Harabor, D.; and Vallati, M., eds., *Proceedings of the 13th Annual Symposium on Combinatorial Search (SoCS 2020)*, 57–65. AAAI Press.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 362–367. AAAI Press.

Seipp, J. 2018. Fast Downward Scorpion. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 77–79.

Seipp, J. 2021. Online Saturated Cost Partitioning 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)*, 317–321. AAAI Press.

Seipp, J. 2023. Scorpion 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Seipp, J.; and Helmert, M. 2013a. Additive Counterexample-guided Cartesian Abstraction Refinement. In desJardins, M.; and Littman, M. L., eds., *Late-Breaking Developments in the Field of Artificial*

Intelligence – Papers Presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013) – AAAI Technical Report WS-13-17, 119–121. AAAI Press.

Seipp, J.; and Helmert, M. 2013b. 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. 2014. Diverse and Additive Cartesian Abstraction Heuristics. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. 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. 2017. Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.

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.; von Allmen, S.; and Helmert, M. 2020. Incremental Search for Counterexample-Guided Cartesian Abstraction Refinement. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 244–248. AAAI Press.

Speck, D.; and Seipp, J. 2022. New Refinement Strategies for Cartesian Abstractions. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 348–352. AAAI Press.

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.

Wehrle, M.; and Helmert, M. 2014. Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The Relative Pruning Power of Strong Stubborn Sets and Expansion Core. 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)*, 251–259. AAAI Press.