Abstraction Heuristics for Classical Planning Tasks with Conditional Effects

Martín Pozo[1] and **Jendrik Seipp**[2]
[1]Universidad Carlos III de Madrid
[2]Linköping University
marcosmartin.pozo@alumnos.uc3m.es, jendrik.seipp@liu.se

Abstract

In planning tasks, conditional effects model action outcomes that depend on the current state of the world. Conditional effects are a crucial modeling feature since compiling them away can cause an exponential growth in task size. However, only a few admissible heuristics support them. To add abstraction heuristics to this set, we show how to compute projections, Cartesian abstractions and merge-and-shrink abstractions for tasks with conditional effects. Our experiments show that these heuristics are competitive with—and often surpass—the state-of-the-art for conditional-effect tasks.

1 Introduction

Conditional effects allow the modelling of action effects that occur only when specified conditions are satisfied in the current state [Nebel, 2000; Helmert, 2009]. Introduced to classical planning in the context of ADL [Pednault, 1989], they have since become an essential tool for compactly representing complex planning tasks. For example, they underlie recent planning formulations for discovering new algorithms for matrix multiplication [Speck *et al.*, 2023] and for transforming quantum circuits [Shaik and van de Pol, 2024]. Their importance is also reflected in the growing number of International Planning Competition (IPC) domains that feature conditional effects [Taitler *et al.*, 2024].

Since designing optimal planners that natively handle conditional effects remains challenging, a common alternative is to compile them away [Gazen and Knoblock, 1997; Haslum, 2013]. However, Nebel [2000] showed that compiling away conditional effects entails an exponential increase in model size if plan lengths are constrained to grow at most linearly. If polynomial growth of plan lengths is allowed, compilation becomes feasible for certain tasks using techniques that exploit the structure of the conditional effects [Gerevini *et al.*, 2024; Percassi *et al.*, 2024]. Nevertheless, the resulting compiled tasks often overwhelm planners due to a combinatorial explosion in the search space, which usually grows exponentially with the depth of the search [Gerevini *et al.*, 2024].

Given these limitations, developing optimal classical planners that can handle conditional effects natively is paramount. While symbolic search can be extended to support conditional effects simply by encoding the effects in the transition relation [Edelkamp and Helmert, 2001; Kissmann *et al.*, 2014; Speck *et al.*, 2025], the situation is more challenging for informed search techniques that rely on admissible heuristics.

To date, only a few admissible heuristics have been extended to natively support conditional effects, such as the $h^{\max}$ heuristic [Bonet and Geffner, 2001] and the LM-Cut heuristic [Helmert and Domshlak, 2009; Röger *et al.*, 2014]. More recently, Büchner *et al.* [2024] showed how to compute *abstractions* for *factored tasks*, a restricted class of conditional-effect tasks where each effect condition considers only the effect variable. As for tasks without conditional effects, abstractions of factored tasks are *induced*, i.e., each abstract transition corresponds to a concrete transition.

To handle arbitrary conditional effects, we over-approximate the abstract transition system and obtain non-induced abstractions. Dropping the requirement that abstractions are induced is necessary for handling conditional effects, because even for projections it is NP-hard to decide whether an abstract transition is induced [Büchner *et al.*, 2024]. Adding conditional-effect support requires only minor modifications for projections, the abstractions underlying pattern databases (PDBs) [Edelkamp, 2001; Haslum *et al.*, 2007; Sievers *et al.*, 2012] and merge-and-shrink (M&S) abstractions [Sievers and Helmert, 2021].

Our main contribution, however, concerns Cartesian abstractions [Seipp and Helmert, 2018], where extending support for conditional effects requires not only redefining how to decide the presence of abstract transitions but also adapting the counterexample-guided abstraction refinement (CEGAR) algorithm, which remains the only method for constructing Cartesian abstractions. We show that all flaw detection strategies from the literature, *forward* [Seipp and Helmert, 2018], *backward* [Pozo *et al.*, 2024b] and *sequence* flaws [Pozo *et al.*, 2024a], can be adapted to handle conditional effects.

Our experimental results for factored tasks show that the CEGAR algorithm for PDBs designed for this setting is preferable to the general counterpart, but for tasks with arbitrary conditional effects, our techniques solve more tasks than state-of-the-art methods across many benchmark domains.

2 Background

We begin by introducing the necessary background on classical planning, abstractions and the CEGAR algorithm.

2.1 Planning Tasks

We consider finite-domain (FDR) planning tasks with action costs but without axioms and derived variables [Helmert, 2009]. Such planning tasks $\Pi = \langle V, O, I, G\rangle$ are defined by a set of finite-domain variables V, a finite-set of operators O, an initial state I and a goal description G. Each variable $v \in V$ has a finite domain $\mathcal{D}_v$. A *partial state* p is a partial variable assignment over some variables $vars(p) \subseteq V$. A (concrete) state s is a full assignment, i.e., $vars(s) = V$. We write $p[v]$ for the value assigned to variable $v \in vars(p)$ in partial state p. Two partial states p_1 and p_2 are consistent, $p_1 \cong p_2$, if $p_1[v] = p_2[v]$ for all $v \in vars(p_1) \cap vars(p_2)$. We let S denote the set of all states and let $[\![p]\!] \subseteq S$ be the set of states consistent with p. We often treat (partial) states as sets of *atoms* $v \mapsto x$, where $x \in \mathcal{D}_v$.

Each *operator* $o = \langle pre(o), \mathit{eff}(o), cost(o)\rangle \in O$ consists of the partial state $pre(o)$, called its *precondition*, a finite set of *effects* $\mathit{eff}(o)$ and a non-negative cost $cost(o) \in \mathbb{R}_0^+$. Each effect $e \in \mathit{eff}(o)$ is a pair $\langle conds(e), atom(e)\rangle$, where $conds(e)$ is a partial state, called the *effect conditions*, and $v \mapsto x = atom(e)$ is the *effect atom*, where v and x are called the *effect variable* and *effect value*. Operator o is *forward-applicable* or, in short, *applicable* in state s iff s is consistent with $pre(o)$. In the resulting state $s' = progr(s, o)$ we have $s'[v] = d$ for all effects $\langle conds(e), v \mapsto x\rangle$ with $conds(e) \cong s$ and $s'[v] = s[v]$ for all variables v where no effect is triggered. We assume that all operator effects are non-conflicting. Note that $conds(e)$ can be empty, in which case the effect is triggered in all states $s \in S$. We write $s \xrightarrow{o} s'$ as a shorthand whenever o is applicable in s and $s' = progr(s, o)$.

2.2 Search in Transition Systems

A task $\Pi = \langle V, O, I, G\rangle$ induces the labelled transition system $\Theta = \langle S, O, T, I, S_G\rangle$, where $S_G = \{s \in S \mid s \cong G\}$ is the set of goal states and $T = \{\langle s, o, s'\rangle \mid s, s' \in S, o \in O, s \xrightarrow{o} s'\}$ is the set of transitions. An *s-plan* 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 a plan is the summed-up cost of its operators. An s-plan is *optimal* if it has the minimum cost among all s-plans. An I-plan is a plan for Π. *Regression* reasons backwards, starting from a partial state p' and deriving from which states $[\![p]\!]$ we can reach some state in $[\![p']\!]$ by applying an operator [Rintanen, 2008; Alcázar *et al.*, 2013]. We write $p \xleftarrow{o} p'$ as a shorthand.

Heuristics. One of the most successful approaches to find optimal plans is to use A* with an admissible heuristic [Dechter and Pearl, 1985]. A *heuristic* is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. The perfect heuristic h^* assigns to each state s the cost of an optimal s-plan, or ∞ if no s-plan exists. A heuristic is *admissible* if $h(s) \leq h^*(s)$ for every state $s \in S$, it is *consistent* if $h(s) \leq h(s') + cost(o)$ for all $s \xrightarrow{o} s' \in T$ and it is *goal-aware* if $h(s) = 0$ for all goal states $s \in S_G$. Consistent and goal-aware heuristics are admissible [Pearl, 1984].

Cost Partitioning. *Cost partitioning* allows for the combination of several admissible heuristics $\mathcal{H}$ into a single heuristic while preserving admissibility [Katz and Domshlak, 2010; Pommerening *et al.*, 2015]. It does so by distributing the cost of each operator among the heuristics, such that the sum of the assigned costs does not exceed the original operator cost. Formally, given a tuple of heuristics $\mathcal{H} = \langle h_1, \ldots, h_n\rangle$ for task Π, the cost functions $\mathcal{C} = \langle c_1, \ldots, c_n\rangle$ form a cost partitioning for $\mathcal{H}$ if $\sum_{i=1}^n c_i(o) \leq cost(o)$ for all $o \in O$. The resulting *cost-partitioned heuristic* is $h^{\mathcal{C}}(s) = \sum_{i=1}^n h_i(c_i, s)$, where $h_i(c_i, s)$ is the heuristic value of h_i for state s evaluated under cost function c_i.

2.3 Abstractions

We focus on heuristics based on *abstraction* [Helmert *et al.*, 2008]. An abstraction is a surjective function $\alpha : S \to S^\alpha$, where S^α is a finite set of abstract states. The abstract transition system $\Theta^\alpha = \langle S^\alpha, O, T^\alpha, I^\alpha, S_G^\alpha\rangle$ is a homomorphism of the concrete transition system, i.e., $T^\alpha = \{(\alpha(s) \xrightarrow{o} \alpha(t) \mid s \xrightarrow{o} t \in T)\}$, $I^\alpha = \alpha(I)$, $S_G^\alpha = \{\alpha(s) \mid s \in S_G\}$. We also call a homomorphism a *conservative* abstraction [Sievers and Helmert, 2021]. An abstraction is *induced* (a *strict* homomorphism) if each abstract transition corresponds to a concrete transition. Each abstract state $s^\alpha \in S^\alpha$ is identified with the set of states mapped to it, $[\![s^\alpha]\!] = \{s \in S \mid \alpha(s) = s^\alpha\}$. *Abstraction heuristics* are a family of admissible heuristics h^α, where $h^\alpha(s)$ is the cost of the cheapest $\alpha(s)$-plan in the abstract transition system.

Projections. The most basic type of abstraction is a projection α_P to a subset of variables $P \subseteq V$. We call P a *pattern* and projection-based heuristics *pattern database* (PDB) heuristics [Edelkamp, 2001].

Merge-and-Shrink Abstractions. The most general type of abstraction for classical planning are *merge-and-shrink abstractions* (M&S) [Sievers and Helmert, 2021]. M&S begins by computing a projection for each variable $v \in V$ to obtain its initial *factored transition system*. Then it iteratively selects two factors and replaces them by their synchronized product (*merge*). To adhere to a given size limit, in between such *merge* operations, M&S may *shrink* a factor by applying an abstraction function on it. The two remaining operations that the M&S framework uses, *label reduction* and *state pruning*, are not important for our contributions, so we refer the reader to Sievers and Helmert [2021]. The M&S procedure ends once there is only a single factor left or in case a time limit is reached, in which case we can combine the estimates of the remaining factors with cost partitioning [Sievers *et al.*, 2024].

Cartesian Abstractions. On the generality scale, *Cartesian abstractions* lie in between projections and M&S abstractions. An abstract state s^α is *Cartesian* if $[\![s^\alpha]\!]$ has the form $A_1 \times A_2 \times \cdots \times A_n$, where $A_i \subseteq \mathcal{D}_{v_i}$ for all $v_i \in V$. (We treat V as a tuple where it simplifies notation.) If all states in an abstraction are Cartesian, we call the abstraction *Cartesian* [Seipp and Helmert, 2018]. 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 and b is the Cartesian set c, with $c[v]{=}a[v] \cap b[v]$ for all $v \in V$. We write $a \cong p$ for (partial) state p if there is a state $s \in [\![a]\!]$ with $s \cong p$. Also, for any such p, we can build a Cartesian set $\mathcal{C}(p) = b$ such that $[\![b]\!]{=}[\![p]\!]$, by setting $b[v]{=}\{p[v]\}$ if $v \in vars(p)$ and $b[v]{=}\mathcal{D}_v$ otherwise.

Algorithm 1: CEGAR loop for a task Π.

1 $\Theta^\alpha \leftarrow$ TrivialAbstraction(Π)
2 **while** *not TerminationCondition()* **do**
3 $\tau^\alpha \leftarrow$ FindOptimalTrace(Θ^α)
4 **if** τ^α = *"no trace"* **then**
5 **return** *task is unsolvable*
6 $\varphi \leftarrow$ FindFlaw(Π, τ^α)
7 **if** φ = *"no flaw"* **then**
8 **return** *plan extracted from* τ^α
9 $\Theta^\alpha \leftarrow$ Refine(Θ^α, φ)
10 **return** Θ^α

2.4 Cartesian Abstraction Refinement

Currently, the only way to build Cartesian abstractions is via counterexample-guided abstraction refinement (CEGAR) [Clarke *et al.*, 2000; Seipp and Helmert, 2013; Seipp and Helmert, 2018], shown in Algorithm 1. This iterative procedure starts from the *trivial abstraction*, which consists of a single abstract state a_0 such that $a_0[v]{=}\mathcal{D}_v$ for all $v \in V$. Then, the abstraction is refined until finding a plan, proving the task unsolvable or hitting a termination condition like a time or memory limit.

In each iteration, an optimal abstract plan trace $\tau^\alpha{=}a_0 \xrightarrow{o_1} \ldots \xrightarrow{o_n} a_n$ is executed in the concrete state space, resulting in a trace $s_0 \xrightarrow{o_1} \ldots \xrightarrow{o_m} s_m, m \leq n$ (line 6). An abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} \ldots \xrightarrow{o_n} a_n$ is *mappable* to a concrete plan trace $\tau = s_0 \xrightarrow{o_1} \ldots \xrightarrow{o_n} s_n$ iff $\alpha(s_i) = a_i$ for all $i \in [0, n]$. If τ^α is mappable and $s_n = s_m \in S_G$, then it is an optimal plan for the task (line 8). If it fails at some step i, this is a flaw and the abstraction is refined by splitting the abstract state a_i into two, in such a way that the same flaw cannot happen again (line 9). A *flaw* is a pair $\langle s_i, c\rangle$ of a concrete state $s_i \in S$ and a Cartesian set c.

There are three causes for a flaw, which correspond to different reasons that can cause the execution of τ^α to fail at step i: (1) *precondition flaw*: s_i is the first state in which o_{i+1} is inapplicable and c is the set of states consistent with a_i in which o_{i+1} is applicable, i.e., $c = a_i \cap \mathcal{C}(pre(o_{i+1}))$; (2) *deviation flaw*: s_i is the first state where o_{i+1} is applicable but its successor is not consistent with a_{i+1}, and c is the set of states consistent with a_i from which a_{i+1} is reached when applying o_i, i.e., $c = a_i \cap regr(a_{i+1}, o_{i+1})$;[1] and (3) *goal flaw*: the sequence can be executed but s_n is not a goal state, which results in the flaw $\langle s_n, a_n \cap \mathcal{C}(G)\rangle$.

3 Abstractions for Conditional-Effect Tasks

Heuristics based on (conservative) abstractions are admissible because all concrete plans are preserved, which guarantees that abstract plans cannot be more expensive than their concrete counterparts. To preserve admissibility in the presence of conditional effects, we need to ensure that the abstraction remains conservative. We have to give up on inducedness, though, and instead over-approximate the set of abstract transitions, since even for the basic class of projections, it is NP-hard to test whether an abstract transition is induced [Büchner *et al.*, 2024].

[1]The regression of a Cartesian set is not Cartesian for operators with conditional effects. We deal with this in Section 4.2.

Algorithm 2: Compute outgoing transitions from abstract state a via operator o in a projection to pattern P.

1 **if** $pre(o) \not\cong a$ **then**
2 **return** $\varnothing$
3 **forall** $v \in P$ **do**
4 $post[v] \leftarrow \{x \mid \langle C, v \mapsto x\rangle \in \mathit{eff}(o), C_{|P} \cong a\} \cup \{a[v]\}$
5 **forall** $\langle C, v \mapsto x\rangle \in \mathit{eff}(o) \mid C_{|P} \cong a, vars(C) \subseteq P$ **do**
6 $post[v] \leftarrow \{x\}$
7 **return** $\{a \xrightarrow{o} b \mid b \in \bigtimes_{v \in P} post[v]\}$

3.1 Projections

We define the *projection* of a (partial) state s to a set of variables P as the partial state $s_{|P}$ that contains only the atoms of s for the variables in P. Algorithm 2 shows how to compute the over-approximation of outgoing transitions from abstract state a induced by operator o in a projection to P. If the precondition of o projected to P is not satisfied in a, we return the empty set. Otherwise, we compute for each variable v in P the set $post[v]$ of possible values for v after applying o, accounting for the possibility that no effect is triggered for v and v keeps the value it has in a. We set $post[v]$ to $\{x\}$ for $v \in P$ if there is an effect $\langle C, v \mapsto x\rangle$ that is guaranteed to be triggered. Finally, we compute the Cartesian product of all sets $post[v]$ for $v \in P$ to obtain the set of successor states b.

Theorem 1. *Projection heuristics computed with Algorithm 2 for tasks with conditional effects are admissible.*

Proof Sketch. Abstractions computed by Algorithm 2 are conservative, since the algorithm considers all subsets of effects that could be triggered. □

3.2 Merge-and-Shrink Abstractions

Since merge-and-shrink abstractions start from (atomic) projections, we can use Algorithm 2 to over-approximate them.

Theorem 2. *Merge-and-shrink heuristics that use Algorithm 2 for computing the initial factors are admissible for tasks with conditional effects.*

Proof Sketch. M&S is compositional. If the initial factors are conservative and only conservative transformations are applied, the resulting abstraction is also conservative. □

4 Cartesian Abstractions for Tasks with Conditional Effects

We now describe how to adapt Cartesian abstractions and the CEGAR algorithm to tasks with conditional effects. Of the five subroutines in Algorithm 1, TrivialAbstraction, TerminationCondition and FindOptimalTrace are not related to conditional effects, so no modification is needed. FindFlaw retrieves the next flaw and Refine repairs it by splitting the abstract state and rewiring transitions. We first explain the rewiring of transitions, and then describe the modifications needed for finding the different types of flaws.

Algorithm 3: Compute outgoing transitions from abstract state a via operator o in Cartesian abstraction α.

```
if pre(o) ⊭ a then
    return ∅
post ← a ∩ C(pre(o))
forall ⟨C, v ↦ x⟩ ∈ eff(o) with C ≅ a do
    post[v] ← post[v] ∪ {x}
forall ⟨C, v ↦ x⟩ ∈ eff(o) do
    if a[w] = {y} for all w ↦ y ∈ C then
        post[v] ← {x}
return {a -o-> b | b ∈ S^α, b ∩ post ≠ ∅}
```

4.1 Rewiring Transitions

A flaw $\langle s, c\rangle$ is repaired by splitting the abstract state $\alpha(s)$ into two child abstract states d and e with $s \in d$ and $c \subseteq e$. Such a split is always possible and is done by partitioning the abstract domain $\alpha(s)[v]$ into $d[v]$ and $e[v]$ for a suitable *split variable* $v \in V$. After such a split, the transitions adjacent to $\alpha(s)$ must be rewired for d and e.

Algorithm 3 shows how to compute the outgoing transitions from a Cartesian state a via operator $o \in O$. If the precondition of o is not satisfied in a, we return the empty set. Otherwise, we use three steps to compute the possible values $post[v]$ for each variable $v \in V$ after applying o in a. First, we intersect a with the Cartesian partial state $pre(o)$ to obtain the subset $[\![post]\!] \subseteq [\![a]\!]$ of states in a in which o is applicable. Second, we iterate over the effects $\langle C, v \mapsto x\rangle \in \mathit{eff}(o)$ that *might* be triggered in a and add x to $post[v]$. Third, we iterate over the effects $\langle C, v \mapsto x\rangle \in \mathit{eff}(o)$ that *are guaranteed* to be triggered in a and replace $post[v]$ with $\{x\}$. There is a transition $a \xrightarrow{o} b$ if $b \in S^\alpha$ contains at least one value from $post[v]$ for each variable $v \in V$.

Example 1. `Briefcase` [Pednault, 1988] is a domain with three types of operators: *store* puts an object into the briefcase, *takeout* leaves an object from the briefcase at the current location, and *move* moves all objects in the briefcase and the briefcase itself between two given locations. In our example task from this domain, a worker who is initially at work (W) has forgotten an important document (D) and must return home (H), take it and return to work (the only goal is to have the document at work). The variables for this task are the location of the briefcase v_B, with domain $\{H, W\}$; the location of the document v_D, with domain $\{H, W\}$; and whether the document is in the briefcase, v_I, with domain $\{\bot, \top\}$. The *move* operators are defined as $move(\ell, m) = \langle pre = \{v_B \mapsto \ell\}, \mathit{eff} = \{\langle\{v \mapsto \ell\}, v \mapsto m\rangle \mid v \in V\}, cost = 1\rangle$. Figure 1 shows the rewiring after splitting abstract state a into d and e. Since v_I is abstracted in all states, the *store* and *takeout* operators only induce self-loops. After the split, the transitions that were adjacent to a are rewired to e, as e is the only new state that contains the effect atom and precondition $v_B \mapsto H$. Moreover, the *move* self-loops for a become transitions between d and e.

Theorem 3. *Abstraction heuristics obtained with CEGAR using Algorithm 3 are admissible for conditional-effect tasks.*

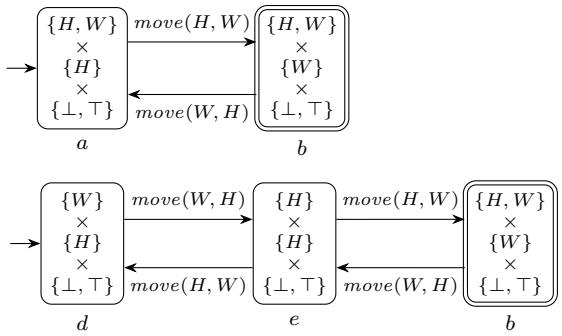

Figure 1: For an abstraction of the task in Example 1, we split a for atom $v_B \mapsto H$, yielding states d and e. Self-loops are omitted and the variable order is v_B, v_D, v_I.

Proof Sketch. The refinement loop maintains a conservative abstraction at all steps, by considering all effects that could be triggered. □

Theorem 4. *Cartesian CEGAR with Algorithm 3 may yield non-induced abstractions for tasks with conditional effects.*

Proof. Consider planning task $\Pi = \langle V, O, I, G\rangle$, with $V{=}\{v_0, v_1, v_2\}$, $\mathcal{D}_{v_0}{=}\mathcal{D}_{v_1}{=}\mathcal{D}_{v_2}{=}\{0, 1\}$, $O{=}\{o\}$, $o = \langle pre = \{\}, \mathit{eff} = \left\{ \begin{array}{l} \langle\{v_0 \mapsto 0\}, \{v_1 \mapsto 1\}\rangle, \\ \langle\{v_0 \mapsto 1\}, \{v_2 \mapsto 1\}\rangle \end{array} \right\}, cost = 1\rangle$, $I = \{v_0 \mapsto 0, v_1 \mapsto 0, v_2 \mapsto 0\}$, and $G = \{v_2 \mapsto 1\}$. After separating the values of v_1 and v_2 in all abstract states, we obtain an abstract transition system that has the four states $a_0 = \{0,1\} \times \{0\} \times \{0\}$, $a_1 = \{0,1\} \times \{1\} \times \{1\}$, $a_2 = \{0,1\}{\times}\{0\}{\times}\{1\}$, and $a_3 = \{0,1\}{\times}\{1\}{\times}\{0\}$, with an optimal plan trace $\tau^\alpha = \langle a_0 \xrightarrow{o} a_1\rangle$. Transition $a_0 \xrightarrow{o} a_1$ has no counterpart in the concrete transition system, as both states mapped to a_0 satisfy only one of the effect conditions. □

4.2 Progression Flaws

Having discussed how to rewire the transition system after a given split, we now focus on deciding which splits to perform in the first place. Out of the three flaw causes—precondition flaws, deviation flaws and goal flaws—conditional effects impact only deviation flaws, as operator applicability only depends on operator preconditions (not their effects) and the goal status of a state does not depend on any operator. There are three main strategies for finding (deviation) flaws in CEGAR: progression flaws, regression flaws and sequence flaws, and we now discuss how to adapt them to tasks with conditional effects, beginning with progression flaws.

A deviation exists for a transition $a \xrightarrow{o} b$ in an abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} \ldots \xrightarrow{o_i} a \xrightarrow{o} b \xrightarrow{o_k} \ldots \xrightarrow{o_n} a_n$, if $s \in [\![a]\!]$ and $t \notin [\![b]\!]$ for the state $s = \mathit{progr}(I, o_1, \ldots, o_i)$ and the state $t = \mathit{progr}(s, o)$. In such a case, the deviation flaw is $\langle s, c\rangle$ with $c = a \cap \mathit{regr}(b, o)$. An obstacle here is that regression is not Cartesian for operators with conditional effects. However, we only need to know the possible values for each variable to detect deviations. Therefore, we over-approximate the regression of a Cartesian state b over operator o with conditional effects as the Cartesian set $\mathit{regr}(b, o)$. For each variable

$v \in V$, we define the set of possible values before applying o as $regr(b, o)[v] =$

$$\begin{cases} \{pre(o)[v]\} & \text{if } v \in vars(pre(o)) \\ \mathcal{D}_v & \text{if } \langle C, v \mapsto x \rangle \in \mathit{eff}(o), x \in b[v] \\ b[v] \cup \{x\} & \text{if } \langle \{v \mapsto x, ...\}, w \mapsto y \rangle \in \mathit{eff}(o), y \in b[w] \\ b[v] & \text{otherwise,} \end{cases} \quad (1)$$

where we always use the first case that applies. We omit the case where $regr(b, o)[v]$ is empty, as it cannot happen during the CEGAR loop.

It is often the case that multiple variables are the cause for a single deviation. Fixing all causes simultaneously would require to modify the CEGAR algorithm to apply multiple splits in the same iteration of the refinement loop. However, it is simpler to split only for a single variable in each iteration, and to account for the rest of the flaw in later iterations. A similar situation already occurs for tasks without conditional effects when several preconditions are not satisfied by a concrete state. Here, existing methods refine for only one of the preconditions in each iteration.

Example 2. A deviation flaw exists in state e of the bottom abstraction shown in Figure 1, because $move(H, W)$ does not change the location of the document if the document is not inside the briefcase. Thus, $v_D \mapsto H$ continues to hold after applying $move(H, W)$ in the concrete state $\{v_B \mapsto H, v_D \mapsto H, v_I \mapsto \bot\}$, but not in the abstract state b. This flaw can be fixed by splitting off $v_I \mapsto \top$ from e.

Theorem 5. *Let $\tau^\alpha = a_0 \xrightarrow{o_1} a_1 \xrightarrow{o_2} \dots \xrightarrow{o_n} a_n$ be an abstract plan trace of a task Π with conditional effects. Then, τ^α is mappable iff τ^α has no progression flaw.*

Proof Sketch. ($\Rightarrow$) If τ^α is mappable, then there is a concrete trace τ without deviation flaws. Since those are the only flaws that can be caused by conditional effects, we know that τ has no flaws at all. ($\Leftarrow$) If there are no flaws, then there are also no deviation flaws, which means that τ^α is mappable. □

4.3 Regression Flaws

Regression flaws are found by executing an abstract plan in regression from the goals [Pozo *et al.*, 2024b]. Given a transition $a \xrightarrow{o} b \in \tau^\alpha = a_0 \xrightarrow{o_1} \dots \xrightarrow{o_i} a \xrightarrow{o} b \xrightarrow{o_k} \dots \xrightarrow{o_n} a_n$, a deviation exists if $p \cong b$ and $q \not\cong a$ for the Cartesian state $p = \mathcal{C}(regr(G, o_n, \dots, o_k))$ and the Cartesian state $q = \mathcal{C}(regr(p, o))$. Handling deviation flaws in regression is easier than in progression because effect conditions are not evaluated in the abstract state b that will be split, but in a. Therefore, the way to find deviations is the same as without conditional effects except that the effect is possibly not triggered, so a deviation may happen in an atom $v \mapsto x$ only if $p[v] = x$ and $|b[v]| > 1$.

Regression flaws are found in an optimal abstract plan by applying the plan from G to I in a Cartesian state space. Again, the obstacle here is that regression over operators with conditional effects is not Cartesian. Therefore, representing the regression of a Cartesian state b over an operator o requires a disjunction of up to c^f Cartesian sets, where f is the number of effect atoms of o that hold in b and c is the highest number of conditions of those effects. To avoid this blow-up, we apply regression as the Cartesian over-approximation in Equation 1. But this over-approximation comes with a compromise: regressed Cartesian sets are supersets of the actual regression, so they may not find all regression flaws.

Theorem 6. *Let $\tau^\alpha = a_0 \xrightarrow{o_1} \dots \xrightarrow{o_n} a_n$ be an abstract plan trace for a task Π. Then, although τ^α has no regression flaw with regression by Cartesian over-approximation, the sequence $\langle o_1, o_2, \dots, o_n \rangle$ may not be a plan for Π.*

Proof. Let Π be the planning task from the proof of Theorem 4. The trace $a_0 \xrightarrow{o} a_1$ has no regression flaw with Cartesian over-approximation because $I \in [\![regr(G, o) = \{0, 1\} \times \{0, 1\} \times \{0, 1\}]\!]$ but $\langle o \rangle$ is not a plan for Π because $progr(I, o) = \{v_0 \mapsto 0, v_1 \mapsto 1, v_2 \mapsto 0\} \notin S_G$. □

Given an abstract plan trace $\tau^\alpha = a_0 \xrightarrow{o_1} \dots \xrightarrow{o_n} a_n$ and a backward plan trace $\tau = p_m \xleftarrow{o_{m+1}} \dots \xleftarrow{o_n} p_n, m \in [0, n)$, we mitigate this issue with two adjustments. The first one is to intersect each p_i with a_i, that is, we start the search of flaws from a_n instead of G and intersect each $regr(p_i, o_i)$ with a_{i-1}. However, this does not guarantee finding a flaw when $\langle o_1, o_2, \dots, o_n \rangle$ is not a plan for the task [Pozo *et al.*, 2024b], so the CEGAR algorithm may finish before finding a solution or reaching the termination condition. The second adjustment addresses this issue by searching for a progression flaw as a fallback when no regression flaw is found.

Theorem 7. *Let $\tau^\alpha = a_0 \xrightarrow{o_1} \dots \xrightarrow{o_n} a_n$ be an abstract plan trace for a task Π. Then, τ^α is mappable iff τ^α has no regression flaw with the progression flaw fallback.*

Proof Sketch. A progression flaw is searched if no regression flaw is found, so the proof of Theorem 5 applies. □

4.4 Sequence Flaws

Sequence flaws allow finding flaws in the middle of a plan by continuing the search for flaws "behind" a progression flaw or "before" a regression flaw in an abstract plan trace [Pozo *et al.*, 2024a]. This is achieved by two *sequence relaxations*: allowing to apply an inapplicable operator o by assigning unmet preconditions to variables $v \notin vars(\mathit{eff}(o))$; and the *undeviation* of a successor Cartesian state r by setting $r[v] = a[v]$ for all $v \in V$ where $r[v] \cap a[v] = \varnothing$.

The deviation of sequence flaws in progression and regression is exactly the same as their "first-flaw" counterparts, since these relaxations do not affect deviations. Nevertheless, an important difference exists compared to sequence flaws for tasks without conditional effects: given a successor abstract state b and a Cartesian state r', intersecting them as in previous work [Pozo *et al.*, 2024a] is not sufficient to guarantee that there is no deviation flaw. This is the case because the absence of deviation requires that the set of outgoing transitions from r' is a subset of outgoing transitions from b. This is true for the intersection without conditional effects but not for an operator o with an effect $e \in \mathit{eff}(o)$ s.t. $conds(e) \cong r'$, $conds(e) \not\cong b$. However, there is a sufficient condition that guarantees the absence of deviation flaws for operators with conditional effects: $r' \subseteq b$. If this condition holds, then we do not need to check for deviation flaws. Otherwise, we perform the deviation check.

5 Experiments

We implemented our algorithms in the Scorpion planning system [Seipp *et al.*, 2020a], which is an extension of Fast Downward [Helmert, 2006]. For all configurations we use the h^2 preprocessor [Alcázar and Torralba, 2015] and limit time and memory for each planner run to 30 minutes and 8 GiB, respectively. We use Downward Lab [Seipp *et al.*, 2017] and run experiments on Xeon Gold 6130 processors.

As our benchmark set, we use the domains with conditional effects from the last two IPCs, the domains used by Röger *et al.* [2014], the matrix multiplication domain [Speck *et al.*, 2023], and the domain for transforming quantum circuits into CNOT-only layouts [Shaik and van de Pol, 2024]. This benchmark set consists of 1963 tasks from 21 domains. All code, benchmarks, and experiment data is available online [Pozo and Seipp, 2025].

We evaluate our techniques in four subsections: computing a single abstraction, multiple abstractions of the same abstraction type, combinations of multiple abstraction types, and a comparison versus baselines. Table 1 shows coverage results.

5.1 Single Abstraction

We begin by comparing a single merge-and-shrink (M&S) to a single Cartesian abstraction. For M&S, we use the recommended values for all parameters [Sievers, 2018] and up to 100 000 abstract states. For Cartesian abstractions, we use incremental search to find abstract plans [Seipp *et al.*, 2020b] and a time limit of 900 seconds for the CEGAR loop.

We focus on the "M&S" and "Single Cartesian" columns in Table 1 for now. Among these algorithms, $h^{\texttt{MS}}$ solves the highest number of tasks overall, mainly due its high coverage for CNOT tasks. Cartesian abstractions with regression flaws and progression flaw fallback (B) yield the highest coverage among Cartesian abstractions, and they dominate $h^{\texttt{MS}}$ in all domains with general conditional effects except for `Miconic` and the CNOT domains because refining closer to goal states yields more informative heuristics. However, Cartesian abstractions using sequence flaws (S) solve the most tasks in all factored domains and `Briefcase`, where fewer regression flaws are found and the progression flaw fallback is needed often.

5.2 Multiple Abstractions

We compare saturated cost partitioning (SCP) over Cartesian abstractions for landmark and goal subtasks[2] [Seipp and Helmert, 2018] with SCP over M&S abstractions ($h^{\texttt{MS}}_{\texttt{SCP}}$) [Sievers *et al.*, 2020], and Sys-SCP PDBs combined by SCP ($h^{\texttt{PDB}}_{\texttt{SCP}}$) [Seipp, 2019].

Again, regression flaws with the progression flaw fallback ($\mathcal{B}$) obtain the highest coverage for Cartesian abstractions for all domains except the `FSC Domains`. However, SCP over Cartesian abstractions solves fewer tasks than a single abstraction in `Rubik's Cube`, CNOT domains and factored domains, where the interaction between goals is more intricate and not captured by subtasks. PDBs solve many `Miconic` tasks and $h^{\texttt{MS}}_{\texttt{SCP}}$ has its strength in factored tasks and CNOT domains.

[2]We do not compute domain abstractions for subtasks because they do not support conditional effects [Seipp and Helmert, 2018].

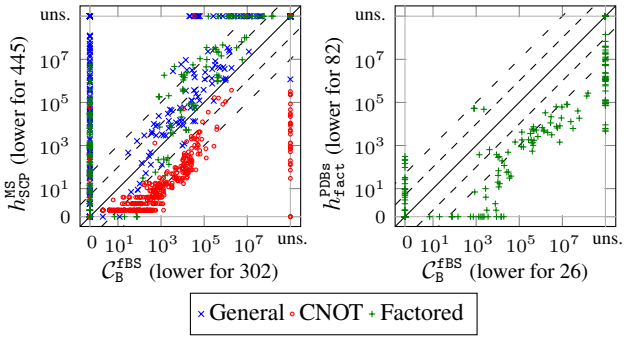

Figure 2: Expansions before the last f-layer for the best combination of abstractions, SCP over M&S and PDBs for factored tasks.

5.3 Combinations of Abstractions

Finally, we combine abstractions of different types via saturated cost partitioning. As the base ingredient, we use PDBs, because they proved to be complementary to Cartesian abstractions. To this, we add different collections of Cartesian abstractions (see caption of Table 1). We use at most 100 seconds to compute PDBs and at most 50 million non-looping transitions in total for Cartesian abstractions. Table 1 shows that $\mathcal{C}^{\texttt{fBS}}_{\texttt{B}}$ is the strongest among the combinations in all domains. It combines multiple Cartesian abstractions over subtasks using regression flaws, a single Cartesian abstraction over the full task using regression flaws and a single Cartesian abstraction over the full task using sequence flaws. In total, this configuration solves 39 more tasks than the second-best combination, $\mathcal{C}_{\texttt{B}}$, mainly because repairing flaws for the full task refines parts of the abstract state space that are kept coarse for individual landmark and goal subtasks, especially in CNOT domains and factored tasks.

5.4 State of the Art

As baselines, we use LM-Cut with context splitting ($h^{\texttt{LMC}}$) [Röger *et al.*, 2014] and symbolic bidirectional search (`Sym`) [Speck *et al.*, 2025]. For factored tasks, we use a single Cartesian abstraction ($h^{\texttt{Cart}}_{\texttt{fact}}$) and multiple PDBs ($h^{\texttt{PDBs}}_{\texttt{fact}}$) [Büchner *et al.*, 2024] as our baselines.

$h^{\texttt{LMC}}$ is very good for `Miconic`—where delete-relaxation works well because each operator appears at most once in optimal plans—but it solves even fewer tasks than a single Cartesian abstraction in the other domains. `Sym` is the strongest baseline, especially for factored tasks, but it is still weaker than $h^{\texttt{PDBs}}_{\texttt{fact}}$ for these tasks. However, $\mathcal{C}^{\texttt{fBS}}_{\texttt{B}}$ dominates all baselines in 9 domains, solving 59 more tasks in total than `Sym`. $h^{\texttt{MS}}_{\texttt{SCP}}$ is the best technique for CNOT domains, but it struggles in `Miconic`. Finally, for factored tasks the dedicated algorithm that computes PDBs is the clear winner in all domains, solving 51 more tasks than $\mathcal{C}^{\texttt{fBS}}_{\texttt{B}}$. Figure 2 shows that the heuristic quality of $\mathcal{C}^{\texttt{fBS}}_{\texttt{B}}$ is higher than $h^{\texttt{MS}}_{\texttt{SCP}}$ except in CNOT domains, but lower than $h^{\texttt{PDBs}}_{\texttt{fact}}$ for factored tasks.

		Baselines		Factored		M&S	Single Cartesian			SCP Abs.		SCP Cartesian			Combinations		
Domain	#Tasks	$h^{\texttt{LMC}}$	Sym	$h^{\texttt{Cart}}_{\texttt{fact}}$	$h^{\texttt{PDBs}}_{\texttt{fact}}$	$h^{\texttt{MS}}$	F	B	S	$h^{\texttt{MS}}_{\texttt{SCP}}$	$h^{\texttt{PDB}}_{\texttt{SCP}}$	$\mathcal{F}$	$\mathcal{B}$	$\mathcal{S}$	$\mathcal{C}_{\text{F}}$	$\mathcal{C}_{\text{B}}$	$\mathcal{C}^{\texttt{fBS}}_{\text{B}}$
General Conds.	543	296	316	–	–	239	245	245	240	250	322	245	255	245	330	339	**341**
Briefcase	50	9	9	–	–	8	9	9	10	8	12	7	7	7	16	16	**17**
Caldera	20	10	10	–	–	12	12	12	12	12	12	12	**17**	12	12	**17**	**17**
CalderaSplit	20	8	**11**	–	–	7	8	8	8	8	8	8	**11**	8	8	9	9
Citycar	20	16	**18**	–	–	16	15	16	14	16	16	16	16	16	16	17	17
FSC Domains	57	**20**	**20**	–	–	19	**20**	19	19	19	19	**20**	19	19	19	19	19
GED Domains	26	15	**20**	–	–	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Miconic	150	142	**150**	–	–	88	85	83	83	88	147	83	84	83	147	147	147
Nurikabe	20	12	11	–	–	12	12	12	12	12	13	**14**	**14**	**14**	**14**	**14**	**14**
Rubik's Cube	20	7	6	–	–	**11**	**11**	**11**	9	**11**	10	7	8	7	10	10	10
Settlers	20	8	9	–	–	9	10	10	10	10	**12**	11	**12**	**12**	**12**	**12**	**12**
Spider	20	11	8	–	–	6	11	11	11	14	15	15	15	15	15	17	**18**
TO Domains	120	38	**44**	–	–	31	32	34	32	32	38	32	32	32	41	41	41
CNOT Domains	992	631	688	–	–	745	677	707	652	**762**	707	594	600	594	725	726	733
CNOT	219	196	210	–	–	**215**	207	214	204	**215**	212	192	193	192	213	213	214
CNOT Hard	526	189	237	–	–	284	226	247	205	**301**	250	163	167	163	266	267	273
CNOT Map	247	**246**	241	–	–	**246**	244	**246**	243	**246**	245	239	240	239	**246**	**246**	**246**
Factored Tasks	428	135	207	182	**247**	172	156	155	180	176	163	136	141	139	166	166	196
Cavediving	17	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Matrix Mult.	11	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
Burnt Pancakes	100	30	49	40	**53**	40	37	36	41	40	38	32	35	35	38	38	45
Pancakes	100	35	52	44	**59**	43	39	39	45	43	39	37	39	37	41	41	51
Rubik's Cube 2	100	37	50	47	**66**	46	41	45	46	51	44	32	32	32	45	45	51
Topspin	100	22	45	40	**58**	32	28	24	37	31	31	24	24	24	31	31	38
Total	1963	1062	1211	182	247	1156	1078	1107	1072	1188	1192	975	996	978	1221	1231	**1270**

Table 1: Number of solved tasks per domain for different algorithms divided into baselines; abstractions for factored tasks; M&S; single Cartesian abstractions; saturated cost partitioning (SCP) heuristics over M&S abstractions and projections; SCP heuristics over Cartesian abstractions; and combinations of different types of abstractions. $h^{\texttt{LMC}}$ is the LM-Cut heuristic, Sym is bidirectional symbolic search, $h^{\texttt{Cart}}_{\texttt{fact}}$ computes a single Cartesian abstraction for factored tasks, $h^{\texttt{PDBs}}_{\texttt{fact}}$ computes PDBs for factored tasks with CEGAR, $h^{\texttt{MS}}$ is the M&S heuristic with SCC-DFP merging and bisimulation shrinking, and $h^{\texttt{MS}}_{\texttt{SCP}}$ and $h^{\texttt{PDB}}_{\texttt{SCP}}$ are M&S and PDB heuristics combined via SCP. For single/multiple Cartesian abstractions F/$\mathcal{F}$ (forward) denotes progression flaws, B/$\mathcal{B}$ (backward) regression flaws, and S/$\mathcal{S}$ sequence flaws. $\mathcal{C}_{\text{F}}$ adds Cartesian abstractions of landmark and goal subtasks refined for progression flaws, $\mathcal{C}_{\text{B}}$ instead uses regression flaws, and $\mathcal{C}^{\texttt{fBS}}_{\text{B}}$ takes the latter and adds a single abstraction of the full task with regression flaws and another one in the last flaw. Bold values denote the highest coverage per domain.

6 Conclusions

We showed how to support conditional effects in PDBs, M&S, and Cartesian abstractions, including support for regression and sequence flaws. Our strongest algorithm, which combines PDBs and Cartesian abstractions, solves many more tasks than symbolic search, the previous state of the art.

CEGAR is also suitable for domain abstractions, which fall in between projections and Cartesian abstractions on the generality scale [Kreft *et al.*, 2023]. Future work could extend them to support conditional effects.

Acknowledgments

We thank Malte Helmert for his valuable feedback on the implementation of conditional-effect support for merge-and-shrink abstractions.

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). This work was also supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[Alcázar and Torralba, 2015] Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. ICAPS 2015*, pages 2–6, 2015.

[Alcázar *et al.*, 2013] Vidal Alcázar, Daniel Borrajo, Susana Fernández, and Raquel Fuentetaja. Revisiting regression in planning. In *Proc. IJCAI 2013*, pages 2254–2260, 2013.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[Büchner *et al.*, 2024] Clemens Büchner, Patrick Ferber, Jendrik Seipp, and Malte Helmert. Abstraction heuristics for factored tasks. In *Proc. ICAPS 2024*, pages 40–49, 2024.

[Clarke *et al.*, 2000] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. CAV 2000*, pages 154–169, 2000.

[Dechter and Pearl, 1985] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[Edelkamp and Helmert, 2001] Stefan Edelkamp and Malte Helmert. The model checking integrated planning system (MIPS). *AI Magazine*, 22(3):67–71, 2001.

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In *Proc. ECP 2001*, pages 84–90, 2001.

[Gazen and Knoblock, 1997] B. Cenk Gazen and Craig A. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proc. ECP 1997*, pages 221–233, 1997.

[Gerevini *et al.*, 2024] Alfonso E. Gerevini, Francesco Percassi, and Enrico Scala. An effective polynomial technique for compiling conditional effects away. In *Proc. AAAI 2024*, pages 20104–20112, 2024.

[Haslum *et al.*, 2007] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI 2007*, pages 1007–1012, 2007.

[Haslum, 2013] Patrik Haslum. Optimal delete-relaxed (and semi-relaxed) planning with conditional effects. In *Proc. IJCAI 2013*, pages 2291–2297, 2013.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, pages 162–169, 2009.

[Helmert *et al.*, 2008] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Explicit-state abstraction: A new method for generating heuristic functions. In *Proc. AAAI 2008*, pages 1547–1550, 2008.

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

[Helmert, 2009] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535, 2009.

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

[Kissmann *et al.*, 2014] Peter Kissmann, Stefan Edelkamp, and Jörg Hoffmann. Gamer and Dynamic-Gamer – Symbolic search at IPC 2014. In *IPC-8 Planner Abstracts*, pages 77–84, 2014.

[Kreft *et al.*, 2023] Raphael Kreft, Clemens Büchner, Silvan Sievers, and Malte Helmert. Computing domain abstractions for optimal classical planning with counterexample-guided abstraction refinement. In *Proc. ICAPS 2023*, pages 221–226, 2023.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

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

[Pednault, 1988] Edwin P. D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Comput. Intell.*, 4:356–372, 1988.

[Pednault, 1989] Edwin P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR 1989*, pages 324–332, 1989.

[Percassi *et al.*, 2024] Francesco Percassi, Enrico Scala, and Alfonso E. Gerevini. Optimised variants of polynomial compilation for conditional effects in classical planning. In *Proc. SoCS 2024*, pages 100–108, 2024.

[Pommerening *et al.*, 2015] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proc. AAAI 2015*, pages 3335–3341, 2015.

[Pozo and Seipp, 2025] Martín Pozo and Jendrik Seipp. Code and data for the IJCAI 2025 paper "Abstraction Heuristics for Classical Planning Tasks with Conditional Effects". https://doi.org/10.5281/zenodo.15397351, 2025.

[Pozo *et al.*, 2024a] Martín Pozo, Álvaro Torralba, and Carlos Linares López. Gotta catch 'em all! sequence flaws in CEGAR for classical planning. In *Proc. ECAI 2024*, pages 4287–4294, 2024.

[Pozo *et al.*, 2024b] Martín Pozo, Álvaro Torralba, and Carlos Linares López. When CEGAR meets regression: A love story in optimal classical planning. In *Proc. AAAI 2024*, pages 20238–20246, 2024.

[Rintanen, 2008] Jussi Rintanen. Regression for classical and nondeterministic planning. In *Proc. ECAI 2008*, pages 568–572, 2008.

[Röger *et al.*, 2014] Gabriele Röger, Florian Pommerening, and Malte Helmert. Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In *Proc. ECAI 2014*, pages 765–770, 2014.

[Seipp and Helmert, 2013] Jendrik Seipp and Malte Helmert. Counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2013*, pages 347–351, 2013.

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

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

[Seipp *et al.*, 2020a] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67:129–167, 2020.

[Seipp *et al.*, 2020b] Jendrik Seipp, Samuel von Allmen, and Malte Helmert. Incremental search for counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2020*, pages 244–248, 2020.

[Seipp, 2019] Jendrik Seipp. Pattern selection for optimal classical planning with saturated cost partitioning. In *Proc. IJCAI 2019*, pages 5621–5627, 2019.

[Shaik and van de Pol, 2024] Irfansha Shaik and Jaco van de Pol. Optimal layout-aware CNOT circuit synthesis with qubit permutation. In *Proc. ECAI 2024*, pages 4207–4215, 2024.

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

[Sievers *et al.*, 2012] Silvan Sievers, Manuela Ortlieb, and Malte Helmert. Efficient implementation of pattern database heuristics for classical planning. In *Proc. SoCS 2012*, pages 105–111, 2012.

[Sievers *et al.*, 2020] Silvan Sievers, Florian Pommerening, Thomas Keller, and Malte Helmert. Cost-partitioned merge-and-shrink heuristics for optimal classical planning. In *Proc. IJCAI 2020*, pages 4152–4160, 2020.

[Sievers *et al.*, 2024] Silvan Sievers, Thomas Keller, and Gabriele Röger. Merging or computing saturated cost partitionings? a merge strategy for the merge-and-shrink framework. In *Proc. ICAPS 2024*, pages 541–545, 2024.

[Sievers, 2018] Silvan Sievers. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proc. SoCS 2018*, pages 90–98, 2018.

[Speck *et al.*, 2023] David Speck, Paul Höft, Daniel Gnad, and Jendrik Seipp. Finding matrix multiplication algorithms with classical planning. In *Proc. ICAPS 2023*, pages 411–416, 2023.

[Speck *et al.*, 2025] David Speck, Jendrik Seipp, and Álvaro Torralba. Symbolic search for cost-optimal planning with expressive model extensions. *Journal of Artificial Intelligence Research*, 82:1349–1405, 2025.

[Taitler *et al.*, 2024] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024.