Is This a Good Decision? Action Optimality Checking in Classical Planning

Jan Eisenhut[1], Daniel Fišer[2], Wheeler Ruml[3], Jörg Hoffmann[1,4]

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2]Aalborg University, Denmark
[3]University of New Hampshire, USA
[4]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{eisenhut,hoffmann}@cs.uni-saarland.de, danfis@danfis.cz, ruml@cs.unh.edu

Abstract

Heuristic search is a prominent method for plan generation in classical planning. Here we address its use for a new problem that we baptize *action optimality checking (AOC)*: checking whether a given action a is optimal in a given state s. AOC has various potential uses, e.g. quality assurance for learned action policies through checking example policy decisions. A vanilla algorithm for AOC is to run two A* searches, on each of s and the outcome state s' of applying a. We show that one can do much better than this. We introduce early termination criteria across multiple searches. Beyond this, we introduce AOCA*, which performs a single search on s that gives preference to paths going through s'. Our experiments show that AOCA* is superior to the vanilla algorithm as well as other multiple-search configurations, consistently across three different state-of-the-art heuristic functions. A version of this work has been accepted at ECAI'25.

1 Introduction

Heuristic search is a prominent method for plan generation in AI Planning. Here we consider *classical planning*, where the initial state is fully known and the action outcomes are deterministic. The task of plan generation is then to find an action sequence that leads from the initial state to the goal, given the semantics of states and actions as specified in the underlying PDDL model (Haslum et al. 2019). Prior work has come up with a wealth of informative heuristic functions in this context (e.g., Haslum and Geffner 2000; Bonet and Geffner 2001; Hoffmann and Nebel 2001; Edelkamp 2001; Haslum et al. 2007; Helmert and Domshlak 2009; Helmert et al. 2014; Pommerening et al. 2014; Seipp and Helmert 2018), in particular with admissible heuristics allowing to find optimal plans that minimize summed-up action cost.

Here we consider the use of heuristic search not for plan generation, but for a problem that has, to the best of our knowledge, not been considered in the planning literature before: *action optimality checking (AOC)*. Given a state s and an action a, does a start an optimal plan for s?

Action optimality checking is natural if the action a was suggested for the state s by some method that we do not have full trust in. Learned action policies π fit this profile. An action policy is a function $\pi\colon S \to A$ from states to actions. If π is a machine learning model, for example a neural action policy (Groshev et al. 2018; Toyer et al. 2020; Ståhlberg, Bonet, and Geffner 2022; Wang and Thiébaux 2024), then π comes without any inherent guarantees, and it is unclear if we can trust individual decisions $\pi(s) = a$. AOC is a means for quality assurance in this context, e.g., applied to sampled states from smaller domain instances where AOC is feasible.

A specific use of AOC arises in the context of policy testing in AI Planning (Steinmetz et al. 2022; Eisenhut et al. 2023, 2024). Such testing methods find *bug* states t, from which the policy run is sub-optimal. In particular, if the policy run from t reaches the goal but t is classified as a bug, then we know that π must take at least one sub-optimal action decision below t. But which decision is it? AOC along the policy path below t provides the answer.

AOC may also have other uses. To give some examples, if a supposedly optimal planner outputs a sub-optimal plan, one may use AOC along that plan to find a specific wrong decision for debugging. In the approach to contrastive explanations by Krarup et al. (2021), a user question "Why is action b used in state s, rather than action a?" could be answered by solving AOC for s and a. Similarly, there is a potential application in the field of goal recognition, where agents are assumed to act optimally: If an agent chooses an action a in a state s that is sub-optimal for some goal G, then the goal G can be ruled out since otherwise the agent would not act optimally. Generally, user-suggested actions can be verified by AOC.

One could also use approximations of AOC during search to quickly prune, or commit to, an action. The methods we introduce here can serve as quick approximations through early termination. Torralba (2017) introduces a sufficient criterion, based on dominance functions, proving that a is optimal in s. As part of our experiments, we design related criteria showing that a is sub-optimal.

Here we focus on the AOC problem per se, i.e., not in the context of any specific application context. Our contribution lies in the design and analysis of algorithms dedicated to AOC, and in their empirical evaluation. Throughout, we assume that s is solvable—the unsolvable case is not interesting as, there, it does not matter which action is applied. We show that AOC is PSPACE-complete, even given this assumption.

A vanilla algorithm is to run two A* searches, on each of s and the outcome state s' of applying a. The optimal plan costs for s and s' determined this way tell us whether or

not a is optimal in s. However, one can do much better than this. We introduce early termination criteria across multiple searches, in particular using the current lower bound computed by A* on s' against an upper bound computed by another algorithm on s. Beyond this, we show that one can decide AOC with a single search from s, through a variant of A* that we baptize *optimality-check A* (AOCA*)*. Within the search space of AOCA*, s' and its descendants are tagged. Ties are broken in favor of tagged nodes. AOCA* can terminate once the first goal node is expanded, just as A*; and it can terminate early if the open list contains only or no tagged nodes.

We run experiments on the International Planning Competition (IPC) benchmarks, including all domains from the optimal classical track without conditional effects after grounding. We construct two AOC problem instances for the initial state of each IPC benchmark instance. We run the vanilla algorithm, a range of more advanced algorithms relying on multiple searches, and AOCA*. AOCA* is superior to all the other algorithms, consistently across three state-of-the-art heuristic functions, namely LM-cut (Helmert and Domshlak 2009), Cartesian abstractions (Seipp and Helmert 2018), and merge-and-shrink abstractions (Helmert et al. 2014). For further comparison, we run sufficient criteria based on dominance functions, including Torralba's (Torralba 2017) criterion and several related ones. These turn out to be much inferior to search in terms of coverage.

2 Background

A STRIPS planning task P is a tuple $P = (\mathcal{F}, \mathcal{A}, I, G)$, where $\mathcal{F}$ is a set of facts, and $\mathcal{A}$ is a set of actions. A **state** $s \subseteq \mathcal{F}$ is a set of facts, $I \subseteq \mathcal{F}$ is the **initial state** and $G \subseteq \mathcal{F}$ is a **goal** specification. An **action** $a \in \mathcal{A}$ is defined by its precondition $\mathrm{pre}(a) \subseteq \mathcal{F}$, add effect $\mathrm{add}(a) \subseteq \mathcal{F}$, delete effect $\mathrm{del}(a) \subseteq \mathcal{F}$ and cost $\mathrm{cost}(a) \in \mathbb{Q}_0^+$.[1] We denote by $\mathcal{S} := 2^{\mathcal{F}}$ the set of all states. An action $a \in \mathcal{A}$ is **applicable** in $s \in \mathcal{S}$ if $\mathrm{pre}(a) \subseteq s$, and we use $\mathcal{A}(s) \subseteq \mathcal{A}$ to denote the set of all actions applicable in s. The **resulting state** of applying an applicable action a in a state s is the state $a[\![s]\!] = (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$. A state s is called a **goal state** if $G \subseteq s$. A sequence of actions $\pi = (a_1, \ldots, a_n)$ is applicable in a state s_0 if there are states $s_1, \ldots, s_n$ such that a_i is applicable in s_{i-1}, i.e., $a_i \in \mathcal{A}(s_{i-1})$, and $s_i = a_i[\![s_{i-1}]\!]$ for $i \in \{1, \ldots, n\}$. The resulting state of this application is $\pi[\![s_0]\!] = s_n$ and $\mathrm{cost}(\pi) = \sum_{i=1}^{n} \mathrm{cost}(a_i)$ denotes the cost of this sequence of actions.

A sequence of actions π is called an **s-plan** if it is applicable in the state s and $\pi[\![s]\!]$ is a goal state. An I-plan is called simply a **plan**. An s-plan (plan) π is called **optimal** if its cost is minimal among all s-plans (plans). A state s is called **solvable** if there exists an s-plan.

We assume readers are familiar with the concept of heuristic search and with the A* algorithm in particular. We provide pseudo-code for A* as part of our pseudo-code for AOCA* in Section 5. In what follows, we introduce the basic terminology for heuristic search and A*.

[1] Using rational instead of real numbers for action costs plays a role for our complexity result in Section 3.

A **heuristic** $h \colon \mathcal{S} \mapsto \mathbb{Q}_0^+ \cup \{\infty\}$ is a function estimating the cost of optimal s-plans where ∞ indicates that s is not solvable. The **perfect heuristic**, denoted by $h^\star$, is a heuristic mapping each state s to the cost of the optimal s-plan or to ∞ if s is not solvable. A heuristic h is called **admissible** if $h(s) \leq h^\star(s)$ for all states s, and h is called **consistent** if $h(s) \leq h(a[\![s]\!]) + \mathrm{cost}(a)$ for all states s and $a \in \mathcal{A}(s)$. For a state s during search, we use the **g-value** $g(s)$ to denote the cost of the current path to s, we call the heuristic value $h(s)$ the **h-value**, and we define the **f-value** as $f(s) = g(s) + h(s)$.

In a nutshell, A* maintains a priority queue of states, called open list, ordered by the f-values. In each iteration, a state s with the lowest f-value is removed from the open list (if the open list is empty, there exists no plan), and moved to the closed list. If s is a goal state, an optimal plan is found. Otherwise, s is expanded and its non-closed successor states s' are inserted into the open list; if they already are in the open list, their g-values are updated so that they are minimal. For inconsistent heuristics h, the g-value can still decrease for closed states, and hence we need to allow re-opening, inserting closed s' into the open list in case the new path to s' is cheaper.

3 Action Optimality Checking

We focus on the AOC problem, deciding whether a given action a is optimal in a given state s. We impose several restrictions. First, if a is not applicable in s, the question becomes trivial and uninteresting. Second, the same is true if s is a goal state. Third, allowing s to be unsolvable would merely overload the AOC problem with the problem of deciding whether s is solvable—after all, if s is unsolvable then it does not matter which action is applied and, again, AOC becomes trivial. Therefore, we restrict to solvable non-goal states s and actions $a \in \mathcal{A}(s)$ applicable in s.

Definition 1. *Given a solvable non-goal state s and an action $a \in \mathcal{A}(s)$ applicable in s, we say that a **is optimal in** s if there exists an optimal s-plan $\pi = (a_1, \ldots, a_n)$ such that $a_1 = a$.*

Before we get to the discussion of techniques that can be used for determining whether an action is optimal in a state s, we show that this decision problem is PSPACE-complete, and hence as hard as finding an optimal s-plan. Remarkably, this is so even when we already know that an s-plan exists.

Definition 2. ***Action-Optimality-Check (AOC) decision problem**: Given a planning task P, a solvable non-goal state s and an action $a \in \mathcal{A}(s)$ applicable in s, is a optimal in s?*

Theorem 1. *Action-Optimality-Check is* PSPACE*-complete.*

Proof. Hardness: Let Plan-Sat denote the decision problem deciding whether a planning task P has a plan, which is a PSPACE-complete problem (Bylander 1994). To show that AOC is PSPACE-hard, we reduce Plan-Sat to AOC as follows. Given a planning task $P = (\mathcal{F}, \mathcal{A}, I, G)$, we construct another planning task $P' = (\mathcal{F} \cup \{x\}, \mathcal{A} \cup \{a, a'\}, I, G)$ where $x \notin \mathcal{F}$ is a fresh fact, and $a, a' \notin \mathcal{A}$ are two fresh actions such that $\mathrm{pre}(a) = \mathrm{pre}(a') = \mathrm{del}(a) =$

$\mathrm{del}(a') = \{x\}$, $\mathrm{add}(a) = I$, $\mathrm{add}(a') = G$, $\mathrm{cost}(a) = 0$, and $\mathrm{cost}(a') = 2^{|\mathcal{F}|} \cdot \max_{b \in \mathcal{A}} \mathrm{cost}(b) + 1$. Consider AOC for $s := \{x\}$ and a in P'. First, s is a solvable non-goal state in P', and a is applicable in s, so the problem instance is valid. Second, P has a plan if and only if a is optimal in s, because $\pi = (a_1, \ldots, a_n)$ is an optimal plan for P if and only if $\pi' = (a, a_1, \ldots, a_n)$ is an optimal s-plan in P', and the cost of both π and π' is strictly smaller than $\mathrm{cost}(a')$.

Membership: Consider the following procedure. As action costs are rational, we can denote $x_i/y_i = \mathrm{cost}(b_i)$ for $b_i \in \mathcal{A}$, where $y_i \neq 0$. Let $k := 1/(\Pi_{b_i \in \mathcal{A}} y_i)$. Then $\mathrm{cost}(b_i)/k$ is an integer for all $b_i \in \mathcal{A}$. We iterate over $K \in \{0, k, 2k, \ldots\}$ and for each K we decide (P1) whether there exists an s-plan of cost $\leq K$ when enforcing the first action to be a, and (P2) whether there exists an s-plan of cost $\leq K$ when enforcing the first action to be $\neq a$. We terminate when the answer for either (P1) or (P2) is "Yes", at which point K is the cost of an optimal s-plan. We return the (P1) answer, which clearly is "Yes" if and only if a is optimal in s. Hence the procedure decides AOC. It remains to prove that we can implement the procedure in polynomial space. This is the case for (P1) and (P2) as deciding whether a planning task has a plan of cost $\leq K$ is in PSPACE (Bylander 1994; Aghighi and Bäckström 2015). We can also compute and represent k in polynomial space as the number of digits required in a non-unary representation of $\Pi_{b_i \in \mathcal{A}} y_i$ grows linearly with that of the factors y_i. This concludes the proof. □

Typically, irrational action costs are allowed in the literature, i.e., costs are not in $\mathbb{Q}_0^+$ but in $\mathbb{R}_0^+$. Hardness of AOC obviously still holds then. For membership, we leave this question open (arguably, it has limited practical relevance).

4 Multiple Searches and Early Termination

We henceforth consider an AOC instance s, a, and we denote $s' = a[\![s]\!]$. A straightforward way to decide AOC is to determine the costs $h^\star(s)$ and $h^\star(s')$ of the optimal s-plan and s'-plan: a is optimal in s if and only if $h^\star(s) = h^\star(s') + \mathrm{cost}(a)$. There are many ways to determine $h^\star(s)$ and $h^\star(s')$. Here, we focus on the use of A*, which means to simply run A* twice, once on s and once on s'. However, there are several ways to improve this vanilla algorithm.

A first observation is that, in some special cases, one of the two searches is enough to provide the desired answer. If A* from s generates a plan starting with a, then a is optimal in s. If A* from s' finds that s' is unsolvable, then a does not start any s-plan.

A more interesting method facilitating early termination is to compute upper bounds $u(s)$ on the cost of the optimal s-plan (i.e., $u(s) \geq h^\star(s)$), as well as lower bounds $l(s')$ on the cost of the optimal s'-plan (i.e., $l(s') \leq h^\star(s')$). If, at some point during the computation, $u(s) < l(s') + \mathrm{cost}(a)$, then a is *not* optimal in s and we can terminate. A straightforward way to leverage this observation is to run A* from s first, obtaining $u(s) = h^\star(s)$ which can then be used for early termination in A* from s', where we obtain the lower bound $l(s')$ as the maximal f-value of any expanded node. As a more advanced option, akin to finding initial upper bounds in branch-and-bound approaches, we can also use a non-optimal (satisficing) planner to quickly find an upper bound $u(s)$ for use in A* from s'.

All algorithms in this design space run several searches, and early termination depends on the order in which we run those. Rather than fixing such a sequential order however, we can also leverage parallelization. We can run A* from s and from s' in parallel, terminating early if we can do so based on either of the two searches individually. If A* from s finishes first, we can use $u(s)$ for early termination of the still running A* from s' as described above. Lastly, we can add one or several satisficing planners to the pool of parallel searches, to more quickly determine upper bounds $u(s)$ that can be communicated to the A* search from s' for possible early termination.

5 Single Search Using AOCA*

So far, we discussed how to decide AOC using multiple searches. Here, we show that we can solve this problem by a single search extending A*. We baptize this algorithm AOCA* (Action-Optimality-Check A*). In what follows, we introduce and explain the algorithm, prove its correctness, and discuss the issue of inconsistent heuristics and reopening.

Algorithm

As before, let s be a state, $a \in \mathcal{A}(s)$ and $s' = a[\![s]\!]$. Algorithm 1 shows pseudo-code for AOCA*. The highlighted parts (in blue) show the differences to A*.

Note that in our pseudo-code we do not use a closed list explicitly. Instead, we use the convention that $g(t) = \infty$ for all states t unless $g(t)$ was explicitly set (on line 19), i.e., a state t is closed if $g(t) \neq \infty$ and t is not in the open list. This also means that lines 16 to 21 perform all of: (a) insertion of new states in the open list; (b) update of states in the open list when their g-value is decreased; and (c) re-opening of closed states (in case of inconsistent heuristic h) when we find lower g-values for already closed states.

In contrast to A*, the AOCA* algorithm maintains an additional *tag* $\top$ or $\bot$ associated with each state t (initialized on line 3). If a state t is tagged with $\top$ ($\mathrm{tag}(t) = \top$), then the cheapest path from s to t found so far starts with the action a. The tag $\bot$ indicates that the cheapest path to t found so far does not start with a. AOCA* then does not order the states in the open list just by their f-values, but in case two states have the same f-value it prefers to pop first the state with the tag $\top$. That is, AOCA* still searches for an optimal s-plan (as A*), but it prefers optimal s-plans starting with the action a. Therefore, the first goal state t popped from the open list (i.e., the one with the lowest f-value and therefore also g-value) is tagged with $\top$ if and only if a starts an optimal s-plan.

The correct tagging of states is ensured as follows. The first state tagged with $\top$ is s' because if a starts an optimal s-plan, then $s' = a[\![s]\!]$ is the first state visited by that s-plan. Then tags are inherited from the cheapest predecessor states while preferring predecessors tagged with $\top$. That is, given a state t, applicable action $b \in \mathcal{A}(t)$ and the successor state $t' = b[\![t]\!]$, we set $g(t') = g(t) + \mathrm{cost}(b)$ (line 19)

Algorithm 1: Pseudo-code of AOCA*

Input: A planning task P with actions $\mathcal{A}$, a solvable non-goal state s, an action $a \in \mathcal{A}(s)$, and an admissible heuristic h

Output: $\top$ if a is optimal in s, otherwise $\bot$

1 $s' \leftarrow a[\![s]\!]$;
2 $g \leftarrow \{(s, 0), (s', \mathrm{cost}(a))\}$; // Mapping of states to g-values.
3 $\mathrm{tag} \leftarrow \{(s, \bot), (s', \top)\}$; // Mapping of states to $\top/\bot$ tags.
/* Open list of triplets of the form (state, f-value, tag). */
4 $\mathrm{open} \leftarrow \{(s, h(s), \mathrm{tag}(s)), (s', g(s') + h(s'), \mathrm{tag}(s'))\}$;
/* From now on, we assume $g(t) = \infty$ if $(t, \_) \notin g$. */
5 **while** open $\neq \emptyset$ **do**
/* Early termination criteria: */
6 **if** $\mathrm{tag}(t) = \top$ for all $(t, f(t), \mathrm{tag}(t)) \in \mathrm{open}$ **then**
7 **return** $\top$;
8 **if** $\mathrm{tag}(t) = \bot$ for all $(t, f(t), \mathrm{tag}(t)) \in \mathrm{open}$ **then**
9 **return** $\bot$;
/* Extract a node with the minimal f-value, breaking ties in favor of states t such that $\mathrm{tag}(t) = \top$. */
10 $(t, f(t), \mathrm{tag}(t)) \leftarrow \mathrm{popMin}(\mathrm{open})$;
11 **if** t is a goal state **then**
12 **return** $\mathrm{tag}(t)$
13 **for** $b \in \mathcal{A}(t)$ **do**
14 $t' \leftarrow b[\![t]\!]$;
15 $g_{\mathrm{tmp}} \leftarrow g(t) + \mathrm{cost}(b)$;
/* Recall that $g(t') = \infty$ if $g(t')$ was not set yet. */
16 **if** $g_{\mathrm{tmp}} < g(t')$
17 or ($g_{\mathrm{tmp}} = g(t')$ and $\mathrm{tag}(t) = \top \neq \mathrm{tag}(t')$) **then**
18 $\mathrm{open} \leftarrow \mathrm{open} \setminus \{(t', \_, \_)\}$;
19 $g(t') \leftarrow g_{\mathrm{tmp}}$;
20 $\mathrm{tag}(t') \leftarrow \mathrm{tag}(t)$;
21 $\mathrm{open} \leftarrow \mathrm{open} \cup \{(t', g(t') + h(t'), \mathrm{tag}(t'))\}$;

and consequently $\mathrm{tag}(t') = \mathrm{tag}(t)$ (line 20) whenever either (a) t' is generated for the first time as a successor of t, or (b) we found a cheaper path to t' via t (i.e., $g(t) + \mathrm{cost}(b)$ is strictly smaller than the current value of $g(t')$), or (c) the path to t' via t has the same cost (i.e., $g(t) + \mathrm{cost}(b)$ is equal to the current value of $g(t')$), but t is tagged with $\top$ and t' is tagged with $\bot$ (see condition on line 17). Cases (a) and (b) ensure that the tag is carried via the cheapest paths, and the case (c) ensures that the tag $\top$ has precedence over the tag $\bot$. Note that the case (c) can cause additional re-opening of states that would not happen in the A* algorithm. Consider the example in Figure 1 where the state s_3 is inserted into the open list for the second time with the same g-value when expanded from s' to ensure that the tag $\top$ is correctly assigned.

Moreover, since we are not interested in the resulting s-plan, we can use an additional early termination criteria (lines 6 to 9): If all states in the open list are tagged with $\top$ (lines 6 and 7), then we can safely conclude that also all consecutively added states will be tagged with $\top$. Therefore, since the start state s is solvable, we can terminate early with the output $\top$. Analogously, if all states in the open list are tagged with $\bot$ (lines 8 and 9), we can immediately return $\bot$.

Correctness

Theorem 2. *Let P be a planning task with actions $\mathcal{A}$, let s be a solvable non-goal state, let $a \in \mathcal{A}(s)$, and let h be an admissible heuristic. Then Algorithm 1 with inputs P, s, a and h terminates with the output $\top$ if and only if a is optimal in s.*

Proof. Observe that the early termination criteria (lines 6 to 9) do not influence the output of Algorithm 1. If $\mathrm{tag}(t) = \top$ ($\mathrm{tag}(t) = \bot$) for all states t in the open list, then $\mathrm{tag}(t') = \top$ ($\mathrm{tag}(t') = \bot$) for all states t' that will be added to the open list from that point on (see line 19). Therefore, if $\mathrm{tag}(t) = \top$ ($\mathrm{tag}(t) = \bot$) for all states t in the open list, then the output will eventually be $\top$ ($\bot$). So, from now on, we consider Algorithm 1 without lines 6 to 9.

Given $(f, t), (f', t') \in \mathbb{Q}_0^+ \times \{\top, \bot\}$, we say that $(f, t) < (f', t')$ if $f < f'$, or $f = f'$ and $t = \top$ and $t' = \bot$. Analogously, we say that $(f, t) \leq (f', t')$ if $(f, t) < (f', t')$ or $(f, t) = (f', t')$. In other words, we order pairs of numbers and tags element-wise so that $\top$ is ordered before $\bot$.

Now, we have that:

(i) In every cycle, we pop (on line 10) the state with the minimum $(f(t), \mathrm{tag}(t))$ pair, i.e., we pop a state t such that $(f(t), \mathrm{tag}(t)) \leq (f(t'), \mathrm{tag}(t'))$ for all $(t', f(t'), \mathrm{tag}(t')) \in \mathrm{open}$.

(ii) The condition on lines 16 and 17 is satisfied for $t' = b[\![t]\!]$ if and only if $(g(t) + \mathrm{cost}(b), \mathrm{tag}(t)) < (g(t'), \mathrm{tag}(t'))$ with the convention that $g(t') = \infty$ and $\mathrm{tag}(t') = \bot$ if g and tag is not defined for t'. Therefore, $g(t')$ and $\mathrm{tag}(t')$ is set to $g(t) + \mathrm{cost}(b)$ and $\mathrm{tag}(t)$, respectively (on lines 19 and 20), when $(g(t) + \mathrm{cost}(b), \mathrm{tag}(t)) < (g(t'), \mathrm{tag}(t'))$ holds.

(iii) If there exists an optimal s-plan starting with a, then $s' = a[\![s]\!]$ must be an intermediate state of that s-plan. The state s' is tagged with $\top$ (line 3) and added to the open list with the f-value $g(s') + h(s') = \mathrm{cost}(a) + h(s')$ (line 4). Therefore, if a starts an optimal s-plan, s' remains tagged with $\top$ (as it follows from (ii) that the tag cannot be changed).

So, it follows from (i), (ii) and (iii) that AOCA* is A* where f-values (g-values) are replaced with pairs of f-values and tags (g-values and tags). Therefore, a state t is tagged with $\top$ if and only if the cheapest path from s to t found so far goes through s'. Since the first goal state t popped from the open list (line 10) has the minimum $(g(t), \mathrm{tag}(t))$ among all goal states, such goal state t is tagged with $\top$ if and only if there exists an optimal s-plan that goes through s' (and therefore starts with the action a). Therefore, AOCA* outputs (line 12) $\top$ if and only if a is optimal in s. □

Inconsistent Heuristics and Re-Opening

Like for A*, re-opening is needed in AOCA* in case the heuristic is inconsistent. However, the modified re-opening condition (line 17) can lead to additional state re-openings. Figure 1 gives an example. AOCA* pops states from the open list (line 10) in the order s, s_2, s_3, s', s_3, s_4. In particular, s_3 is only re-opened when expanding s' because, on the new path found to s_3, it is tagged (line 17). The algorithm correctly determines that a is optimal in s.

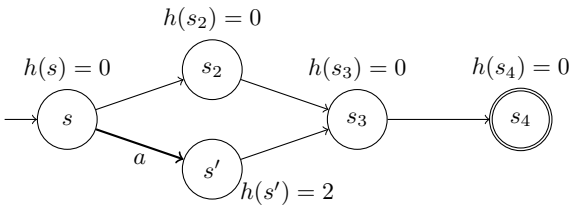

Figure 1: Example state space with an inconsistent heuristic h. Action costs are $\mathrm{cost}(a) = 1$ for all a.

Such additional re-openings can only occur if the heuristic is inconsistent. With a consistent heuristic, as in A*, re-opening never happens at all. In particular, condition on line 17 is never triggered. The reason is that when there are multiple states with the same (lowest) f-value, the states tagged with $\top$ are popped from the queue before states tagged with $\bot$. In other words, where A* with a consistent heuristic guarantees to pop states in non-decreasing order of their f-values, AOCA* with a consistent heuristic guarantees to pop states in non-decreasing order of their f-values *and tags*, with $\top$ ordered before $\bot$.

6 Criteria Based on Dominance Functions

As mentioned in the introduction, prior work has already come up with a sufficient criterion for action optimality, which from the viewpoint of our work here is a sufficient criterion for AOC. Specifically, Torralba (2017) introduces dominance functions $\mathcal{D}\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R} \cup \{-\infty\}$ such that $\mathcal{D}(x, y) \leq h^\star(x) - h^\star(y)$ for all $x, y \in \mathcal{S}$ with $h^\star(x) < \infty$. Intuitively, "state y is better than state x by at least $\mathcal{D}(x, y)$", where "better" means smaller $h^\star$-value.[2]

As Torralba observed, if $c(a) \leq \mathcal{D}(s, s')$ (i.e., s' is better than s by at least $c(a)$), then $c(a) \leq h^\star(s) - h^\star(s')$ and therefore a is optimal in s.

In our context, we can use three more criteria. We can swap the roles of s and s' to obtain an equally simple criterion for showing that a is sub-optimal: If $-\mathcal{D}(s', s) < c(a)$ (i.e., s is worse than s' by less than $c(a)$, or s is better than s'), then a is not optimal in s. Moreover, we can consider all other actions applicable in s: If, for every action $a' \in \mathcal{A}(s)$ such that $a' \neq a$, $c(a) - c(a') \leq \mathcal{D}(a'[\![s]\!], s')$ (i.e., s' is better than $a'[\![s]\!]$ by at least the difference in action costs), then a is optimal in s. Conversely, if there exists $a' \in \mathcal{A}(s)$ such that $a' \neq a$ and $c(a') - c(a) < \mathcal{D}(s', a'[\![s]\!])$ (i.e., $a'[\![s]\!]$ is better than s' by more than the difference in action costs), then a is sub-optimal in s.

Proposition 3. *Let $\mathcal{D}$ be a dominance function, let s be a solvable non-goal state, and let $a \in \mathcal{A}(s)$ be an action applicable in s with resulting state $s' = a[\![s]\!]$. Then the following claims hold:*

(a) If $c(a) \leq \mathcal{D}(s, s')$, then a is optimal in s.

(b) If $-\mathcal{D}(s', s) < c(a)$, then a is sub-optimal in s.

(c) If $c(a) - c(a') \leq \mathcal{D}(a'[\![s]\!], s')$ for all $a' \in \mathcal{A}(s) \setminus \{a\}$, then a is optimal in s.

(d) If $c(a') - c(a) < \mathcal{D}(s', a'[\![s]\!])$ for any $a' \in \mathcal{A}(s) \setminus \{a\}$, then a is sub-optimal in s.

Proof. We prove each claim individually:

(a) We have $h^\star(s') + c(a) \leq h^\star(s') + \mathcal{D}(s, s') \leq h^\star(s)$, therefore a is optimal in s.

(b) If s' is not solvable, a is not optimal in s (because s is solvable). Otherwise, if s' is solvable and $-\mathcal{D}(s', s) < c(a)$, then $h^\star(s) \leq h^\star(s') - \mathcal{D}(s', s) < h^\star(s') + c(a)$, therefore a is not optimal.

(c) We show that $c(a) + h^\star(s') \leq c(a') + h^\star(a'[\![s]\!])$ for all $a' \in \mathcal{A}(s) \setminus \{a\}$. If $a'[\![s]\!]$ is not solvable, then $c(a) + h^\star(s') \leq c(a') + h^\star(a'[\![s]\!]) = \infty$. If $a'[\![s]\!]$ is solvable and $c(a) - c(a') \leq \mathcal{D}(a'[\![s]\!], s')$, then $c(a) - c(a') \leq h^\star(a'[\![s]\!]) - h^\star(s')$. Since $c(a)$, $c(a')$, and $h^\star(a'[\![s]\!])$ are finite, we have that $c(a) + h^\star(s') \leq c(a') + h^\star(a'[\![s]\!])$.

(d) If s' is not solvable, a is not optimal (since s is solvable). If s' is solvable and $c(a') - c(a) < \mathcal{D}(s', a'[\![s]\!])$, then $c(a') - c(a) < h^\star(s') - h^\star(a'[\![s]\!])$. Since $c(a)$, $c(a')$, and $h^\star(s')$ are finite, we have that $c(a') + h^\star(a'[\![s]\!]) < c(a) + h^\star(s')$. Therefore a is not optimal in s. □

We have implemented Torralba's criterion (a) together with our new three criteria (b), (c) and (d), i.e., we always check all four criteria. We henceforth denote this algorithm with $\mathcal{D}$.

7 Experiments

We implemented our algorithms in the code-base of Fast Downward (Helmert 2006). Source code and benchmarks are publicly available (Eisenhut et al. 2025). We conducted broad experiments. In what follows, we describe our benchmark design, and the search algorithms we compare. We then discuss our results in terms of coverage and runtime, and we provide a comparison of the most competitive algorithm (AOCA*) to A* to gauge the practical effort for AOC compared to the standard optimal planning problem.

Benchmarks

We evaluate our methods on all planning domains from the optimal tracks of the International Planning Competitions (IPCs) from 1998 to 2023. We exclude domains containing conditional effects after grounding with cpddl.[3] We merged, for each domain, all benchmark suites across different IPCs, eliminating duplicate instances. We removed benchmark tasks with only a single applicable action in the initial state (as this case is uninteresting). For each of the remaining 1589 planning tasks, we designed two AOC problem instances, each using the task's initial state as the state s, and randomly selecting two actions a applicable in s. We hence have a total of 3178 AOC problem instances.

[2] Note that $\mathcal{D}(x, y)$ can also be negative: if $-\infty < \mathcal{D}(x, y) < 0$, $h^\star(y) \leq h^\star(x) - \mathcal{D}(s, t)$ means that y could be worse than x, but only by at most $-\mathcal{D}(x, y)$.

[3] https://gitlab.com/danfis/cpddl

		LM-cut						Cartesian						Merge & Shrink					
	$\mathcal{D}$	*SMS*	*SMS^{REV}*	*SMS^{BC}*	*PMS^{BC}*	*PMS^{LAMA}*	*AOCA**	*SMS*	*SMS^{REV}*	*SMS^{BC}*	*PMS^{BC}*	*PMS^{LAMA}*	*AOCA**	*SMS*	*SMS^{REV}*	*SMS^{BC}*	*PMS^{BC}*	*PMS^{LAMA}*	*AOCA**
barman (68)	0	8	8	8	8	10	**16**	22	22	22	22	22	**27**	22	22	22	22	22	**27**
cavediving (40)	0	14	30	14	30	30	**32**	14	**23**	14	**23**	**23**	15	14	22	14	22	22	**24**
elevators (100)	1	78	78	80	80	80	**81**	76	76	79	82	83	**85**	66	66	66	67	67	**70**
freecell (160)	0	**30**	**30**	**30**	**30**	**30**	**30**	52	52	51	54	54	**57**	**42**	**42**	**42**	**42**	**42**	**42**
ged (40)	0	**30**	**30**	**30**	**30**	**30**	**30**	33	33	**38**	**38**	**38**	32	33	33	**38**	**38**	**38**	**38**
labyrinth (40)	0	3	10	2	10	**11**	10	3	10	4	**11**	**11**	**11**	7	**10**	8	**10**	**10**	9
parking (80)	0	11	10	12	**14**	**14**	13	6	6	6	7	8	**12**	12	12	**14**	**14**	**14**	**14**
p-net-align (40)	0	17	17	18	18	18	**20**	3	6	4	**7**	**7**	4	12	10	13	15	15	**16**
rico-robots (40)	0	6	6	8	8	8	**20**	16	15	17	21	23	**30**	2	2	2	**4**	**4**	2
scanalyzer(100)	8	46	45	57	**62**	**62**	60	**42**	**42**	**42**	**42**	**42**	**42**	46	46	46	**50**	**50**	48
snake (30)	0	10	10	11	10	10	**12**	18	17	18	18	18	**21**	13	10	13	16	16	**17**
termes (40)	0	8	8	8	12	12	**40**	24	22	23	24	24	**40**	26	26	26	26	28	**40**
tidybot (80)	2	**60**	**60**	**60**	**60**	**60**	**60**	65	64	**66**	**66**	**66**	65	58	57	56	62	**64**	62
others (2320)	303	1186	1186	1197	1222	**1227**	1216	1185	1184	1192	1219	**1224**	1207	1188	1187	1196	1214	**1230**	1207
total (3178)	314	1507	1528	1535	1594	1602	**1640**	1559	1572	1576	1634	1643	**1648**	1541	1545	1556	1602	**1622**	1616

Table 1: Number of solved AOC tasks. We show all domains where there is a difference of at least 6 for at least one heuristic (excluding $\mathcal{D}$).

Compared Algorithms

Our implementation of AOCA* is directly based on Fast Downward's A* implementation and closely follows the pseudo-code shown in Algorithm 1. In particular, we do not use separate search queues for tagged and untagged states, instead all states are stored in the same queue and the tag is stored for each state. The early termination criterion is efficiently implemented using two counters, for both the number of tagged and untagged states in the open list. Ties of f-value and tags are broken by lower h-values, which coincides with the way f-value ties are broken in A*.

We compare AOCA* to the following multi-search algorithms sketched in Section 4:

- SMS: sequential multi-search, which runs A* first from s and then from s', omitting A* from s' if a plan starting with a is found with A* from s,
- SMS$^{\text{REV}}$: variant of SMS, which runs A* first from s' and then from s, omitting A* from s if s' is found to be unsolvable,
- SMS$^{\text{BC}}$: SMS with bounds check, which runs SMS with early termination based on $u(s)$ and $l(s')$,
- PMS$^{\text{BC}}$: parallel multi-search, which runs the two A* searches in parallel (each in its own thread) and with all early termination optimizations enabled,
- PMS$^{\text{LAMA}}$: PMS$^{\text{BC}}$ extended with an additional thread running the anytime satisficing planner LAMA (Richter and Westphal 2010) from s, gradually improving $u(s)$.

We perform this comparison for three different state-of-the-art heuristic functions, namely LM-cut (Helmert and Domshlak 2009), Cartesian abstractions (Seipp and Helmert 2018), and merge-and-shrink (Helmert et al. 2014). The heuristics are based on the recommendations made in Fast Downward as of Release 23.06. In particular, in merge-and-shrink, we limit the transition system size to 50 000 states, and for the additive Cartesian heuristic, we use a transitions limit of one million. To ensure that the exact same heuristic is constructed across separate invocations of Fast Downward – and hence to enable the direct comparison of performance across search algorithms – we disabled time limits as well as an optimization allowing the Cartesian heuristic to recover from out-of-memory cases.

The experiments were run on a cluster with AMD EPYC 7702 processors. The time and memory limits were set to 30 minutes and 8 GB for the sequential search variants, i.e., AOCA*, SMS, SMS$^{\text{REV}}$ and SMS$^{\text{BC}}$. For the parallel variants, i.e., PMS$^{\text{BC}}$ and PMS$^{\text{LAMA}}$, we used the same time limit for each parallel thread, and increased the shared memory to 16 and 24 GB of memory, respectively. Hence, summed up over the parallel threads, PMS$^{\text{BC}}$ (PMS$^{\text{LAMA}}$) enjoy twice (thrice) the resources as the sequential variants. Our main motivation for this is the comparison to AOCA*, which as we shall see is superior to PMS$^{\text{BC}}$ and PMS$^{\text{LAMA}}$ despite their generous allowances.

We also compare to the method utilizing a dominance function ($\mathcal{D}$) described in Section 6. To compute the dominance function (which is done once per planning task) we used the same time and memory limits (30 minutes and 8 GB) as for search. The time spent in the evaluation of $\mathcal{D}$ was negligible.

Results: Number of Solved Tasks

Table 1 summarizes the number of solved AOC tasks (coverage), i.e., the number of tasks where the corresponding methods proved either that the tested action is optimal or that it is not optimal. The upper part of Table 2 shows the the number of domains in which one method solved strictly more tasks than the other. The lower part of Table 2 shows the number of tasks solved by one method but not the other. That is, Table 2 shows domain and task dominance between

	LM-cut							Cartesian							Merge & Shrink						
	$\mathcal{D}$	SMS	SMS^{REV}	SMS^{BC}	PMS^{BC}	PMS^{LAMA}	$AOCA^\star$	$\mathcal{D}$	SMS	SMS^{REV}	SMS^{BC}	PMS^{BC}	PMS^{LAMA}	$AOCA^\star$	$\mathcal{D}$	SMS	SMS^{REV}	SMS^{BC}	PMS^{BC}	PMS^{LAMA}	$AOCA^\star$
domain dominance																					
$\mathcal{D}$		**46**	**46**	**46**	**46**	**46**	**46**		**46**	**46**	**46**	**47**	**47**	**46**		**47**	**47**	**47**	**47**	**47**	**47**
SMS	3		8	**14**	**27**	**29**	**30**	3		7	**12**	**27**	**28**	**25**	3		3	**11**	**26**	**30**	**23**
SMS$^{\text{REV}}$	3	**10**		**18**	**21**	**25**	**25**	3	**10**		**18**	**27**	**28**	**24**	3	**6**		**11**	**23**	**27**	**22**
SMS$^{\text{BC}}$	3	2	5		**19**	**23**	**23**	3	3	8		**22**	**24**	**21**	3	3	3		**17**	**22**	**14**
PMS$^{\text{BC}}$	3	0	0	1		**7**	8	3	0	0	0		**7**	14	3	0	0	0		**9**	7
PMS$^{\text{LAMA}}$	3	0	0	1	1		9	3	0	0	0	1		14	3	0	0	0	0		7
AOCA*	3	0	0	0	8	**13**		3	4	8	6	**16**	**18**		3	0	1	1	**9**	**15**	
task dominance																					
$\mathcal{D}$		**1263**	**1284**	**1290**	**1347**	**1355**	**1396**		**1317**	**1330**	**1334**	**1390**	**1399**	**1407**		**1299**	**1303**	**1314**	**1360**	**1380**	**1374**
SMS	70		**35**	**32**	**88**	**95**	**136**	72		**27**	**22**	**76**	**84**	**97**	72		**17**	**21**	**61**	**81**	**77**
SMS$^{\text{REV}}$	70	14		**39**	**66**	**74**	**112**	72	14		**32**	**62**	**71**	**100**	72	13		**26**	**57**	**77**	**73**
SMS$^{\text{BC}}$	69	4	32		**61**	**68**	**110**	72	5	28		**58**	**67**	**85**	72	6	15		**46**	**66**	**63**
PMS$^{\text{BC}}$	67	1	0	2		**10**	**58**	70	1	0	0		**10**	**58**	72	0	0	0		**20**	**31**
PMS$^{\text{LAMA}}$	67	0	0	1	2		**58**	70	0	0	0	1		**54**	72	0	0	0	0		29
AOCA*	70	3	0	5	12	20		73	8	24	13	44	49		72	2	2	3	17	**35**	

Table 2: Domain and task dominance. Upper part: the value in cell (x, y) is the number of domains where the algorithm in column y solves strictly more tasks than the one in row x. Lower part: the value in cell (x, y) is the number of tasks solved by y but not by x.

the methods as well as complementarity between them.

AOCA* has significantly higher coverage than both vanilla baselines running two A* searches. There are only very few domains where using SMS or SMS$^{\text{REV}}$ is beneficial, and very few tasks solved by SMS or SMS$^{\text{REV}}$ but not by AOCA*. Thus, there is almost no complementarity as AOCA* is clearly a better choice: AOCA* dominates SMS or SMS$^{\text{REV}}$ in 22 to 30 domains and it solves from 73 to 136 tasks not solved by SMS or SMS$^{\text{REV}}$ (depending on the heuristic). The early termination criteria (SMS$^{\text{BC}}$) are an improvement over SMS and SMS$^{\text{REV}}$, and are rarely detrimental. AOCA* is still clearly better than SMS$^{\text{BC}}$ though, both in total coverage and per-domain/per-task dominance.

Consider now the parallel-search variant PMS$^{\text{BC}}$. This turns out to be highly complementary to AOCA* in terms of coverage, with a small advantage for AOCA* in total coverage as well as per-task dominance, but an almost even split and small advantages for PMS$^{\text{BC}}$ in terms of per-domain dominance. Recall here that PMS$^{\text{BC}}$ has twice the time and memory resources across threads though; and we will see below that, in the more detailed runtime view on performance, AOCA* is superior.

When the satisficing planner LAMA is used for finding better bounds in addition to two parallel A* searches (PMS$^{\text{LAMA}}$), we can still observe higher overall coverage of AOCA* for two out of three tested heuristics. The difference is, as expected, smaller than between PMS$^{\text{BC}}$ and AOCA* as even more computational resources are used by PMS$^{\text{LAMA}}$. It is, however, still clear that there is a high degree of complementarity between these methods.

Consider finally the dominance-based sufficient criteria $\mathcal{D}$. These are clearly not competitive with the search-based methods in terms of coverage. There are only 70 tasks from three domains (pathways, rovers, and satellite) where $\mathcal{D}$ performs well. Nevertheless, dominance criteria could be useful as they offer very different computational trade-offs than search. The computation of $\mathcal{D}$ can be costly but needs be done only once per task. So in settings where many AOC problem instances need to be answered for the same task, dominance criteria could be used as pre-checks to avoid costly calls to search.

Results: Runtime

Let us now consider runtime, for a more fine-grained view on performance than coverage. Figure 2 compares AOCA* with the multi-search variants SMS$^{\text{BC}}$, PMS$^{\text{BC}}$, and PMS$^{\text{LAMA}}$. We specifically measure *AOC decision time*, the wall-clock time until the decision is made (which in parallel searches means we stop when the first thread has found the decision). We distinguish whether the action a under consideration is optimal or not (which we know for all AOC decision problems solved by at least one of the compared algorithms).

The results show at a glance that, overall, AOCA* outperforms all competitors in terms of AOC decision time. For the best sequential multi-search variant (SMS$^{\text{BC}}$), AOCA* dominates almost completely. The advantage is particularly pronounced in cases where the tested action a was proved to be optimal. This is due to two factors. First, SMS$^{\text{BC}}$ terminates when it finds a plan whereas AOCA* can terminate early if all open states are tagged with the same tag. Second, the tie-breaking of AOCA* in the last f-layer favors plans starting

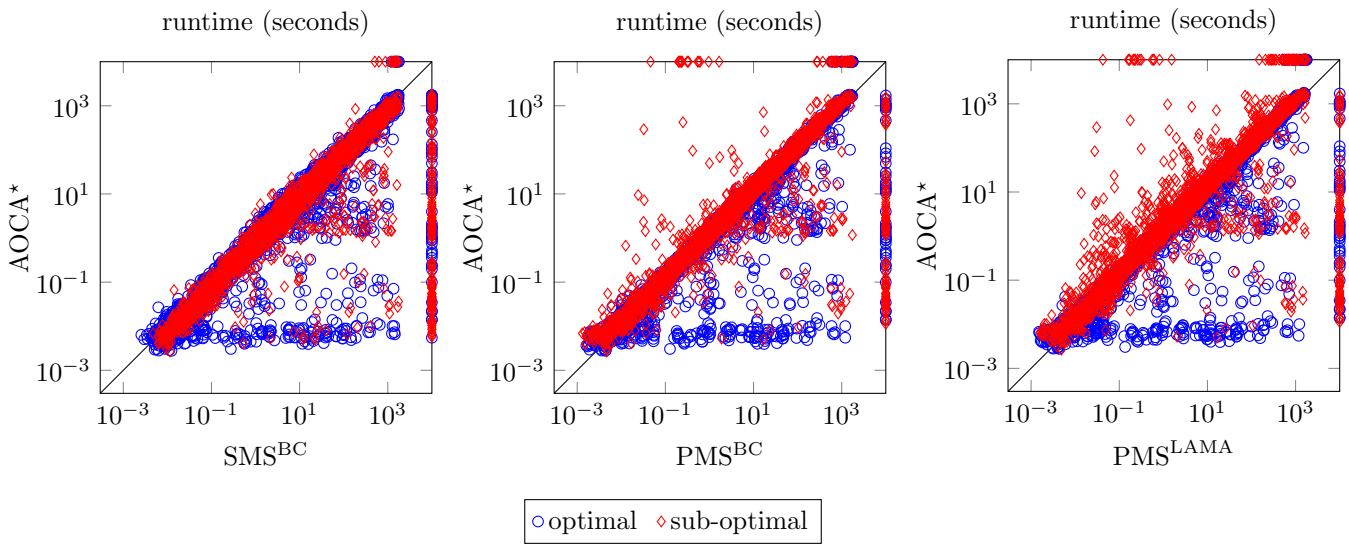

Figure 2: Runtime comparison categorized by AOC result. For each search algorithm, we include the runs for all heuristics.

with a, which can be faster when a is, indeed, optimal. The comparison to PMS$^{\text{BC}}$ and PMS$^{\text{LAMA}}$ shows a similar advantage for AOCA* in the optimal cases. In the non-optimal cases, we see the increased ability of PMS$^{\text{BC}}$, and in particular PMS$^{\text{LAMA}}$ for early termination, so that here performance is more complementary.

Comparison between AOCA* and A*

Let us finally gauge the practical effort for AOC compared to the standard optimal planning problem. To this end, Figure 3 compares AOCA* for s and a against A* from s. The results clearly indicate that AOC is not harder in practice than optimal planning. In many cases, AOCA* is actually faster than A*, which is mainly due to early termination. In fact, search time can be drastically reduced, e.g., for LM-cut alone, there are 106 problem instances that AOCA* solves in less than a tenth of a second while A* takes at least a minute (or fails to solve the problem at all).

The theoretical weakness of additional state re-openings in AOCA* for inconsistent heuristics does not appear to be an issue in practice, i.e., re-openings can only occur when using LM-cut and, in this setting, AOCA* does not perform worse (relative to A*) than with other heuristics.

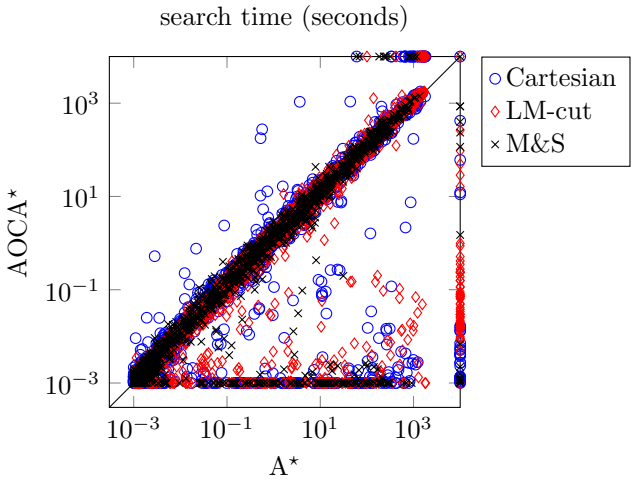

Figure 3: Search time (without heuristic initialization) of AOCA* and A* from s. All times below one millisecond are shown as 1 ms.

8 Conclusion

Action optimality checking (AOC) arguably is relevant in several different application contexts in AI Planning, but has not been addressed yet. Here we begin its investigation, focusing on heuristic search as one state-of-the-art method for planning. We introduce AOCA* which extends A* with maintenance of tags identifying the search sub-tree below the action under scrutiny. We run broad experiments including a range of other search configurations, showing the merits of parallel searches and early termination criteria, and showing AOCA* to be superior overall. AOCA* also compares quite favorably to A*, suggesting that action optimality checking is not harder in practice than optimal planning.

We believe that the further exploration of algorithms for, and applications of, AOC is an important direction for AI Planning. Regarding algorithms, optimal planning algorithms other than heuristic search should be tried, and there presumably is room for further fast under- or over-approximations of AOC. Regarding applications, for example, the use in policy testing challenges scalability, and poses opportunities for additional optimizations (e.g., the re-use of information across AOC instances). There also are interesting extensions of AOC, in particular to bounded suboptimality, or deciding about optimality of plan prefixes instead of single actions.

Acknowledgments

This work was funded by DFG Grant 389792660 as part of TRR 248 (CPEC, https://perspicuous-computing.science).

References

Aghighi, M.; and Bäckström, C. 2015. Cost-Optimal and Net-Benefit Planning - A Parameterised Complexity View. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 1487–1493. AAAI Press.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1–2): 5–33.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1–2): 165–204.

Edelkamp, S. 2001. Planning with Pattern Databases. In *Proceedings of the 6th European Conference on Planning (ECP'01)*, 13–24. Springer-Verlag.

Eisenhut, J.; Fišer, D.; Ruml, W.; and Hoffmann, J. 2025. Code and Data for "Is This a Good Decision? Action Optimality Checking in Classical Planning". Available at http://doi.org/10.5281/zenodo.16744556.

Eisenhut, J.; Schuler, X.; Fišer, D.; Höller, D.; Christakis, M.; and Hoffmann, J. 2024. New Fuzzing Biases for Action Policy Testing. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS'24)*, 162–167. AAAI Press.

Eisenhut, J.; Torralba, Á.; Christakis, M.; and Hoffmann, J. 2023. Automatic Metamorphic Test Oracles for Action-Policy Testing. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS'23)*, 109–117. AAAI Press.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, 408–416. AAAI Press.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, 1007–1012. AAAI Press.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*, 140–149. AAAI Press.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

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

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Krarup, B.; Krivic, S.; Magazzeni, D.; Long, D.; Cashmore, M.; and Smith, D. E. 2021. Contrastive Explanations of Plans through Model Restrictions. *Journal of Artificial Intelligence Research*, 72: 533–612.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based Heuristics for Cost-Optimal Planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 226–234. AAAI Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

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

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 629–637. AAAI Press.

Steinmetz, M.; Fišer, D.; Enişer, H. F.; Ferber, P.; Gros, T.; Heim, P.; Höller, D.; Schuler, X.; Wüstholz, V.; Christakis, M.; and Hoffmann, J. 2022. Debugging a Policy: Automatic Action-Policy Testing in AI Planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 353–361. AAAI Press.

Torralba, Á. 2017. From Qualitative to Quantitative Dominance Pruning for Optimal Planning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, 4426–4432.

Toyer, S.; Thiébaux, S.; Trevizan, F. W.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Wang, R. X.; and Thiébaux, S. 2024. Learning Generalised Policies for Numeric Planning. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS'24)*, 633–642. AAAI Press.