Optimality Certificates for Classical Planning

Esther Mugdan, Remo Christen, Salomé Eriksson

University of Basel, Switzerland
{esther.mugdan,remo.christen,salome.eriksson}@unibas.ch

Abstract

Algorithms are usually shown to be correct on paper, but bugs in their implementations can still lead to incorrect results. In the case of classical planning, it is fortunately straightforward to check whether a computed plan is correct. For optimal planning, however, plans are additionally required to have minimal cost, which is significantly more difficult to verify. While some domain-specific approaches exists, we lack a general tool to verify optimality for arbitrary problems. We bridge this gap and introduce two approaches based on the principle of certifying algorithms, which provide a computer-verifiable certificate of correctness alongside their answer. We show that both approaches are sound and complete, analyze whether they can be generated and verified efficiently, and show how to apply them to concrete planning algorithms. The experimental evaluation shows that verifying optimality comes with a cost, but is still practically feasible. Furthermore, it confirms that the tested planner configurations provide optimal plans on the given instances, as all certificates were verified successfully.

Introduction

Classical planning is concerned with techniques for finding sequences of actions, called plans, that lead from a given initial state to a goal state within a fully-observable, deterministic world model. In *optimal* planning, a task is only considered solved if the found plan has minimal cost among all plans. While optimal planning algorithms are usually theoretically proven to only return optimal plans, their implementation can still contain bugs whose frequency only increases with the level of complexity of the underlying algorithm, and with the level of optimization within the code.

Extensive testing, a common way of detecting bugs, can only ensure correctness on the tested cases. For classical planning, we fortunately can do better: The validity of a found plan can be verified by using tools such as VAL (Howey and Long 2003), INVAL (Haslum 2011), or the formally verified validator by Abdulaziz and Lammich (2018). In the case of unsolvability, we recently introduced ways to verify that a planner reached this conclusion with sound reasoning (e. g. Eriksson, Röger, and Helmert 2018). However, no such tool exists to verify the optimality of a found plan.

The planning community has faced this problem before when running the international planning competition (IPC), which has maintained a dedicated optimal planning track ever since its fourth installment in 2004. Organizers have put great effort towards ensuring optimality by computing minimal-cost plans ahead of time using domain-specific solvers and high resource bounds, or by proving lower bounds on paper. However, these techniques are highly domain-specific and not suitable for a general setting, where we want to verify optimality for so far unseen problems.

A principled, long-known solution is offered by *formal verification* (McCarthy 1963; Floyd 1967; Hoare 1969). Formally verified implementations are guaranteed to always give a correct output, but this usually comes with a prohibitively high overhead. We thus pursue a different approach, called *certifying algorithms* (McConnell et al. 2011). Along with its usual output, a certifying algorithm provides a machine-readable certificate proving the validity of said output. The validity of the triple $\langle$input, output, certificate$\rangle$ can then be checked by an independent verifier. This approach cannot guarantee correctness on every possible input, but still verifies whether every concrete input results in a correct output.

In the SAT community, it is a long-standing practice to verify unsatisfiability with a certificate. The most commonly used proof formalisms are based on reverse asymmetric tautology (Järvisalo, Heule, and Biere 2012), such as DRAT (Heule, Hunter, and Wetzler 2013a,b; Wetzler, Heule, and Hunter 2014) or LRAT (Cruz-Filipe et al. 2017). More recent development in the area uses the general purpose proof system VeriPB (e. g. Elffers et al. 2020; Bogaerts et al. 2022) for turning MaxSAT solvers into certifying algorithms (Vandesande, Wulf, and Bogaerts 2022).

For classical planning, the mentioned tools for verifying plan (non-)existence are based on the concept of certifying algorithms. Our work now applies this concept to verifying optimality of a found plan π. Concretely, we verify that the cost of π is a *lower bound* for plan cost. Together with a verification that π is indeed a plan, we can thus guarantee that π must be an optimal plan. We present two types of certificates for verifying lower bounds. The first reduces optimality to unsolvability, allowing us to make use of existing unsolvability certificates. The second constructs native optimality certificates that directly reason about lower bounds.

In order to be useful in practice, certificates should have certain properties. First and foremost we require *soundness* and *completeness*, i. e., a certificate for a lower bound exists if and only if it is indeed a lower bound. Furthermore, we want to be able to *efficiently generate and verify* certificates, meaning that their creation only imparts polynomial overhead on planner runtime and the verifier runs in polynomial time in the certificate size. Finally, they should be *general* in the sense that a large number of commonly used planning techniques can be modified to be certifying. We theoretically analyze our approaches with respect to these properties, provide empirical studies that support our results, and verify that the tested algorithms indeed produce optimal plans.

Background

We consider planning tasks in the STRIPS formalism (Fikes and Nilsson 1971), where a task is defined as a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ containing a finite set of propositional variables $\mathcal{V}$, a finite set of actions $\mathcal{A}$, an initial state $\mathcal{I}$, and a goal description $\mathcal{G}$. A state $s \subseteq \mathcal{V}$ is defined as the set of variables that are true in s. The set of all states in a planning task Π is therefore the power set of $\mathcal{V}$ and we denote it by $\mathcal{S}$. The goal description $\mathcal{G} \subseteq \mathcal{V}$ implicitly defines the set of goal states as $\mathcal{S}_\mathcal{G} = \{s \in \mathcal{S} \mid \mathcal{G} \subseteq s\}$. An *action* in $\mathcal{A}$ is given by $a = \langle pre(a), add(a), del(a), cost(a) \rangle$ where $pre(a) \subseteq \mathcal{V}$ are the action's *preconditions*, $add(a) \subseteq \mathcal{V}$ its *add effects*, $del(a) \subseteq \mathcal{V}$ its *delete effects*, and $cost(a) \in \mathbb{N}_0$ its cost. We denote the maximal action cost in $\mathcal{A}$ by $c^{\max}(\mathcal{A}) = \max_{a \in \mathcal{A}} cost(a)$, and for $0 \leq i \leq c^{\max}(\mathcal{A})$ we define the set of all actions that have cost i as $A_{\mathcal{A}|i} = \{a \in \mathcal{A} \mid cost(a) = i\}$.

Action a is *applicable* in state s iff $pre(a) \subseteq s$. Applying a in s yields its *successor*, the state $s[\![a]\!] = (s \setminus del(a)) \cup add(a)$. We use the same notation when applying a sequence of action, or *path*, $\pi = \langle a_1, \dots, a_n \rangle$, leading to $s[\![\pi]\!] = s[\![a_1]\!] \dots [\![a_n]\!]$, whereby every action a_i must be applicable in the state $s[\![a_1]\!] \dots [\![a_{i-1}]\!]$ for all $1 \leq i \leq n$. We further define the application of an action a to a set of states S as $S[\![a]\!] = \{s[\![a]\!] \mid s \in S \text{ and } a \text{ applicable in } s\}$. Finally, we also extend application to a set of actions $A \in \mathcal{A}$ and define it as $S[\![A]\!] = \{s \mid a \in A \text{ and } s \in S[\![a]\!]\}$.

A state s' is *reachable* from state s, if there exists a path π such that $s[\![\pi]\!] = s'$. A path π has cost $cost(\pi) = \sum_{a \in \pi} cost(a)$ and is called a *plan* for state s iff $\mathcal{G} \subseteq s[\![\pi]\!]$. A plan for $\mathcal{I}$ is called a plan for Π or simply plan without explicitly referring to a state. A plan for s is *optimal* if there exists no plan for s with lower cost.

Proof Systems A *proof system* is a framework that facilitates the construction of formal proofs by defining how initial knowledge can be acquired and how this knowledge can be expanded. For our purposes, we consider proof systems based on natural deduction (Gentzen 1935), which reason over *inference rules*. Such rules are defined by assumptions and conclusions, with the natural semantic that if all assumptions hold, the conclusions also hold. As an example of such a proof system, we here briefly describe the proof system for verifying unsolvability (Eriksson 2019), since our second approach is heavily based on it.

The system reasons about sets of states and sets of actions. While action sets are simply represented by explicit enumeration, state sets can either be represented in some logic formalism or as a composition of other sets. Possible formalisms are BDDs, Horn formulas, or a DNF formula, whereas possible composition operations for state sets S and S' are the complement $\overline{S}$, the intersection $S \cap S'$, and the union $S \cup S'$. Using an action set A, we can additionally define the progression $S[\![A]\!]$.

The proof system expresses two types of knowledge with these sets: $E \subseteq E'$, where E and E' are either two state sets or two action sets, and S *is dead* for a state set S, which means that no state $s \in S$ can be part of any plan. Knowledge about the deadness of state sets is exclusively derived by using inference rules, which must only be verified once and are thereafter universally true. Knowledge about subset relations on the other hand can either be derived by standard rules of set theory, or by so-called *basic statements*. These differ from inference rules in that their truth is not universal, but depends on the concrete sets used in each application. For our work, only the following two basic statements over state sets S and action sets A are relevant:

B1 $\bigcap_{i=1}^{n} S_i \subseteq \bigcup_{j=1}^{m} S'_j$

B2 $(\bigcap_{i=1}^{n} S_i)[\![A]\!] \cap \bigcap_{j=1}^{m} \overline{S'_j} \subseteq \bigcup_{k=1}^{o} S''_k$.

A proof within such a system can be expressed as a sequence of instantiated basic statements and inference rule applications. This sequence can then be independently verified.

Compilation to Unsolvability

In order to prove that a task has optimal cost $c(\Pi)$, we can make use of unsolvability certificates (e. g. Eriksson, Röger, and Helmert 2018). We do this by reformulating the original task Π such that it has an upper cost bound of $c(\Pi) - 1$, rendering it unsolvable.

Definition 1 (Unsolvability Compilation). *Given a STRIPS planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ and $x \in \mathbb{N}_1$, we define the task $\Pi^x = \langle \mathcal{V}^x, \mathcal{A}^x, \mathcal{I}^x, \mathcal{G} \rangle$. The set of variables for Π^x is defined as $\mathcal{V}^x = \mathcal{V} \cup \{c^i \mid 0 \leq i < x\}$, where variables c^i indicate the cost of reaching that state. The set of actions is $\mathcal{A}^x = \{a^i \mid a \in \mathcal{A}, 0 \leq i < x - cost(a)\}$, where $pre(a^i) = pre(a) \cup \{c^i\}$, $add(a^i) = add(a) \cup \{c^{i+cost(a)}\}$, $del(a^i) = del(a) \cup \{c^i\}$, and $cost(a^i) = cost(a)$. Lastly, we define the initial state $\mathcal{I}^x = \mathcal{I} \cup \{c^0\}$.*

In order to verify that $c(\Pi)$ is the optimal plan cost for Π, we can use an algorithm that certifies unsolvability on $\Pi^{c(\Pi)}$. With the resulting certificate, we can verify that $\Pi^{c(\Pi)}$ is unsolvable. Assuming that the unsolvability certificate is sound and complete, the compilation approach is sound if Π^x being unsolvable implies that x is a lower bound for $c(\Pi)$, and complete if for any $x \leq c(\Pi)$ we have that Π^x is unsolvable.

Theorem 1 (Soundness and completeness). *Π^x is unsolvable iff the optimal plan cost in Π is at least x.*

Proof. We will show both directions in their contraposition. For soundness, this means we need to show that if a plan π for Π with $cost(\pi) < x$ exists, then Π^x is solvable. We will first show through induction over path length that for all paths π from Π applicable to $\mathcal{I}$ with $cost(\pi) < x$, there exists a corresponding path π' from Π^x that is applicable to $\mathcal{I}^x$. In this path every reached state only differs in the variable c^i, where i is the cost of reaching the state over π':

Induction Basis l=0: The only path of length 0 is the empty path. It is applicable to both $\mathcal{I}$ and $\mathcal{I}^x$, and results in $\mathcal{I}$ and $\mathcal{I}^x$ respectively. Furthermore, we have $\mathcal{I}^x = \mathcal{I} \cup \{c^0\}$.

Induction Hypothesis: To prove the statement for paths of length $l+1$, we may use that it holds for paths of length l.

Induction step from l to $l+1$: From the induction hypothesis, we know that any path π_l applicable to $\mathcal{I}$ with length l and $cost(\pi_l) < x$ has a corresponding path π'_l in Π^x such that $\mathcal{I}^x[\![\pi'_l]\!] = \mathcal{I}[\![\pi_l]\!] \cup \{c^{cost(\pi_l)}\}$. Any action a in Π has corresponding actions a^i in Π^x for $0 \leq i < x - cost(a)$. These differ in the precondition c^i (for a^i) where i indicates the cost of the path so far. The differences in the effect of a^i are that c^i is being deleted and replaced by $c^{i+cost(a)}$. Any action a with $cost(\pi_l) + cost(a) < x$ that is applicable in $I[\![\pi_l]\!]$ has a corresponding action $a^{cost(\pi_l)}$, which has the precondition $c^{cost(\pi_l)}$ in addition to all preconditions from a. As $I^x[\![\pi'_l]\!]$ consists of all variables of $I^x[\![\pi_l]\!]$ as well as the variable $c^{cost(\pi_l)}$, the action $a^{cost(\pi_l)}$ has to be applicable in $I^x[\![\pi'_l]\!]$ and leads to a state where only the variable $c^{cost(\pi_l)+cost(a)}$ is true in addition to all variables of $I[\![\langle \pi, a \rangle]\!]$. Hence, any path $\pi = \langle \pi_l, a \rangle$ for Π with $cost(\pi) = cost(\pi_l) + cost(a) < x$ has a corresponding path $\pi' = \langle \pi'_l, a^{cost(\pi)} \rangle$ for Π^x.

In particular, this also means that if a plan π in Π with $cost(\pi) < x$ exists, then there is a corresponding plan in Π^x, since the resulting state will satisfy all goal variables.

We also show completeness with its contrapositive, namely that if Π^x is solvable, then Π must have a plan π with $cost(\pi) < x$. Because we can view Π as a projection of Π^x to the variables $\mathcal{V}^x \setminus \{c^i \mid 0 \leq i < x\} = \mathcal{V}$, it must hold that for every plan in Π^x, there is a corresponding plan in Π with equal cost. As every plan π in Π^x has $cost(\pi) < x$ by construction, the compilation approach is complete. □

Beyond soundness and completeness, we have defined three additional essential properties: generality, as well as efficient verification and generation.

For generality, it is easy to see that we can apply the described compilation to arbitrary planning tasks, and the approach is therefore as general as the unsolvability certificates it relies upon. Efficient verification holds for the same reason. Critical problems arise, however, when considering efficient generation, the last remaining property. Compiling the problem only requires space and time linear in the cost of the optimal plan, but exploring the state space of the modified task Π^x can incur an exponential overhead in general.

To illustrate this, let us consider a simple planning task where n lights must be turned on using n corresponding switches. To generate an optimal plan, let us use the A^* algorithm (Hart, Nilsson, and Raphael 1968) with the LM-cut heuristic (Helmert and Domshlak 2009), which yields the perfect heuristic for this problem and A^* thus finds a plan after n expansions.

The next step would then be to rerun the algorithm on the compiled task Π^n to prove that it is unsolvable. There are $\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n-1} = 2^n - 1$ reachable states in total. Since LM-cut is based on delete-relaxation, it can only detect dead-ends in Π^x if it does so for the corresponding state in Π, or if the state has no successor. Since in the original task no dead-ends exist, LM-cut will only detect $\binom{n}{n-1} = n$ states as dead-ends, namely those where $n-1$ lights are on and which thus have no successors. This leaves us with $2^n - 1 - n$ states that have to be expanded in order to generate the unsolvability certificate. The existence of this counterexample shows that efficient generation is not guaranteed.

We finish our discussion on unsolvability compilation with a final remark: While the task compilation is straightforward, it is a part of the certificate that is not verified, meaning bugs occurring in this step would not be detected.

Optimality Proof System

Aside from potentially inefficient generation and an unverified translation step, the compilation approach does not follow the spirit of certifying algorithms: Instead of building a certificate that explains the reasoning of the algorithm, it translates the input into a new problem solved by a different algorithm. We thus introduce a second type of certificate that is based on the idea of proof systems and uses the reasoning of the planner.

Our optimality proof system adapts almost all components of the unsolvability proof system described in the background. The only exceptions are the rules that reason about deadness. When proving optimality, we need to reason about *costs* of plans going through s, which we represent with a new type of knowledge.

Definition 2. *We say* s *has a lower bound of* x*, denoted by* $gc(s) \geq x$*, if all paths from* s *to some goal state* $s_g \in \mathcal{S}_\mathcal{G}$ *have at least cost* x*. For state set* S*, we define* $gc(S) \geq x$ *if* $gc(s) \geq x$ *for all* $s \in S$*.*

We acquire bound knowledge with several new inference rules. First, we observe the following trivial rules:

Theorem 2 (**T**rivial **C**ost bound; **E**mpty set **C**ost bound)**.** *Let* S *be a state set. The following statements are true without conditions:*

TC $gc(S) \geq 0$.
EC $gc(\emptyset) \geq \infty$.

Proof. Rule **TC** follows directly from the fact that we only consider non-negative action costs, while for rule **EC** we have that a condition over all elements of an empty set is trivially satisfied, meaning we can derive any arbitrary bound for the empty set. □

Next, we can easily see how lower bounds transfer to subsets or unions:

Theorem 3 (**S**ubset **C**ost bound; **U**nion **C**ost bound)**.** *Let* S, S' *be state sets.*

SC *If* $gc(S') \geq x$ *and* $S \subseteq S'$*, then* $gc(S) \geq x$.
UC *If* $gc(S) \geq x$ *and* $gc(S') \geq x'$*, then* $gc(S \cup S') \geq \min(x, x')$.

Proof. For rule **SC**, we know from Definition 2 that if $gc(S') \geq x$ then $gc(s) \geq x$ for all $s \in S'$ and thus also for all $s \in S \subseteq S'$, from which $gc(S) \geq x$ directly follows. For rule **UC**, we know for each $s \in S \cup S'$ that $gc(s) \geq x$ or $gc(s) \geq x'$ holds, and since $x \geq \min(x, x')$ and $x' \geq \min(x, x')$ we have $gc(s) \geq \min(x, x')$. □

Furthermore, we want to infer new lower bounds based on existing ones. We know that an optimal plan π for any non-goal state s must go through one of its successors s', and the cost of π is equal to the cost of the action reaching s' plus the cost of an optimal plan from s'. If we know lower bounds of the goal cost for all successors, we can derive a lower bound for s by minimizing over all successors, similar to Bellman updates: For example, assume a non-goal state s has two successors s_1 and s_2 reached with actions a_1 and a_2 respectively, with $\mathit{cost}(a_1) = 6$ and $\mathit{cost}(a_2) = 4$, and we know $gc(s_1) \geq 2$ and $gc(s_2) \geq 5$. Then an optimal plan for s must cost at least $\min_{i \in \{1,2\}}(\mathit{cost}(a_i) + gc(s_i)) = \min(6+2, 4+5) = 8$. The following rule generalizes this argument for state sets, grouping successors reached through actions with same cost:

Theorem 4 (**P**rogression **C**ost bound)**.** *Let S and S_i with $1 \leq i \leq c^{\max}(\mathcal{A})$ be state sets. As a reminder, $A_{\mathcal{A}|i}$ is the set of all actions from $\mathcal{A}$ with cost i.*

PC *If $S \cap \mathcal{S}_{\mathcal{G}} \subseteq \emptyset$, and for all $1 \leq i \leq c^{\max}(\mathcal{A})$ we have $S[\![A_{\mathcal{A}|i}]\!] \subseteq S \cup S_i$ and $gc(S_i) \geq c_i$, then $gc(S) \geq \min_{1 \leq i \leq c^{\max}(\mathcal{A})}(c_i + i)$.*

Proof. Consider an arbitrary $s \in S$. If there is no plan for s, then $gc(s) \geq x$ is true for any x. Otherwise, let $\pi = \langle a_1, \ldots, a_n \rangle$ be an optimal plan for s. We define $\pi_x^{x'} = \langle a_x, \ldots, a_{x'} \rangle$ to be a subsequence of π, with $\pi_x^{x'} = \langle\rangle$ if $x' < x$. Since $S \cap \mathcal{S}_{\mathcal{G}} \subseteq \emptyset$, we know that the plan must leave S at some point t, i. e., $s[\![\pi_1^{t-1}]\!] \in S$ and $s[\![\pi_1^t]\!] \notin S$ for some $1 \leq t \leq n$. We know that $S[\![A_{\mathcal{A}|\mathit{cost}(a_t)}]\!] \subseteq S \cup S_{\mathit{cost}(a_t)}$ and thus $s[\![\pi_1^t]\!] \in S_{\mathit{cost}(a_t)}$. We also know that the path π_{t+1}^n must be an optimal plan for $s[\![\pi_1^t]\!]$ since π is an optimal plan for s, and together with $gc(S_{\mathit{cost}(a_t)}) \geq c_{\mathit{cost}(a_t)}$, we have that the cost of π_{t+1}^n is at least $c_{\mathit{cost}(a_t)}$. From this follows that the cost of π_t^n is at least $c_{\mathit{cost}(a_t)} + \mathit{cost}(a_t)$. Since π_t^n is a subsequence of π, we can conclude that the cost of π, and thus the optimal cost for s, is at least $c_{\mathit{cost}(a_t)} + \mathit{cost}(a_t) \geq \min_{1 \leq i \leq c^{\max}(\mathcal{A})}(c_i + i)$. Since s was chosen arbitrarily, the bound holds for any $s \in S$, meaning we have $gc(S) \geq \min_{1 \leq i \leq c^{\max}(\mathcal{A})}(c_i + i)$. □

The reason we allow the successors of S with $A_{\mathcal{A}|i}$ to be in S itself is twofold. First, it allows us to consider states where not all successors have a (nontrivial) lower bound yet; for example, in a unit-cost task we can derive that all non-goal states have a lower bound of 1 with a single application of **PC**. Secondly, it allows us to derive higher bounds in the presence of 0-cost cycles by defining S such that it contains all states reached with only 0-cost actions. We can then set $S_0 = \emptyset$, for which we can derive a lower bound of ∞, since all successors of $S[\![A_{\mathcal{A}|0}]\!]$ must be in S itself.

Finally, we define a rule that infers a lower bound for the optimal plan cost of Π using a known lower bound for $\mathcal{I}$.

Theorem 5 (**I**nitial state **B**ound)**.** *Let $\mathcal{I}$ be the initial state of a planning task Π.*

IB *If $gc(\{\mathcal{I}\}) \geq x$, then the optimal plan cost of Π is at least x.*

Proof. If we have $gc(\{\mathcal{I}\}) \geq x$, then all plans for $\mathcal{I}$ must have at least cost x. These are exactly the plans for Π, meaning we must have optimal plan cost for Π of at least x. □

Having fully defined our optimality proof system, we analyze its practical usefulness based on our four properties.

Theorem 6. *The optimality proof system is sound and complete.*

Proof. A proof system is sound iff all its rules are sound. For all rules from the unsolvability proof system this has been shown in previous work, and we showed soundness for the additional rules concerning costs in Theorems 2–5.

For completeness, we present a certificate for an arbitrary planning task Π with optimal plan cost $c(\Pi)$, and prove that each step is correct. Table 1 depicts such a certificate, where S_i contains all states with optimal plan cost of at least i.

Step (g_i) shows with basic statements that no S_i contains a goal state, which is correct because goal states have an optimal plan of cost 0 (the empty path) and thus cannot have an optimal plan with at least cost i (for $i > 0$). Next, the three versions of $(p_{i,j})$ state that if we apply an action with cost j to any state s in S_i, the successor s' must be contained either (a) in $S_i \cup \emptyset = S_i$ if $j = 0$, (b) in $S_i \cup S_{i-j}$ if $i > j$, or (c) in $S_i \cup \mathcal{S}$ if $i \leq j$. This is correct since (a) if s' is reached with a 0-cost action its optimal plan cost must be at least as high as the optimal plan cost for s and thus $s' \in S_i$, (b) if s' is reached with cost j its optimal plan must be at least the optimal plan cost of s minus j (i. e. $i-j$) and thus $s' \in S_{i-j}$, and (c) the set of all states $\mathcal{S}$ trivially contains s'. Note that while we do not need a union in any of the cases, we write it in this fashion such that we can later directly apply rule **PC**.

The main part of the proof consists of iteratively deducing lower bounds for all S_i. We first use rule **EC** to show that $\emptyset$ has an infinite lower bound, as well as rule **TC** to get a trivial lower bound of 0 for $\mathcal{S}$. We can then apply rule **PC** repeatedly, starting with S_1 and incrementing the index up to $c(\Pi)$, which shows that S_i has a lower bound of i. To show that the deduction is correct, consider an arbitrary step (c_i). To apply rule **PC**, we need to know

- that S_i does not contain goal states (shown in (g_i)),
- that for $1 \leq j \leq c^{\max}(\mathcal{A})$ we have that $S_i[\![A_{\mathcal{A}|i}]\!] \subseteq S_i \cup S'$ (shown in steps (p) with $S' = \emptyset$ for $j = 0$, $S' = S_{i-j}$ for $i > j$ and $S' = \mathcal{S}$ otherwise), and
- that we have a bound $gc(S') \geq c'$ for all S' occurring above (shown in steps (c)).[1]

The rule thus correctly deduces that the lower bound for S_i is the minimum of all $c' + j$, that is the minimum over $\infty + 0 = \infty$ for $j = 0$, $(i-j) + j = i$ for all $i > j$ and $0 + j = j$ for all $i \leq j$, which is i.

[1]This does not lead to circular reasoning since we only require $(c_{i'})$ with $i' < i$, which has already been established at this point.

index	knowledge	rule	preconditions	comment
(g_i)	$S_i \cap \mathcal{S}_\mathcal{G} \subseteq \emptyset$	**B1**		for all $1 \leq i \leq c(\Pi)$
$(p_{i,0})$	$S_i[\![A_{\mathcal{A}\|0}]\!] \subseteq S_i \cup \emptyset$	**B2**		for all $1 \leq i \leq c(\Pi)$
$(p_{i,j})$	$S_i[\![A_{\mathcal{A}\|j}]\!] \subseteq S_i \cup S_{i-j}$	**B2**		for all $1 \leq i \leq c(\Pi)$, $1 \leq j < i$
$(p_{i,j'})$	$S_i[\![A_{\mathcal{A}\|j'}]\!] \subseteq S_i \cup \mathcal{S}$	**B2**		for all $1 \leq i \leq c(\Pi)$, $i \leq j' < c^{\max}(\mathcal{A})$
$(c_\emptyset)$	$gc(\emptyset) \geq \infty$	**EC**		
(c_0)	$gc(\mathcal{S}) \geq 0$	**TC**		
(c_i)	$gc(S_i) \geq i$	**PC**	(g_i), $(p_{i,0})$, $(c_\emptyset)$, $(p_{i,1})$, (c_{i-1}), $\ldots$, $(p_{i,c_\mathcal{A}})$, $(c_{\max(0,i-c_\mathcal{A})})$	for all $1 \leq i \leq c(\Pi)$
(1)	$\{\mathcal{I}\} \subseteq S_{c(\Pi)}$	**B1**		
(2)	$gc(\{\mathcal{I}\}) \geq c(\Pi)$	**SC**	$(c_{c(\Pi)})$, (1)	
(3)	optimal plan cost $\geq c(\Pi)$	**IB**	(2)	

Table 1: Optimality proof for a planning task with optimal cost $c(\Pi)$, where S_i represents all states with goal cost of at least i.

The proof then finishes by establishing with basic statement **B1** that the initial state is contained in $S_{c(\Pi)}$, which must be the case because the optimal plan costs are $c(\Pi)$, and applying rule **SC** to transfer the lower bound from $S_{c(\Pi)}$ to $\{\mathcal{I}\}$, which is used to finish the proof with rule **IB**, concluding that Π has an optimal plan cost of at least $c(\Pi)$. $\square$

The other three properties for practical usability do not have clear-cut answers. Contrary to our first approach, we can only show that the optimality proof system is general by specifying certificates for a variety of concrete algorithms. We remark however that proof systems are expandable by nature, meaning that we can introduce new inference rules that capture different reasoning techniques if needed. Efficient generation and verification depends on the concrete algorithm as well. For generation, we need to show that certificate size does not grow exponentially, i. e. both the number of steps needed in the proof and the size of the state set representation must be bounded polynomially. For verification, we only need to consider basic statements, since verifying the correct application of inference rules is a purely syntactical check. While this generally depends on the basic statements and formalisms used, we will show that all state sets needed for the certificates we describe in this work can be efficiently represented with BDDs, for which previous work has already shown that efficient verification is guaranteed.

Generating Optimality Proofs

While Table 1 provides a possible optimality proof for any solvable planning task, we can in general not efficiently generate it since it would require to know the optimal plan cost for every state $s \in \mathcal{S}$ in order to build the sets S_i. Instead, we consider concrete planning algorithms based on forward search and show how we can create a similar certificate only using knowledge the algorithm derives while finding an optimal plan.

Blind Search

Blind forward search algorithms annotate each state s they see with a g value denoting the cost of the cheapest path from $\mathcal{I}$ to s. We consider uniform-cost search, which expands states in ascending order of their g value. The algorithm terminates upon expanding a goal state, at which point it is guaranteed that the found solution has optimal cost $c(\Pi)$ and that all seen states with $g(s) < c(\Pi)$ were expanded.

While blind search does not reason about cost from state s to the goal, we can infer a lower bound based on the g value and the optimal plan cost, namely $gc(s) \geq c(\Pi) - g(s)$. We define an optimality proof that will establish these bounds iteratively for states s with $c(\Pi) - g(s) \geq i$ for increasing i. It does so by applying rule **PC** with the knowledge that for all successors s' of s reached with an action with cost j, we either have $c(\Pi) - g(s') \geq i$ or the lower bound $\max(0, c(\Pi) - (g(s) + j))$ has already been established. If we define sets $S_i = \{s | c(\Pi) - g(s) \geq i\}$ for $1 \leq i \leq c(\Pi)$, we can actually directly use the proof shown in Table 1. Intuitively, this is the case because this proof works for any definition of S_i with the property that they do not contain goal states and that all successors $s' \in S_i[\![A_{\mathcal{A}|j}]\!]$ with $j < i$ are in S_{i-j}. This is true for the S_i defined here because all $s \in S_i$ are expanded, meaning they are not a goal state and the algorithm has seen their successors and put them in their corresponding sets. To show the claim more formally, we show that the basic statements still hold, which directly proves the correctness of the inference rules and thus the entire proof.

Knowledge (g_i) still holds for all redefined S_i, since no goal state can have a g value smaller than $c(\Pi)$ (otherwise the algorithm would have expanded it). Looking at knowledge derived in steps (p), the statement trivially holds for any $j \geq i$, since all states are contained in $\mathcal{S}$. For $j < i$ we know from $s \in S_i$ that $g(s)$ must be lower than $c(\Pi)$, meaning s must have been expanded, and thus any successor s' reached with an action from $A_{\mathcal{A}|j}$ must have been assigned a g value of $g(s) + j$ or lower (if a cheaper path to s' not going through s was found). From this, we can conclude that $s' \in S_{i-j}$ must hold. This in particular also means for $(p_{i,0})$ that $s' \in S_i$ must hold. Finally, knowledge (1) also still holds since $c(\Pi) - g(\mathcal{I}) = c(\Pi) - 0 \geq c(\Pi)$ and thus $\mathcal{I} \in S_{c(\Pi)}$.

The number of steps in the proof is linear in $c^{\max}(\mathcal{A}) \cdot c(\Pi)$. Furthermore, we can efficiently represent sets S_i as BDDs by adding each expanded state to the appropriate sets, which can be done in time linear to the amount of variables (Eriksson 2019). We thus conclude that proofs for blind search can be generated efficiently.

index	knowledge	rule	preconditions	comment
(g_i)	$G_i \cap \mathcal{S}_\mathcal{G} \subseteq \emptyset$	**B1**		for all $1 \leq i \leq c(\Pi)$
(ch_s)	$gc(\{s\}) \geq h(s)$	[shown by heuristic]		for all $s \in \bigcup_{i=1}^{c(\Pi)} H_i$
(ch_i)	$gc(H_i) \geq i$	**UC**	(ch_s) for all $s \in H_i$	for all $1 \leq i \leq c(\Pi)$
$(p_{i,0})$	$G_i[\![A_{\mathcal{A}\|0}]\!] \subseteq G_i \cup H_i$	**B2**		for all $1 \leq i \leq c(\Pi)$
$(p_{i,j})$	$G_i[\![A_{\mathcal{A}\|j}]\!] \subseteq G_i \cup (G_{i-j} \cup H_{i-j})$	**B2**		for all $1 \leq i \leq c(\Pi), 1 \leq j < i$
$(p_{i,j'})$	$G_i[\![A_{\mathcal{A}\|j'}]\!] \subseteq G_i \cup \mathcal{S}$	**B2**		for all $1 \leq i \leq c(\Pi), i \leq j' \leq c^{\max}(\mathcal{A})$
(c_0)	$gc(\mathcal{S}) \geq 0$	**TC**		
(cg_i)	$gc(G_i) \geq i$	**PC**	(g_i), $(p_{i,0})$, (ch_i), $(p_{i,1})$, (c_{i-1}), $\ldots$, $(p_{i,c_\mathcal{A}})$, $(c_{\max(0,i-c_\mathcal{A})})$	for all $1 \leq i \leq c(\Pi)$
(c_i)	$gc(G_i \cup H_i) \geq i$	**UC**	(cg_i), (ch_i)	for all $1 \leq i \leq c(\Pi)$
(1)	$\{\mathcal{I}\} \subseteq G_{c(\Pi)} \cup H_{c(\Pi)}$	**B1**		
(2)	$gc(\{\mathcal{I}\}) \geq c(\Pi)$	**SC**	$(c_{c(\Pi)})$, (1)	
(3)	optimal plan cost $\geq c(\Pi)$	**IB**	(2)	

Table 2: Optimality proof for heuristic search.

Heuristic Search

Heuristic forward search is similar to its blind counterpart, but instead of expanding states according to only their g value, heuristic search makes use of a *heuristic* h that estimates the cost from s to the goal. We consider A^*, which expands states according to $f(s) = g(s) + h(s)$ and guarantees that the plan obtained from the first expanded goal state is optimal if the used heuristic is *admissible*, i. e. never overestimates the true cost to the goal. Furthermore, we have for all states s that if $g(s) + h(s) < c(\Pi)$ then s has been expanded.

In this scenario we can no longer directly use the proof from Table 1 since not all successors $s' \in S_i[\![A_{\mathcal{A}|j}]\!]$ of $s \in S_i$ are in S_{i-j} for $i > j$, namely in the case where s was not expanded, and we thus never saw s'. However, this implies $h(s) + g(s) \geq c(\Pi)$, meaning the heuristic value $h(s)$ provides a cost bound at least as tight as $c(\Pi) - g(s)$. Assuming that the heuristic can translate its reasoning into the proof system, i. e. derive $gc(s) \geq h(s)$, we can modify the proof by first letting the heuristic prove lower bounds for all states that were seen but not expanded, and then using this information to prove the lower bounds for expanded states. Table 2 presents a formal definition of a proof following this idea, where for $1 \leq i \leq c(\Pi)$ we define

- $G_i = \{s \mid c(\Pi) - g(s) \geq i, c(\Pi) - g(s) > h(s)\}$ and
- $H_i = \{s \mid h(s) \geq i, c(\Pi) - g(s) \leq h(s)\}$.

The proof divides seen states into two types of sets, G_i and H_i, based on whether their g or h value is used for proving a lower bound. Again, we first state in step (g_i) that G_i does not contain any goal state for all $1 \leq i \leq c(\Pi)$, which holds because all states in any G_i have $g(s) + h(s) < c(\Pi)$, meaning they have been expanded and cannot be goal states (otherwise the algorithm would have terminated earlier). For each state s in some H_i, knowledge (ch_s) lets the heuristic establish why $h(s)$ is a correct lower bound for s, and from this knowledge (ch_i) deduces with rule **UC** that $gc(H_i) \geq i$ holds for all $1 \leq i \leq c(\Pi)$.[2]

[2]Technically we would need to do this in $|H_i| - 1$ steps as **UC** only considers the union of two sets, but we summarize for brevity.

Knowledge $(p_{i,j})$ for $1 \leq i \leq c(\Pi)$ and $0 \leq j \leq c^{\max}(\mathcal{A})$ states where we can find the successors of any $s \in G_i$: Let $s' = s[\![a]\!]$ with $a \in A_{\mathcal{A}|j}$ be an arbitrary successor of s. If $j \geq i$ we trivially have $s' \in \mathcal{S}$. Otherwise, it is sufficient to show that $s' \in G_{i-j} \cup H_{i-j}$. From $s \in G_i$, we have $c(\Pi) - g(s) \geq i$ and $c(\Pi) - g(s) > h(s)$. From the latter we see that s must have been expanded, meaning the algorithm has seen s' and we must have $g(s') \leq g(s) + j$. Combining this with $c(\Pi) - g(s) \geq i$ yields $g(s') - j \leq c(\Pi) - i$ or $c(\Pi) - g(s') \geq i - j$. If $c(\Pi) - g(s') > h(s')$, then $s' \in G_{i-j}$, otherwise we have $h(s') \geq i - j$ and thus $s' \in H_{i-j}$.

The proof continues with establishing in (c_0) that $\mathcal{S}$ has a trivial lower bound of 0. The lower bound for G_i is then proven by applying rule **PC**, which can be applied since for all $A_{\mathcal{A}|j}$ we have shown with $(p_{i,j})$ and (c_{i-j}) (or (ch_i) for $j = 0$) that $G_i[\![A_{\mathcal{A}|j}]\!] \subseteq G_i \cup S'$ and $gc(S') \geq i - j$, meaning we must have $gc(G_i) \geq i$. Since we already have a lower bound for H_i from (ch_i), we can now say in (c_i) with rule **UC** that $gc(G_i \cup H_i) \geq i$. We can then finish the proof similar to previous cases: We first state in (1) that $\mathcal{I}$ must be in $G_{c(\Pi)}$ or $H_{c(\Pi)}$, which is true since from $g(\mathcal{I}) = 0$ we have $c(\Pi) - g(\mathcal{I}) \geq c(\Pi)$ and thus $\mathcal{I} \in G_c(\Pi)$ if $h(\mathcal{I}) < c(\Pi)$, otherwise $h(\mathcal{I}) \geq c(\Pi)$ and thus $\mathcal{I} \in H_c(\Pi)$. With this, we can conclude with **SC** that $gc(\{\mathcal{I}\}) \geq c(\Pi)$ from $c_{c(\Pi)}$ and (1), because the initial state is in a set with a lower bound of $c(\Pi)$. Finally, we apply rule **IB** using (2), showing that the optimal plan cost of Π must be at least $c(\Pi)$.

Looking at the size of the proof, we can bound the number of (ch_s) steps by the number of evaluated states, which also bounds the number of (ch_i) steps by the same number, even when only building a union of two sets at a time. Furthermore, as with blind search, the number of $(p_{i,j})$ steps is linear in $c^{\max}(\mathcal{A}) \cdot c(\Pi)$, and the number of (g_i), (cg_i) and (c_i) steps is linear in $c(\Pi)$. Putting everything together, we can see that the number of steps is polynomially bounded by the runtime of the algorithm. Additionally, we can again represent all sets mentioned in Table 2 as BDDs, which means we can guarantee efficient generation for everything except the sub-proofs from the heuristic. Note however that this requires the heuristic to provide a proof for $\{s\}$ *represented as a BDD*, which can be challenging if other formalisms would be better suited for describing its reasoning.

What is left to show is if and how we can translate the heuristic's reasoning into the proof system. Since every heuristic needs to be investigated separately this is beyond the scope of this paper, but we present the case of h^{max} here.

h^{max} Heuristic

Given state s, the h^{max} heuristic (Bonet and Geffner 2001) computes for every variable $v \in \mathcal{V}$ the cost to reach v from s in the delete relaxation with a fixpoint iteration: $c(v, s) = 0$ if $v \in s$ and otherwise $c(v, s) = \min_{a \in \mathcal{A}, v \in add(a)}(\max_{p \in pre(a)} c(p, s) + cost(a))$. The overall heuristic value is $h^{max}(s) = \max_{g \in \mathcal{G}} c(g, s)$.

The h^{max} heuristic assumes that from s we can reach state s' with cost i as long as s' only contains variables v with $c(v, s) \leq i$. In particular, it assumes that we can reach a goal with cost h^{max}. Since this is an underestimation of the true cost, we can deduce that the cost from s' to any goal state must be at least $h^{max}(s) - i$. We define state sets $S_i = \{s' \mid c(v, s) \leq h^{max}(s) - i \text{ for all } v \in s'\}$ for $1 \leq i \leq h^{max}(s)$, representing all states for which h^{max} deduces $gc(S_i) = h^{max}(s) - (h^{max}(s) - i) = i$. These sets can be used in the proof structure of Table 1 to show $gc(S_{h^{max}(s)}) \geq h^{max}(s)$. Instead of finishing the proof with steps (1)–(3), we will apply rule **B1** to show $\{s\} \subseteq S_{h^{max}(s)}$ (which holds since s contains only variables with $c(v, s) = 0$), and then use rule **SC** to conclude $gc(\{s\}) \geq h^{max}(s)$.

As mentioned before, the proof structure from Table 1 works if (a) no S_i contains any goal state and (b) we have $S_i[\![A_{\mathcal{A}|j}]\!] \subseteq S_{i-j}$ for all $j < i$. For (a), we know that $c(g, s) = h^{max}(s)$ for some goal variable $g \in \mathcal{G}$, and since states in any S_i can only contain variables with $c(v, s) \leq h^{max}(s) - i$, no S_i can contain a state where g is true. For (b) we consider an arbitrary $1 \leq i \leq h^{max}(s)$, $s' \in S_i$, $1 \leq j \leq c^{\max}(\mathcal{A})$ and $a \in A_{\mathcal{A}|j}$ applicable to s'. To show that $s'[\![a]\!]$ is contained in S_{i-j}, we need to show $c(v, s) \leq h^{max}(s) - (i - j) = h^{max}(s) - i + j$ for all $v \in s'[\![a]\!]$. We know that any such v was either already true in s' or was added by a. In the former case, we must have $c(v, s) \leq h^{max}(s) - i \leq h^{max}(s) - i + j$ since $s' \in S_i$. In the latter case we have from the definition of h^{max} that $c(v, s) = min_{a' \in \mathcal{A}, v \in add(a')}(\max_{p \in pre(a')} c(p, s) + cost(a')) \leq \max_{p \in pre(a)} c(p, s) + j$ because a is one of the actions that add v. Furthermore, all preconditions are true in s' and thus must have $c(p, s) \leq h^{max}(s) - i$, meaning we have $c(v, s) \leq h^{max}(s) - i + j$ in this case as well.

To show efficient verification, we observe that the length of the proof is linear in $c^{\max}(\mathcal{A}) \cdot h^{max}(s)$, with the latter bounded by $c(\Pi)$. Furthermore, We can represent S_i with BDD $\varphi_i = \bigwedge_{\{v \mid c(v,s) > h^{max}(s) - i\}} \neg v$, whose representation size is linear in $\mathcal{V}$.

Experimental Evaluation

In order to test the practical feasibility of our approaches, we implemented them on top of Fast Downward 21.12 (Helmert 2006) and ran them on the STRIPS subset of the standard IPC benchmark collection. Our experiments were run on single cores of Intel Xeon E5-2660 processors with a clock speed of 2.2 GHz and using the Downward Lab environment (Seipp et al. 2017). Unless stated otherwise, each run was given a 30-minute time and 3.5 GiB memory limit. We published all code, benchmarks, and data (Mugdan, Christen, and Eriksson 2023).

For the compilation approach, we first calculated an optimal plan with A^* and the LM-cut heuristic (Helmert and Domshlak 2009). Using the computed plan length x as well as the original PDDL files as input, we ran our purpose-built translator to generate the compiled task Π^x. The current implementation only supports unit-cost tasks, but more sophisticated encodings are possible. This step takes time in the order of milliseconds and is not represented in later results. Next, we executed two algorithms to generate unsolvability certificates for the compiled tasks, namely h^{max} (u-h^{max}), which computes the same dead-ends as LM-cut but is faster to compute, and $h^{\text{M\&S}}$ (u-$h^{\text{M\&S}}$; Helmert, Haslum, and Hoffmann 2007; Helmert et al. 2014), which offers a more informed heuristic to compare to. Implementations stem from previous work (Eriksson, Röger, and Helmert 2018).

For the proof system approach, we extended Fast Downward with the ability to generate optimality certificates for A^* search with the blind[3] heuristic (o-blind) and the h^{max} heuristic (o-h^{max}). Because both the plan and certificate are computed in a single step, the optimality certificate approach is more resource constrained than the compilation. To see the overhead caused by certifying the result, we also run the non-certifying versions of these algorithms as a baseline.

Verification of both the unsolvability and optimality certificates was performed by the verifier Helve[4], which was extended with new rules to support optimality certificates. This step ran for an additional 4 hours for each successfully created certificate. In the following we will use *create* to denote the approaches' respective steps where certificates are created, and *verify* to denote their verification steps.

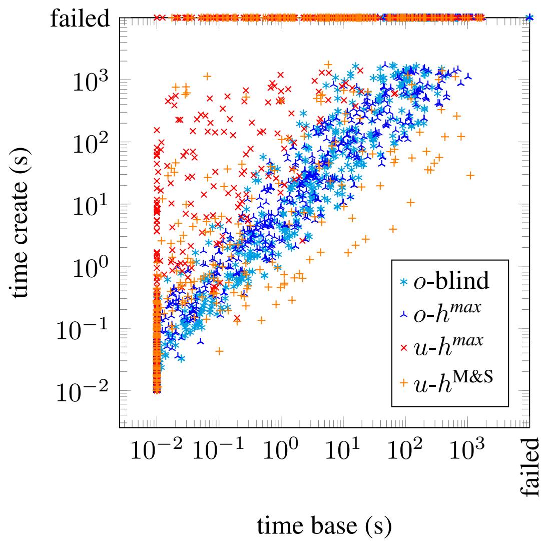
Table 3 shows for how many tasks the configuration could successfully find an optimal plan (without certificate), create a certificate, and verify said certificate. Moreover, it is shown how much coverage is lost between steps due to ex-

		base	created	verified
U	u-h^{max}	549	292 ($-229/-28$)	278 ($-8/-6$)
	u-$h^{\text{M\&S}}$	549	339 ($-174/-36$)	316 ($-0/-23$)
	o-blind	413	372 ($-36/-5$)	**364** ($-7/-1$)
	o-h^{max}	448	**380** ($-34/-34$)	324 ($-50/-6$)
$\overline{\text{U}}$	o-blind	287	248 ($-34/-5$)	**243** ($-5/-0$)
	o-h^{max}	345	**272** ($-41/-32$)	234 ($-22/-16$)

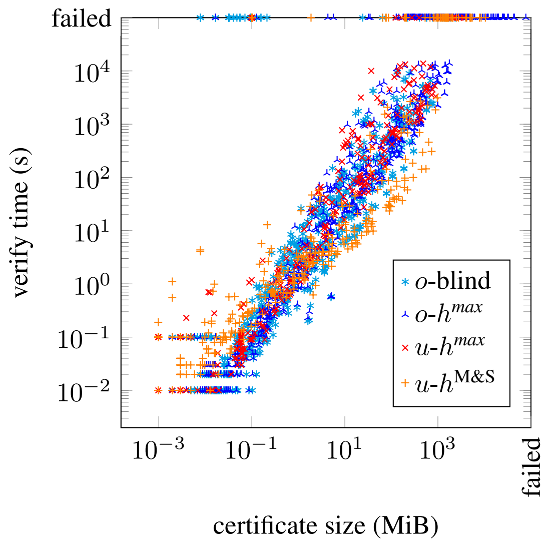
Table 3: Coverage for non-certifying (base) planners, certifying (created) planners, and for successful verification (verified) of generated certificates. In the created and verified columns, ($-x/-y$) denotes that x tasks were lost due to timeout and y due to memory limit. Part U considers 1190 unit-cost tasks and part $\overline{\text{U}}$ considers 637 non unit-cost tasks.

[3]We changed the implementation of the blind heuristic to always return 0 instead of the minimal action cost.

[4]https://github.com/salome-eriksson/helve



(a) Runtime comparison between base and create, showing how much overhead creating a certificate causes.



(b) Verification time in relation to the size of the certificate to be verified.

Figure 1: Efficiency analysis of certificate generation and verification.

ceeding either time or memory limits. Looking at creating certificates, we see that the compilation based approaches perform significantly worse, despite having more forgiving resource constraints. Of the two heuristics, u-$h^{\mathrm{M\&S}}$ performs better than u-h^{max}. This aligns with our theoretical result that compilation based certificates cannot always be generated efficiently, and that u-h^{max} has weaker dead-end detection than $h^{\mathrm{M\&S}}$, which is detrimental in the compiled task.

The optimality proof system based configurations can create certificates for 88% (o-blind) respectively 82% (o-h^{max}) of tasks solved by base. While this is still a significant drop, we deem these results encouraging given that certifying optimality is expected to come with some overhead. In terms of verification, o-h^{max} only verifies 86% of created certificates, while o-blind reaches 98%. We explain these results by observing that the certificate size and thus complexity grows significantly with the number of sub-certificates needed for heuristic values: For o-blind we do not need any, for compilation based approaches we only need them for dead-ends, and for o-h^{max} we need one for every state that was not expanded due to its heuristic value.

Figure 1 gives a more detailed look into the results, focusing on runtime. In Figure 1a, which compares certifying and non-certifying time, we can see that o-h^{max} and o-blind are on average about a factor of 10 slower when generating certificates, but stay within polynomial overhead (shown by the linear relationship in the log plot). This is not the case for u-h^{max} and u-$h^{\mathrm{M\&S}}$: The former is often slower by a factor of 100 or higher, while the latter can perform equally bad but better than optimality proof based approaches. We attribute this to the strength of $h^{\mathrm{M\&S}}$. In rare cases, it is even faster than its base version, which indicates that $h^{\mathrm{M\&S}}$ provides better pruning in the compiled task than LM-cut provides guidance in the original task.

Figure 1b shows verification time as a function of the size of the certificate and confirms that all approaches can efficiently verify created certificates. Moreover, it confirms that o-h^{max} tends to have the largest certificates, which is especially visible by the size of the non-verified certificates.

Conclusion

We introduce two approaches for verifying optimality in classical planning based on the idea of certifying algorithms; one based on a compilation to unsolvability and one based on a proof system that directly reasons about costs. While both approaches are sound and complete, the compilation suffers from several drawbacks, chief among which is that efficient generation cannot be guaranteed. Our experimental evaluation confirms these theoretical results, showing that the proof system approach generally performs better, but both approaches are feasible in the majority of cases.

The compilation approach can be a useful fallback when no other method of verification is available, since it can be used in any setting as long as we know the optimal plan cost. But if applicable, the optimality proof system is the better choice, and we are confident that it can be used for a large number of optimal planning algorithms due to its expandable nature. Difficulties can arise when different parts of the algorithm require different formalisms to efficiently represent their reasoning. However, the existing unsolvability proof system already has strategies to deal with this, and we expect to be able to adapt them given how similar the two systems are. Furthermore, it is unclear if and how knowledge provided by different heuristics can be combined. A maximum over several heuristics can be covered by simply selecting the appropriate heuristic for each state separately, but techniques like cost partitioning require more sophisticated reasoning, opening an interesting avenue for future research.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Certified Correctness and Guaranteed Performance for Domain-Independent Planning" (CCGP-Plan). Moreover, this research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

References

Abdulaziz, M.; and Lammich, P. 2018. A Formally Verified Validator for Classical Planning Problems and Solutions. In Tsoukalas, L. H.; Grégoire, É.; and Alamaniotis, M., eds., *Proceedings of the 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*, 474–479. IEEE.

Bogaerts, B.; Gocht, S.; McCreesh, C.; and Nordström, J. 2022. Certified Symmetry and Dominance Breaking for Combinatorial Optimisation. In *Proceedings of the Thirty-Sixth AAAI conference on Artificial Intelligence (AAAI 2022)*, 3698–3707. AAAI Press.

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

Cruz-Filipe, L.; Heule, M. J. H.; Hunter, W. A., Jr; Kaufmann, M.; and Schneider-Kamp, P. 2017. Verifying Refutations with Extended Resolution. In de Moura, L., ed., *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, 220–236. Springer.

Elffers, J.; Gocht, S.; McCreesh, C.; and Nordström, J. 2020. Justifying All Differences Using Pseudo-Boolean Reasoning. In *Proceedings of the Thirty-Fourth AAAI conference on Artificial Intelligence (AAAI 2020)*, 1486–1494. AAAI Press.

Eriksson, S. 2019. *Certifying Planning Systems: Witnesses for Unsolvability*. Ph.D. thesis, University of Basel.

Eriksson, S.; Röger, G.; and Helmert, M. 2018. A Proof System for Unsolvable Planning Tasks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 65–73. AAAI Press.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Floyd, R. W. 1967. Assigning meaning to programs. In Schwartz, J. T., ed., *Mathematical Aspects of Computer Science: Proceedings of Symposia in Applied Mathematics*, volume 19, 19–31. American Mathematical Society.

Gentzen, G. 1935. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39: 176–210.

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

Haslum, P. 2011. INVAL: the Independent PDDL plan Validator. https://github.com/patrikhaslum/INVAL. Accessed November 29, 2022.

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 Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. 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 ACM*, 61(3): 16:1–63.

Heule, M. J. H.; Hunter, W. A., Jr; and Wetzler, N. 2013a. Trimming while Checking Clausal Proofs. In *Formal Methods in Computer-Aided Design (FMCAD 2013)*, 181–188. IEEE.

Heule, M. J. H.; Hunter, W. A., Jr; and Wetzler, N. 2013b. Verifying Refutations with Extended Resolution. In Bonacina, M. P., ed., *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, 345–359. Springer.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10): 576–580.

Howey, R.; and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In Edelkamp, S.; and Hoffmann, J., eds., *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks.*

Järvisalo, M.; Heule, M. J. H.; and Biere, A. 2012. Inprocessing Rules. In Gramlich, B.; Miller, D.; and Sattler, U., eds., *Proceedings of the Sixth International Joint Conference on Automated Reasoning (IJCAR 2012)*, 355–370. Springer.

McCarthy, J. 1963. A basis for a mathematical theory of computation. In Braffort, P.; and Hirschberg, D., eds., *Computer Programming and Formal Systems*, volume 35, 33–70. North-Holland.

McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying algorithms. *Computer Science Review*, 5(2): 119–161.

Mugdan, E.; Christen, R.; and Eriksson, S. 2023. Benchmarks, Code and Data from Mugdan et al, ICAPS 2023. https://doi.org/10.5281/zenodo.7733612.

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

Vandesande, D.; Wulf, W. D.; and Bogaerts, B. 2022. QMaxSATpb: A Certified MaxSAT Solver. In *Logic Programming and Nonmonotonic Reasoning (LPNMR 2022)*, volume 13416 of *Lectures in Computer Science*, 429–442. Springer.

Wetzler, N.; Heule, M. J. H.; and Hunter, W. A., Jr. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In Sinz, C.; and Egly, U., eds., *Theory and Application of Satisfiability Testing (SAT 2014)*, volume 8561 of *Lecture Notes in Computer Science*, 422–429. Springer.