Finding Minimal Plan Reductions Using Classical Planning

MAURICIO SALERNO, Universidad Carlos III de Madrid, España
RAQUEL FUENTETAJA, Universidad Carlos III de Madrid, España
JENDRIK SEIPP, Linköping University, Sweden

While classical planning research has made tremendous progress in the last decades, many complex tasks can still only be solved suboptimally. The satisficing plans found for these tasks often contain actions that can be removed while maintaining plan validity. Removing such redundant actions is desirable since it can decrease the plan cost and simplify the plan. Reducing a plan to a minimum-cost plan without redundant actions is NP-complete and previous work addressed this problem with a compilation to weighted MaxSAT. In this work, we propose several simple and natural formulations to encode this problem as a classical planning task, and prove that solving the resulting tasks optimally guarantees finding minimal plan reductions. We analyze the relation of the classical planning formulations to the MaxSAT compilation, and prove theoretical properties of the known concept of *plan action landmarks*. Finally, we evaluate the new approaches experimentally and show that they are competitive with the previous state of the art in minimal plan reduction.

JAIR Associate Editor: Siddharth Srivastava

JAIR Reference Format:
Mauricio Salerno, Raquel Fuentetaja, and Jendrik Seipp. 2025. Finding Minimal Plan Reductions Using Classical Planning. *Journal of Artificial Intelligence Research* 84, Article 10 (October 2025), 35 pages. DOI: 10.1613/jair.1.19437

1 Introduction

Modern satisficing planning systems are able to solve large planning tasks efficiently (Lipovetzky and Geffner 2017; Richter and Westphal 2010). However, solutions found by these planners can be far from optimal, even producing plans with *redundant actions*. Intuitively, an action in a plan is redundant if it can be removed without affecting the plan's validity. In turn, a subsequence of actions in a plan is redundant if it can be removed without affecting validity, and plans without redundant subsequences are called *perfectly justified* (Fink and Yang 1992; Nebel et al. 1997).

It is easy to see that justified plans are preferable to non-justified plans. First, cheaper plans are preferable, and removing redundant actions can lead to cost reductions. Second, in a plan reuse setting (Fink and Yang 1992), where a plan found for a task is reused to solve a subtask, removing redundant actions which are only related to goals in the original task that are not considered in the subtask can lead to a more efficient plan. Third, Olz and Bercher 2019 argue that removing redundant actions also improves plan explanations. Fourth, finding justified plans is particularly important in settings such as top-k planning (Katz and Sohrabi 2022; Katz, Sohrabi, and Udrea 2020; Katz, Sohrabi, Udrea, and Winterer 2018; Speck et al. 2020), especially when diversity is required (Katz, Sohrabi, and Udrea 2022; Srivastava et al. 2007). Alternative plans with redundant actions are bound to have little practical value since they are the result of adding loops or other types of redundant actions to actually

Authors' Contact Information: Mauricio Salerno, ORCID: 0000-0002-7664-5847, msalerno@pa.uc3m.es, Universidad Carlos III de Madrid, Leganés, Madrid, España; Raquel Fuentetaja, ORCID: 0000-0002-3856-2629, rfuentet@inf.uc3m.es, Universidad Carlos III de Madrid, Leganés, Madrid, España; Jendrik Seipp, ORCID: 0000-0002-2498-8020, jendrik.seipp@liu.se, Linköping University, Linköping, Sweden.

DOI: 10.1613/jair.1.19437

useful plans. Paths that are bound to contain redundant actions can also be pruned dynamically during the search (Karpas and Domshlak 2011).

The existence of redundant actions in plans generated by modern planners is not limited to top-k settings, and it has been documented widely (Balyo, Chrpa, et al. 2014; Chrpa et al. 2012a,b; Med and Chrpa 2022; Nakhost and Müller 2010). For example, in the agile setting, where the goal is to find a plan as fast as possible, redundant actions in plans are quite common. We will analyze this in detail in Section 7.1, but highlight some extreme cases here already: 50% of the actions in plans found by the Freelunch Madagascar planner (Balyo and Gocht 2018) for the DataNetwork domain are redundant. Similarly, 55% of actions are redundant in plans found by YASHP3 (Vidal 2014) for the Hiking domain. These examples show that in some settings it is crucial to identify redundant actions in plans. However, checking whether a plan is perfectly justified is NP-complete (Fink and Yang 1992; Nakhost and Müller 2010), so it is important to develop efficient methods.

In our work, we filter redundant actions from plans in a post-planning step while preserving the order of the remaining actions, in the same spirit as previous work (Balyo, Chrpa, et al. 2014; Chrpa et al. 2012a,b; Fink and Yang 1992; Med and Chrpa 2022; Nakhost and Müller 2010). Specifically, we focus on obtaining perfectly justified plans of minimum cost. These plans are referred to as *minimal reductions* of the original plan (Nakhost and Müller 2010). To our knowledge, there is only one previous approach for finding minimal plan reductions: Balyo, Chrpa, et al. 2014 cast the problem as a weighted partial MaxSAT formula. While eliminating redundant actions reduces the plan length and often its cost, our main motivation is to find justified plans. In contrast, optimizing plan length or cost is central for post-planning *plan optimization*, which usually involves modifying the plan actions and/or the action order (Muise et al. 2016; Olz and Bercher 2019; Say et al. 2016; Siddiqui and Haslum 2015; Waters et al. 2020). Bercher et al. (2024) present a comprehensive survey on plan optimization, including some methods focused in removing redundant actions from plans.

Given a plan for a planning task, we propose several automatic reformulations to a new planning task whose optimal solution is a minimal reduction of the original plan. Automated planning is PSPACE-complete even in its simplest form (Bylander 1994), but there are several reasons that justify its use to find minimal plan reductions: (1) the planning formulation is *natural*, since choosing which actions are needed to go from an initial state to a goal state is *planning*; (2) it is *simple*, in contrast to the previous weighted MaxSAT approach; (3) in many cases it is also more *efficient*; and (4) it is *portable* to other planning settings where the solution is a sequence of actions, such as temporal or conformant planning (Bonet and Geffner 2000).

This article extends a conference publication (Salerno et al. 2023a). In that paper, we presented the first classical planning formulation to find minimal plan reductions. Here, we make the following novel contributions:

- We thoroughly review existing work on eliminating redundant actions from plans.
- We study, both theoretically and empirically, four different alternative formulations of the minimal reduction problem as a classical planning task, including the one from the conference paper, which is explained in more detail here including an analysis of its theoretical properties.
- We analyze *fix-point plan action landmarks* theoretically (Salerno et al. 2023a).
- We introduce the concept of *always redundant* actions, characterizing actions that can never be part of a minimal reduction of a plan.
- We identify situations where the use of planning is preferable to the weighted MaxSAT approach by Balyo, Chrpa, et al. 2014, and vice versa.
- We carry out a comprehensive evaluation, aimed at
  - empirically comparing the different classical planning-based approaches among themselves and with the weighted MaxSAT approach; and
  - obtaining empirical evidence for the identified situations where a planning approach is preferable to weighted MaxSAT.

The rest of the article is organized as follows: Section 2 introduces background on classical planning and plan justifications. Section 3 summarizes related work. Section 4 presents four planning compilations and their theoretical properties. Sections 5 and 6 introduce plan action landmarks and always unnecessary actions, respectively. Finally, Section 7 contains the empirical evaluation, before Section 8 concludes the paper.

2 Background

In this section we describe the planning formalism we use throughout this work, as well as the concepts of plan justification and redundant actions. We consider classical planning tasks in the SAS$^+$formalism with action costs (Bäckström and Nebel 1995).

Definition 1 (**planning task**). *A planning task is a tuple* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$*, where:*

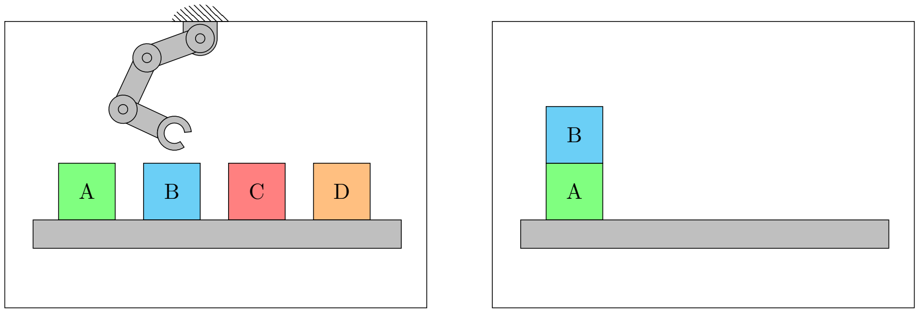
- $\mathcal{V}$ *is a set of finite-domain* state variables v*, each with a finite* domain $\mathcal{D}(v)$*. Any pair* $\langle v, d \rangle$ *such that* $v \in V$ *and* $d \in \mathcal{D}(v)$ *is a* fact*, written as* $v \mapsto d$*. For Boolean variables* v*, we refer to fact* $v \mapsto \top$ *as* v*, and* $v \mapsto \bot$ *as* $\neg v$*. A* partial state s *is a mapping of a subset of variables* $vars(s) \subseteq \mathcal{V}$ *to values in their respective domains, and we often treat partial states as sets of facts. The value of variable* $v \in vars(s)$ *in partial state* s *is denoted as* $s[v] \in \mathcal{D}(v)$*. Partial states that assign values to all variables (*$vars(s) = \mathcal{V}$*) are called* states*. The set of all states in the planning task is denoted as* $S(\Pi)$*.*
- $\mathcal{A}$ *is a finite set of actions. Each* action $a \in \mathcal{A}$ *is a pair* $\langle pre(a), eff(a) \rangle$*, where* $pre(a)$ *and* $eff(a)$ *are both partial states defining the* precondition *and the* effect *of* a*, respectively. An action* $a \in \mathcal{A}$ *is applicable in state* s *iff* $pre(a) \subseteq s$*. Applying action* a *in* s *yields the successor state* $s[\![a]\!]$*, with* $s[\![a]\!][v] = eff(a)[v]$ *if* $v \in vars(eff(a))$ *and as* $s[\![a]\!][v] = s[v]$ *otherwise. Each action* $a \in \mathcal{A}$ *has a non-negative cost* $c(a) \in \mathbb{R}^+_0$*.*
- $\mathcal{I}$ *is the* initial state.
- $\mathcal{G}$ *is a partial state describing the* goal condition.

A *solution* or *plan* for Π is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that, when applied in succession starting from the initial state, induces a state sequence $\mathcal{S}_\pi = \langle s_0, \ldots, s_n \rangle$ such that $s_0 = \mathcal{I}$, $\mathcal{G} \subseteq s_n$, and for each i with $1 \leq i \leq n$, a_i is applicable in s_{i-1}, and $s_i = s_{i-1}[\![a_i]\!]$. We denote the *length* of plan $\pi = \langle a_1, \ldots, a_n \rangle$ as $|\pi| = n$. The *cost* of a plan π is $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A plan is *optimal* if there is no cheaper plan. We slightly abuse notation and let $a_i \in \pi$ denote that action a_i occurs at position i of π.

Example 1. A well-known domain in automated planning is *Blocksworld* (Slaney and Thiébaux 2001). In this domain, a set of blocks can be either on a table or stacked on top of each other. A mechanical arm can hold one block at a time, and it can place the block it is holding on the table or stack it on top of another block. The arm can only pick up blocks that do not have another block on top of them. We will use the Blocksworld task in Figure 1 as a running example. Figure 1a shows the initial state, where four different blocks are on the table, and Figure 1b shows the goal description, where block A is on the table, while B is on top of A. Figure 2 presents a SAS$^+$representation of the task.

It is easy to see that the single optimal plan for this planning task is: $\pi_1 = \langle$*pick-up-b*, *stack-b-a*$\rangle$. However, an equally valid plan for this planning task is: $\pi_2 = \langle$*pick-up-c*, *stack-c-d*, *pick-up-b*, *stack-b-a*$\rangle$. Since π_1 is a subsequence of π_2 and both achieve the goals, it is trivial for a human to realize that the actions *pick-up-c*, *stack-c-d* can be removed from the plan and still reach the goals. In other words, they are not *justified.*

The notion of action and plan justifications can be traced back to the early 1990s. Fink and Yang 1992 define three types of plan justification: *backwards* justification, *well*-justification and *perfect* justification. Backwards justification is the weakest of the three, and it is defined using the notion of causal links between actions in a plan (McAllester and Rosenblitt 1991). In essence, there is a causal link between two actions a_i and a_j, if a_i produces a fact f that is a precondition of a_j, and no action in between them overwrites that fact. The set of causal links of a plan can be extracted in polynomial time (Jiménez Celorrio et al. 2013).



(a) Initial state (b) Goal description

Fig. 1. Example Blocksworld planning task.

Definition 2 (**causal link**). *Given a planning task* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *and a plan* $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ *for* Π, *there exists a* causal link $\ell = \langle a_i, f, a_j \rangle$ *between actions* a_i *and* a_j $(1 \leq i < j \leq n)$ *if*

- *there is a fact* $f = v \mapsto d \in \mathit{eff}(a_i) \cap \mathit{pre}(a_j)$, *and*
- *there is no action* a_k *with* $i < k < j$, *such that* $v \in \mathit{vars}(\mathit{eff}(a_k))$.

An action is backward justified if it is causally related to a goal fact: there is a chain of causal links that connects the action to a goal fact.

Definition 3 (**backward justified action**). *Given a planning task* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *and a plan* $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ *for* Π, *an action* $a_i \in \pi$ *is* backward-justified *if at least one of the following conditions is true:*

- *Action* a_i *achieves a goal fact that is not achieved by any action after* a_i. *Formally, there is a fact* $f \in \mathit{eff}(a_i) \cap \mathcal{G}$ *such that there is no action* $a_j \in \pi$ *with* $j > i$ *and* $f \in \mathit{eff}(a_j)$.
- *There exists a causal link* $\ell = \langle a_i, f, a_j \rangle$ *and* a_j *is backward justified.*

Backward justification fails to capture if an action can be removed from the plan without invalidating it, because it only considers the last achiever of a fact before an action needs it. In other words, it could be the case that an action that can be removed from the plan is backward justified. The next type of justification is *well-justification*, which captures if a single action can be removed from the plan without invalidating it:

Definition 4 (**well-justified action**). *Given a planning task* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *and a plan* $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ *for* Π, *an action* $a_i \in \pi$ *is* well-justified *if and only if its removal from the plan invalidates* π.

A plan is backward/well justified if all of its actions are backward/well justified. The action sequence resulting from removing any number of actions from a plan (including no actions) is a *subsequence* of the original plan. When the resulting subsequence is also a plan for the planning task, we call it a *plan reduction*.

Definition 5 (**plan reduction**). *Let* π *be a plan for a planning task* Π *and* ρ *be a subsequence of* π. *Then* ρ *is a* plan reduction *of* π *if and only if* ρ *is also a plan for* Π.

Throughout the article, when it is necessary to relate a specific subsequence of actions of length m with the sequence (plan) of length n, $m \leq n$, from which it originates, we use the notation $\langle a_{f(1)}, \ldots, a_{f(m)} \rangle$, where f is a strictly monotonic function mapping the action indices in the subsequence to action indices in the sequence. For instance, if the plan is $\langle a_1, a_2, a_3, a_4, a_5 \rangle$, the subsequence containing the first and third actions is $\langle a_{f(1)}, a_{f(2)} \rangle$ with $f(1) = 1$ and $f(2) = 3$.

Variables

$\mathcal{V} = \{arm, a\text{-}pos, b\text{-}pos, c\text{-}pos, d\text{-}pos, a\text{-}top, b\text{-}top, c\text{-}top, d\text{-}top\}$

Domains

$\mathcal{D}(arm) = \{free, full\}$
$\mathcal{D}(a\text{-}pos) = \{arm, b, c, d, table\}$
$\mathcal{D}(a\text{-}top) = \{clear, blocked\}$
...

Actions

$\mathcal{A} = \{$pick-up-a, put-down-a, stack-a-b, unstack-a-b, ...$\}$

pick-up-a $= \langle \{a\text{-}pos \mapsto table, a\text{-}top \mapsto clear, arm \mapsto free\}, \{a\text{-}pos \mapsto arm, arm \mapsto full\}\rangle$
put-down-a $= \langle \{a\text{-}pos \mapsto arm\}, \{a\text{-}pos \mapsto table, arm \mapsto free\}\rangle$
stack-a-b $= \langle \{a\text{-}pos \mapsto arm, b\text{-}top \mapsto clear\}, \{a\text{-}pos \mapsto b, b\text{-}top \mapsto blocked, arm \mapsto free\}\rangle$
unstack-a-b $= \langle \{a\text{-}pos \mapsto b, a\text{-}top \mapsto clear, arm \mapsto free\}, \{a\text{-}pos \mapsto arm, b\text{-}top \mapsto clear, arm \mapsto full\}\rangle$
...

Initial state

$\mathcal{I} = \{ a\text{-}pos \mapsto table, b\text{-}pos \mapsto table, c\text{-}pos \mapsto table, d\text{-}pos \mapsto table, a\text{-}top \mapsto clear, b\text{-}top \mapsto clear, c\text{-}top \mapsto clear, d\text{-}top \mapsto clear, arm \mapsto free\}$

Goal state

$\mathcal{G} = \{ a\text{-}pos \mapsto table, b\text{-}pos \mapsto a, b\text{-}top \mapsto clear\}$

Fig. 2. SAS$^+$representation of the Blocksworld task in Figure 1. For each block, there is a variable representing its position, which can be on the table, on top of another block, or being held by the mechanical arm. Another variable represents whether a block has a block on top of it. The last variable represents if the arm is holding a block or if it is free. For each block, there is one action to pick it up from the table, and one to put it down on the table. Finally, for each pair of blocks, there are actions to either stack or unstack them.

A plan is well-justified if all of its actions are well-justified, which means that a plan π is well-justified if and only if there does not exist a plan reduction ρ such that $|\rho| = |\pi| - 1$ (i.e. no action can be removed individually without invalidating the plan). Going back to our running example, the plan ⟨*pick-up-b*, *stack-b-a*, *pick-up-c*⟩ is *not* well-justified, since action *pick-up-c* can be removed from the plan without affecting its validity. However, the plan ⟨*pick-up-c*, *stack-c-d*, *pick-up-b*, *stack-b-a*⟩ *is* well-justified: removing any action creates an invalid plan (either because some actions are not applicable or because the goals are not achieved). In order to identify this type of redundant actions, we need the strongest type of justification, which is the main interest of this work: *perfect justification*. In contrast to backward and well-justification, perfect justification is defined over plans instead of over actions.

Definition 6 (**Perfect justification**). *A plan π for planning task Π is perfectly justified if and only if there is no plan reduction ρ of π such that $|\rho| < |\pi|$.*

This means that a plan is perfectly justified if no non-empty *subset* of actions can be removed from it without invalidating the plan, so the plan ⟨*pick-up-c*, *stack-c-d*, *pick-up-b*, *stack-b-a*⟩ is *not* perfectly justified.

The task of finding a well-justified plan reduction of a given plan can be solved in polynomial time, while verifying if a plan is perfectly justified, as well as the problem of finding a perfectly justified plan reduction, are NP-complete (Fink and Yang 1992; Nakhost and Müller 2010). Finally, the task of finding the *cheapest* perfectly justified plan reduction of a given plan is known as the minimal reduction problem (Balyo, Chrpa, et al. 2014; Nakhost and Müller 2010).

3 Related Work on Action Elimination

Over the years, several methods have been proposed to identify and remove redundant actions from plans. We group them into two coarse categories: optimal action elimination methods, and sub-optimal (greedy) action elimination methods. Optimal action elimination methods produce a minimal reduction, while greedy action elimination methods can quickly remove redundant actions, but have no guarantees that the resulting plan reduction is minimal. The main focus of this work are optimal action elimination methods.

3.1 Greedy Methods

Fink and Yang 1992 present, to the best of our knowledge, the first three methods to identify and eliminate redundant actions from plans.

(1) The first method, which we will call FY1, guarantees producing backward justified plans. FY1 checks, for each action in a plan, if there exists a causal link chain rooting from the action to a goal fact. Any action that is not causally linked to a goal is removed, ensuring that the resulting plan is backward justified.
(2) The second method, FY2, yields well-justified plans. FY2 checks, for each action a in the plan, if the sequence resulting from removing a is still a plan. If so, a is removed. This loop is repeated until no actions can be removed from the plan, producing a well-justified plan.
(3) The final method, FY3, greedily tries to remove each action from the plan. If any of the remaining actions are not applicable afterwards, they are also removed. If the resulting sequence is not a plan, FY3 restores the actions removed in this step. Otherwise, if the resulting sequence is a plan, the method continues with the same greedy process for each remaining action. This process guarantees finding well-justified plans, and it was actually rediscovered by Nakhost and Müller 2010.

Chrpa et al.Chrpa et al. 2012a,b study redundant actions from the view of action dependencies. If two actions in a plan are *inverse* (applying the second after the first has the same effect as not applying any of them), and no action in between them depends on the first action, then they can both be removed from the plan.

Balyo, Chrpa, et al. 2014 build on FY3, taking into account action costs when deciding which set of redundant actions to remove from the plan. The original method removes sets of redundant actions as soon as it discovers them, but their algorithm is a bit less greedy, by identifying multiple sets of redundant actions, and eliminating the one with the highest cost. Med and Chrpa 2022 further improve this greedy action elimination method by introducing *Plan Action Landmarks* (PALs) and action cycles.

We will explain plan action landmarks in detail in Section 5, but put simply, a plan action landmark is an action that must be a part of any plan reduction (i.e., it is never redundant). Action cycles extend the notion of inverse actions from pairs of actions to sequences, identifying (not necessarily consecutive) subsequences of actions in a plan that, if removed, lead to the same state after the execution of the last action. Naively, one could try to preprocess a plan by identifying loops or action cycles in the plan and removing them. However, while the resulting sequence (without the loop) will be a plan, this cannot be done if the purpose is to find a minimal reduction. We illustrate this with a simple example: a task with three Boolean variables v_1, v_2, v_3. The initial state is $\mathcal{I} = \{\neg v_1, \neg v_2, \neg v_3\}$ and the goal is $\mathcal{G} = \{v_1, v_2, v_3\}$. Consider the plan $\pi = \langle a_1, a_2, a_3, a_4, a_5 \rangle$, where:

- $a_1 = \langle \{\neg v_1, \neg v_2\}, \{v_1, v_2\} \rangle$
- $a_2 = \langle \{v_1, v_2\}, \{\neg v_1, \neg v_2\} \rangle$

- $a_3 = \langle\{\neg v_1\}, \{v_1\}\rangle$
- $a_4 = \langle\{\neg v_3\}, \{v_3\}\rangle$
- $a_5 = \langle\{\neg v_2\}, \{v_2\}\rangle$

After applying a_1 and a_2, we end up in the initial state again. So removing all action cycles from π yields plan $\pi' = \langle a_3, a_4, a_5\rangle$. Assuming unit costs for all actions, the minimal reduction of π' is $\langle a_3, a_4, a_5\rangle$, since no action is redundant. In contrast, the minimal reduction of π is $\langle a_1, a_4\rangle$.

3.2 Optimal Methods

To the best of our knowledge, the only previous optimal method to solve the minimal reduction problem was proposed by Balyo, Chrpa, et al. 2014. Given a planning task Π and a plan $\pi = \langle a_1, \ldots, a_n\rangle$ for Π, they define a CNF formula $F_{\Pi,\pi}$, such that each satisfying assignment represents a plan reduction of π. By solving $F_{\Pi,\pi}$ with a weighted partial MaxSAT solver, they find a plan reduction π' of minimal cost. Then, by solving a new $F_{\Pi,\pi'}$ created from this reduction using unit cost clauses, they guarantee there are no zero-cost redundant actions in the plan reduction, finding a minimal reduction. We include a full description of the formulation of $F_{\Pi,\pi}$ in Appendix A.

3.3 Plan Justification Beyond Classical Planning

Muise et al. 2016 proposed a partial weighted MaxSAT encoding that is able to identify redundant actions in plans, as well as produce partial order plans from linear plans. A partially ordered plan, instead of imposing a total order over the actions, specifies a set of ordering constraints (action x is before action y). Their encoding is able to find a *minimum cost least commitment partial order plan*, that, given an input partially ordered plan, finds a partially ordered plan with the cheapest cost that has the least possible ordering constraints over the actions. Solving this problem can potentially generate cheaper and more flexible plans compared to finding minimal reductions, as it also considers re-ordering actions as well as partially ordered plans. However, while related to our work, re-ordering and de-ordering are distinct research questions with different aims and challenges. Finding a minimal-cost reordering can be exponentially harder than finding minimal reductions in practice, as it requires considering all possible permutations of subsets of actions, whereas minimal reduction only considers subsets. The computational difficulty not only differs significantly in theory, but it can also be observed in practice: Muise et al. 2016 consider re-orderings and mention that plans with more than 200 actions are problematic for their method, since their encoding is too big to fit in memory. In contrast, our methods handle input plans with more than 3000 steps within a few seconds.

More recently, Sreedharan et al. 2023 proposed a generalization of plan justifications to policies, in the context of fully observable, non-deterministic planning (FOND). The work focuses on generalizing the notion of well-justified actions. An action is well-justified in a policy if it is needed in every possible trace from the initial state to a goal state. They propose a planning compilation to identify when an action is well-justified in this context. When this new planning task is unsolvable, the action in question is well-justified.

4 Minimal Reduction as Planning

In this section, we introduce four planning formulations to solve the minimal reduction problem. First, we describe the design space within which we create the formulations, motivating each option and analyzing their trade-offs. We then describe each formulation and show that an optimal solution to the resulting planning tasks is a minimal reduction of the input plan. Finally, we compare the theoretical properties of each formulation, which will help to interpret the empirical results shown in Section 7.

4.1 Design Space

Given a planning task Π and a plan π that solves Π, the minimal reduction problem asks for a least-cost, perfectly justified reduction π' of π. To solve this minimal reduction problem with a classical planner, we must devise a new planning task Π' that allows applying or skipping the actions of the original plan π, while preserving their relative order, and eventually achieving the goal of Π. The simplest task variant that guarantees this is a formulation where each action in the plan is either applied or skipped in order. This implies that plans for the new task have the same length as the input plan. However, when many consecutive actions π are redundant, it might be more efficient to skip multiple actions at a time, allowing us to find solutions at shallower depths of the search tree. Similarly, for plans with little redundancy, applying multiple actions at a time might improve search performance. With these idea in mind, we propose four different planning compilations:

- Π^{s1}_{a1} ("skip one, apply one"): each action of the plan is either applied or skipped individually.[1]
- Π^{sm}_{a1} ("skip multiple, apply one"): each action of the plan can be applied individually, but multiple actions can be skipped at a time.
- Π^{s1}_{am} ("skip one, apply multiple"): multiple actions can be applied at a time, but actions are skipped individually.
- Π^{sm}_{am} ("skip multiple, apply multiple"): multiple actions can be applied and skipped simultaneously.

4.2 Π^{s1}_{a1} Compilation: Skipping and Applying Single Actions

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ be a plan for Π. We define R_π as the set of facts that appear in a precondition of the actions in the plan π or in the goal: $R_\pi = \bigcup_{a_i \in \pi} pre(a_i) \cup \mathcal{G}$. R_π represents the set of *relevant facts* for π, or the facts *read* by π. We also define W_π as the facts that appear in the effects of the actions or the initial state: $W_\pi = \bigcup_{a_i \in \pi} eff(a_i) \cup \mathcal{I}$. W_π are the facts *written* by π. We refer to the set of all facts in π as $F_\pi = R_\pi \cup W_\pi$. With this, we define the function τ, that given a fact $v \mapsto d$, maps the fact to itself if $v \mapsto d$ is read by π and to the irrelevant fact $v \mapsto \theta$ otherwise.

$$\tau(v \mapsto d) = \begin{cases} v \mapsto d & \text{if } v \mapsto d \in R_\pi \\ v \mapsto \theta & \text{otherwise.} \end{cases}$$

Then, the new planning task $\Pi^{s1}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ has facts $F'_\pi = \{\tau(v \mapsto d) \mid v \mapsto d \in F_\pi\}$, and is defined as:

- $\mathcal{V}' = \{v' \mid v \in \mathcal{V} \wedge |\mathcal{D}(v')| > 1\} \cup \{pos\}$, where $\mathcal{D}(v') = \{d \mid v \mapsto d \in F'_\pi\}$; and variable *pos* with $\mathcal{D}(pos) = \{0, \ldots, n\}$ tracks the current position in the original plan. Note that $\mathcal{D}(v')$ only contains θ if there is a fact $v \mapsto d \in W_\pi \setminus R_\pi$.[2]
- $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\} \cup \{skip_i \mid 1 \leq i \leq n\}$, so that for each original plan action a_i, there is one *apply* action a'_i and one *skip* action $skip_i$. Actions a'_i are defined as:

$$\begin{aligned} pre(a'_i) &= pre(a_i) \cup \{pos \mapsto i-1\} \\ eff(a'_i) &= \{\tau(v \mapsto d) \mid v \mapsto d \in eff(a_i) \wedge v \in \mathcal{V}'\} \cup \{pos \mapsto i\} \end{aligned}$$

  where the new action has the same preconditions as the original one, plus an additional precondition to ensure that the current position is the preceding one. The effects are the same as the original action, but mapped with function τ to only keep relevant values, plus an additional effect to update the current position.

[1]This compilation was first presented in our original conference paper (Salerno et al. 2023a).
[2]Just removing effects that map a variable v to θ is not enough. Even though $v \mapsto \theta$ is never read itself, it can affect the applicability of a later action that has $v \mapsto d$, with $d \neq \theta$ in its preconditions.

The $skip_i$ actions just increase the value of pos from $i-1$ to i:

$$\begin{aligned} pre(skip_i) &= \{pos \mapsto i-1\} \\ e\!f\!f(skip_i) &= \{pos \mapsto i\} \end{aligned}$$

The a'_i actions maintain the cost $c(a_i)$ of the corresponding a_i, while the $skip_i$ actions have zero-cost.[3]

- $\mathcal{I}' = \{\tau(v \mapsto d) \mid v \mapsto d \in \mathcal{I} \wedge v \in \mathcal{V}'\} \cup \{pos \mapsto 0\}$ is composed of two sets of facts: (1) the facts from the original initial state that are relevant for the plan, i.e., those that belong to $(\mathcal{I} \cap R_\pi)$ and new facts of the form $v \mapsto \theta$, setting the variables in $\mathcal{V}'$ with an irrelevant initial value to θ; and (2) a set with just one fact initializing pos variable to zero.
- $\mathcal{G}' = \mathcal{G} \cup \{pos \mapsto n\}$ contains the original goals and requires the pos variable to be at the end of the original plan (the latter could be omitted, but it can be useful for heuristics).

With this definition of the task, plans for Π^{s1}_{a1} can only contain *skip* actions (with a corresponding skipped action in the original plan) and actions from the original plan in the same order (if they appear in the plan at position i, they are only applicable when pos is $i-1$).

Proposition 1. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task, $\pi = \langle a_1, \ldots, a_n\rangle$ be a plan for Π, $\Pi^{s1}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}'\rangle$ be the reformulated planning task and $\pi^{s1}_{a1} = \langle b_1, \ldots, b_m\rangle$ be a plan for Π^{s1}_{a1}. Then, there is a one-to-one correspondence between the actions in π and the actions in π^{s1}_{a1} such that $n = m$ and $a_i \cong b_i$, where b_i is either a'_i (the reformulation of a_i) or $skip_i$.*

Proof. This follows directly from the construction of Π^{s1}_{a1}. Every $a'_i \in \mathcal{A}'$ maintains the precondition and relevant effects of $a_i \in \pi$. Also, both a'_i and $skip_i$ actions are only applicable if pos is $i-1$, and they set pos to i. Since pos is 0 in the initial state and n in all goal states, all plans for Π^{s1}_{a1} select either a'_i or $skip_i$ at each of the n steps, producing a plan of length $m = n$. □

Given Proposition 1, it is trivial to translate a plan π^{s1}_{a1} for Π^{s1}_{a1} into a subsequence of the original plan π for Π.

Definition 7 (**translated plan**). *Let $\pi = \langle a_1, a_2, \ldots, a_n\rangle$ be a plan for Π, $\pi^{s1}_{a1} = \langle b_1, b_2, \ldots, b_n\rangle$ be a plan for Π^{s1}_{a1}, and m, $m \leq n$, the number of actions in π^{s1}_{a1} that are not skip actions. The translated plan is the subsequence $\pi' = \langle a_{f(1)}, \ldots, a_{f(m)}\rangle$ of π, containing exactly those actions $a_{f(k)} \in \pi$ for which the corresponding $b_{f(k)} \in \pi^{s1}_{a1}$ is not a skip action.*

We now prove that a solution for Π^{s1}_{a1} translated by Definition 7 is guaranteed to be a plan reduction of the original plan. We also show that, since the compilation **only** allows for the application of actions in the original plan or for their explicit elimination, the set of all plans that solve Π^{s1}_{a1} translated by Definition 7 is exactly the set of all plan reductions of the original plan. Formally:

Proposition 2. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task, π be a plan for Π, $\Pi^{s1}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}'\rangle$ be the reformulated planning task, and P^{s1}_{a1} be the set of all plans for Π^{s1}_{a1} translated by Definition 7. Then P^{s1}_{a1} is exactly the set of all plan reductions of π for Π.*

Proof. We have to show that (i) any plan $\pi' \in P^{s1}_{a1}$ is a plan reduction of π, and that (ii) any plan reduction of π for Π belongs to P^{s1}_{a1}.
(i) Let π' be a plan in P^{s1}_{a1}, obtained with Definition 7 from original plan π and plan π^{s1}_{a1} for Π^{s1}_{a1}. By Definition 7, $\pi' = \langle a_{f(1)}, \ldots, a_{f(m)}\rangle$ is a subsequence of π. The initial state $\mathcal{I}'$ of Π^{s1}_{a1} contains all facts from $\mathcal{I}$ that are relevant for π. Every $a'_i \in \mathcal{A}'$ maintains the precondition of its corresponding action $a_i \in \pi$. Actions $skip_i$ in π^{s1}_{a1}, omitted from π to generate π' by Definition 7, only modify the pos variable and do not contribute to the achievement of other facts in action preconditions or goals. Then $a_{f(1)}$ is applicable in $\mathcal{I}$, and every action $a_{f(i)} \in \pi'$ is applicable

[3]Below, we elaborate on how to handle problems with zero-cost actions.

in the state resulting from the application of the previous action $a_{f(i-1)} \in \pi'$. The goal description $\mathcal{G}'$ of Π^{s1}_{a1} contains the goals $\mathcal{G}$ of Π. Thus, π' achieves the goals of Π from initial state $\mathcal{I}$, and therefore it is a plan for Π. Since π' is a subsequence of π, by Definition 5 π' is a plan reduction of π.
(ii) Let ρ be a plan reduction of π. By Definition 5, ρ is a subsequence of π, so there exists a strictly monotonic function f mapping its action indices into action indices in π. The image of f, $img(f)$, is the set of indices of the actions in π included in ρ. Assume a plan π^{s1}_{a1} is generated from π by replacing every action $a_i \in \pi$, where $i \in img(f)$, by the corresponding reformulated action $a'_i \in \mathcal{A}'$; and every $a_i \in \pi$, where $i \notin img(f)$ by $skip_i$. Since ρ is a plan for Π, π^{s1}_{a1} is a plan for P^{s1}_{a1}. Then, the plan reduction ρ can be obtained from π and π^{s1}_{a1} by Definition 7 and therefore $\rho \in P^{s1}_{a1}$. □

Now that we have shown that all plan reductions of π for Π can be generated from Π^{s1}_{a1}, it is easy to see that, if there are no zero-cost actions in Π, an optimal solution for Π^{s1}_{a1} is a minimal reduction of π.[4]

THEOREM 1. *Let π be a plan for a planning task Π without zero-cost actions, and π^{s1}_{a1} an optimal plan for the task Π^{s1}_{a1} generated for Π and π. The plan π' obtained from π^{s1}_{a1} using Definition 7 is a minimal reduction of π.*

Proof. To prove that π' is a minimal reduction of π, we have to show that (i) π' is a plan reduction of minimal cost and (ii) that π' is perfectly justified. Proposition 1 proves (i), since π^{s1}_{a1} is a plan for Π^{s1}_{a1} if and only if the translated plan π' is a plan reduction of π. Hence, since action costs in Π and Π^{s1}_{a1} are the same, except for skip actions which have zero-cost, and π^{s1}_{a1} is a plan of minimal cost for Π^{s1}_{a1}, π' is a plan reduction of minimal cost of π. Furthermore, from the fact that there are no zero-cost actions in Π, π' cannot contain redundant actions: if it did, there would exist a plan reduction of lesser cost, implying that π^{s1}_{a1} is not an optimal plan for Π^{s1}_{a1}, which is assumed. By contradiction, it follows that π' must be perfectly justified, proving (ii). Since π' is both a plan reduction of minimal cost and perfectly justified, it is a minimal reduction of π. □

Zero-cost Actions. In the presence of zero-cost actions in Π, further steps must be taken to guarantee that an optimal solution for Π^{s1}_{a1} is also perfectly justified. One option, in a similar fashion as the WPMaxSAT approach (Balyo, Chrpa, et al. 2014), is to create a new Π^{s1}_{a1} task, taking as input a plan reduction of minimal cost, but assigning cost 1 to all non-skip actions. However, this implies solving two different planning tasks, which can be time-consuming. To avoid this, we adapt the original costs of the input plan actions by setting the cost of all zero-cost actions to 1, and multiplying all other costs by the factor $f = \lceil \frac{m}{mincost} + \epsilon \rceil$, where m is the number of zero-cost actions in the input plan, *mincost* is the smallest positive action cost in the plan, and ϵ is an arbitrarily small positive real number. If all actions have zero-cost, f is undefined but also unneeded. This factor satisfies $m < f \cdot mincost$, which guarantees that removing any action with an original cost greater than zero will be more beneficial than removing any set of actions that originally cost zero. So, with this modification, an optimal plan for Π^{s1}_{a1} is a minimal reduction for π.

Example 2. To better illustrate what a Π^{s1}_{a1} task looks like, we return to the running example from Figure 1. Figure 3 shows the Π^{s1}_{a1} task resulting from the input plan ⟨*pick-up-c*, *stack-c-d*, *pick-up-b*, *stack-b-a*⟩. The domain of each variable is mapped as explained, only keeping facts in R_π (those that are relevant for the new planning task). Variables *a-pos* and *d-pos* can be removed from the task, since their domains only have one value. We keep them in the example to explicitly illustrate cases where variables would disappear. Note that the number of actions in Π and Π^{s1}_{a1} differ. The reformulated task has exactly 8 actions, while the original task had 20 (4 *pick-up*, 4 *put-down*, 12 *stack*). This discrepancy stems from the fact that the number of actions in Π^{s1}_{a1} depends on the number of actions in the plan π, not the actions in the task. If $|\pi| = n$, then $|\mathcal{A}'| = n * 2$.

The Π^{s1}_{a1} formulation allows for the explicit application or elimination (skipping) of each action in the input plan. A Π^{s1}_{a1} task induces a state space in the form of a tree, rooted in the initial state. It has a branching factor of

[4]We leave zero-cost actions out of the proof for clarity, but we show how to handle them immediately afterwards.

Variables

$\mathcal{V} = \{arm, a\text{-}pos, b\text{-}pos, c\text{-}pos, d\text{-}pos, a\text{-}top, b\text{-}top, c\text{-}top, d\text{-}top, \mathbf{pos}\}$

Domains

$\mathcal{D}(arm) = \{free, \theta\}$	$\mathcal{D}(a\text{-}pos) = \{\theta\}$
$\mathcal{D}(a\text{-}top) = \{clear, \theta\}$	$\mathcal{D}(b\text{-}pos) = \{table, arm, a\}$
$\mathcal{D}(b\text{-}top) = \{clear, \theta\}$	$\mathcal{D}(c\text{-}pos) = \{table, arm, \theta\}$
$\mathcal{D}(c\text{-}top) = \{clear, \theta\}$	$\mathcal{D}(d\text{-}pos) = \{\theta\}$
$\mathcal{D}(d\text{-}top) = \{clear, \theta\}$	$\mathcal{D}(\boldsymbol{pos}) = \mathbf{\{0,1,2,3,4\}}$

Actions

$\mathcal{A}' = \{$pick-up-c$'$, skip-pick-up-c, stack-c-d$'$, skip-stack-c-d, pick-up-b$'$, skip-pick-up-b, stack-b-a$'$, skip-stack-b-a$\}$

pick-up-c$'$ $= \langle pre(\text{pick-up-c}) \cup \{pos \mapsto 0\}, eff(\text{pick-up-c}) \cup \{pos \mapsto 1\}\rangle$
skip-pick-up-c $= \langle \{pos \mapsto 0\}, \{pos \mapsto 1\}\rangle$

stack-c-d$'$ $= \langle pre(\text{stack-c-d}) \cup \{pos \mapsto 1\}, eff(\text{stack-c-d}) \cup \{pos \mapsto 2\}\rangle$
skip-stack-c-d $= \langle \{pos \mapsto 1\}, \{pos \mapsto 2\}\rangle$

pick-up-b$'$ $= \langle pre(\text{pick-up-b}) \cup \{pos \mapsto 2\}, eff(\text{pick-up-b}) \cup \{pos \mapsto 3\}\rangle$
skip-pick-up-b $= \langle \{pos \mapsto 2\}, \{pos \mapsto 3\}\rangle$

stack-b-a$'$ $= \langle pre(\text{stack-b-a}) \cup \{pos \mapsto 3\}, eff(\text{stack-b-a}) \cup \{pos \mapsto 4\}\rangle$
skip-stack-b-a $= \langle \{pos \mapsto 3\}, \{pos \mapsto 4\}\rangle$

Initial state

$\mathcal{I}' = \{arm \mapsto free, a\text{-}pos \mapsto \theta, b\text{-}pos \mapsto table, c\text{-}pos \mapsto table, d\text{-}pos \mapsto \theta,$
$a\text{-}top \mapsto clear, b\text{-}top \mapsto clear, c\text{-}top \mapsto clear, d\text{-}top \mapsto clear, pos \mapsto 0\}$

Goal state

$\mathcal{G}' = \{b\text{-}pos \mapsto on\text{-}a, b\text{-}top \mapsto clear, pos \mapsto 4\}$

Fig. 3. SAS$^+$representation of the Π^{s1}_{a1} task generated from the Blocksworld task in Figure 1 and the plan ⟨*pick-up-c*, *stack-c-d*, *pick-up-b*, *stack-b-a*⟩.

at most 2, and each path from the root to a leaf has length n, where $n = |\pi|$ is the number of actions in the input plan. Since only leaf nodes can be goal states, all plans for Π^{s1}_{a1} tasks have length n as well. Figure 4 shows the full state space of the Π^{s1}_{a1} task for our running example.

4.3 Π^{sm}_{a1} Compilation: Skipping Multiple Actions at Once

Looking at the definition of a Π^{s1}_{a1} task, the naive way to allow skipping multiple actions at a time, is to introduce additional *skip* actions that change the *pos* variable to any index greater than its current value. For that, if $|\pi| = n$, the set of skip actions would be $SKIPS = \{skip_{ij} \mid 0 \leq i < j \leq n\}$, where $skip_{ij} = \langle \{pos \mapsto i\}, \{pos \mapsto j\}\rangle$. However, this would incur $|SKIPS| = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ skip actions. Since plans commonly have thousands of steps, avoiding this quadratic growth can be beneficial. For this, we now propose a compilation where multiple actions can be skipped without incurring this quadratic growth.

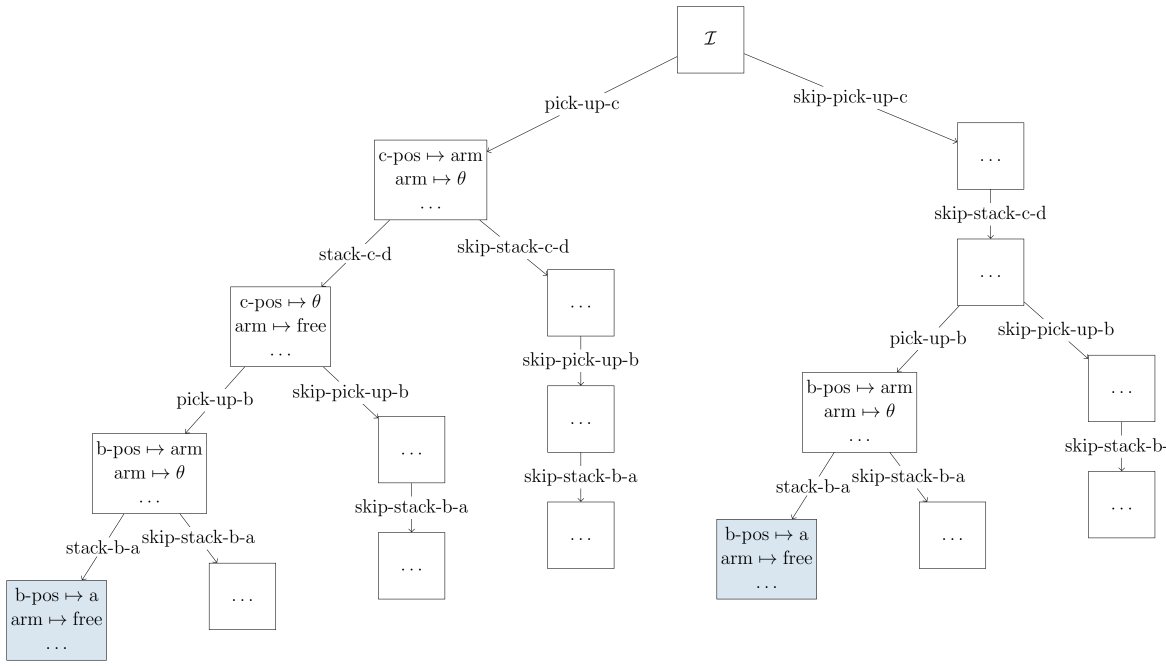


Fig. 4. Π^{s1}_{a1} task state space for the running example. In each node, only the relevant facts that differ from the parent node are shown. Goal states are marked gray. In the initial state all four blocks are on the table, and the mechanical arm is empty. Here, we can either apply the action *pick-up-c*, or skip it. If it is applied, in the resulting state the mechanical arm is holding block C. Otherwise, nothing changes (except for the *pos* variable, that we omit for simplicity). Continuing the explanation on the branch of the tree that skips the first action (right branch), the second action of the plan (*stack-c-d*) is not applicable, since the arm is not holding block C. In this case, the only applicable action is to skip this action. If we continue down this branch, we can *pick-up-b* and *stack-b-d*, finding a plan reduction that is a minimal reduction of the input plan. Its cost is $0 + 0 + 1 + 1 = 2$.

The idea of this new formulation is simple: when one action is applied, all preceding actions become inapplicable. For this, we simply need to introduce a new Boolean variable $active_i$ to the task for each action a_i in the input plan. Initially, all actions are active ($active_i$, for all i), and each action a_i has $active_i$ in its precondition. Applying action a_i makes the preceding actions and itself inapplicable by setting $\neg active_j$ for all $j \leq i$.

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan for Π. We formally define the planning task $\Pi^{sm}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ as follows:

- $\mathcal{V}' = \{v' \mid v \in \mathcal{V} \wedge |\mathcal{D}(v')| > 1\} \cup \{active_i \mid 1 \leq i \leq n\}$, where $\mathcal{D}(v') = \{d \mid v \mapsto d \in F'_\pi\}$, as for Π^{s1}_{a1}, and $\mathcal{D}(active_i) = \{\top, \bot\}$, for all $1 \leq i \leq n$.
- $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\}$, where each a'_i is defined as:

$$\begin{aligned} pre(a'_i) &= pre(a_i) \cup \{active_i\} \\ eff(a'_i) &= \{\tau(v \mapsto d) \mid v \mapsto d \in eff(a_i) \wedge v \in \mathcal{V}'\} \cup \{\neg active_j \mid 1 \leq j \leq i\} \end{aligned}$$

- $\mathcal{I}' = \{\tau(v \mapsto d) \mid v \mapsto d \in \mathcal{I} \wedge v \in \mathcal{V}'\} \cup \{active_i \mid 1 \leq i \leq n\}$
- $\mathcal{G}' = \mathcal{G}$

In a Π^{sm}_{a1} task, actions a_i in the original plan can be applied if their original preconditions $pre(a_i)$ are met and no action that comes after a_i in the plan has been applied. Thus, plans for Π^{sm}_{a1} can only contain actions from π maintaining their original order, implying that those plans are plan reductions of π. To translate Π^{sm}_{a1} plans into plans for Π, we simply consider for each action its corresponding action in Π. Since each action in Π^{sm}_{a1} deactivates itself and all preceding actions, its corresponding action index in the original plan is the highest index of the $active_i$ variables it sets to $\perp$ in its effects. Formally:

Definition 8 (**translated Π^{sm}_{a1} plan**). *Let $\pi = \langle a_1, a_2, \ldots, a_n\rangle$ be a plan for Π, $\pi^{sm}_{a1} = \langle b_1, b_2, \ldots, b_m\rangle$ be a plan for Π^{sm}_{a1}. The translated plan π', for each action $b \in \pi^{sm}_{a1}$, takes the corresponding action a from the original task.*

A plan for Π^{sm}_{a1} translated by Definition 8 is a plan reduction of the original plan. Additionally, the set of all plans that solve Π^{sm}_{a1} translated by Definition 8 is exactly the set of all plan reductions of the original plan.

Proposition 3. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task, π be a plan for Π, $\Pi^{sm}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}'\rangle$ be the reformulated planning task, and P^{sm}_{a1} be the set of all plans for Π^{sm}_{a1} translated by Definition 8. Then P^{sm}_{a1} is exactly the set of all plan reductions of π for Π.*

Proof. We will show that (i) any plan $\pi' \in P^{sm}_{a1}$ is a plan reduction of π for Π, and that (ii) any plan reduction of π for Π is in P^{sm}_{a1}.
(i) Let π' be a plan in Π^{sm}_{a1} translated from a plan π^{sm}_{a1} for Π^{sm}_{a1}. Since plans for Π^{sm}_{a1} maintain the order of the actions, π' is a subsequence of π. The initial state $\mathcal{I}'$ contains all relevant facts for π, and all actions in $\mathcal{A}'$ maintain all preconditions and relevant effects of their corresponding actions in π. Thus, since all actions in π^{sm}_{a1} are applicable in succession starting from $\mathcal{I}'$, all actions in π' must be applicable in succession starting from $\mathcal{I}$. Furthermore, since $\mathcal{G}' = \mathcal{G}$ and π^{sm}_{a1} is a plan for Π^{sm}_{a1}, π' is a plan for Π.
(ii) Let ρ be a plan reduction of π for Π. By translating actions in ρ following the Π^{sm}_{a1} compilation, we obtain the action sequence π^{sm}_{a1}. Since ρ is a plan reduction of π, all of its actions are applicable in succession starting from $\mathcal{I}$. Actions in Π^{sm}_{a1} maintain all preconditions and relevant effects of the actions in π, and the only extra precondition is that the corresponding *active* variable is true. All *active* variables are set to true in $\mathcal{I}'$, and each action only deactivates preceding actions and itself. Hence, given that ρ is a subsequence of π, all actions in π^{sm}_{a1} are applicable in succession starting from $\mathcal{I}'$. Finally, since $\mathcal{G}' = \mathcal{G}$, π^{sm}_{a1} is a plan for Π^{sm}_{a1}. Thus, π^{sm}_{a1} translated with Definition 8 is in P^{sm}_{a1}. □

If there are no zero-cost actions in Π, an optimal solution for Π^{sm}_{a1} is a minimal reduction of π for Π. We omit the corresponding proof since it is analogous to the one for Theorem 1, and we handle zero-cost action in the exact same way as for Π^{s1}_{a1} tasks.

Theorem 2. *Let π be a plan for planning task Π without zero-cost actions, and π^{sm}_{a1} an optimal plan for the task Π^{sm}_{a1} generated for Π and π. The plan π' obtained from π^{sm}_{a1} using Definition 8 is a minimal reduction of π.*

Proof. The proof is analogous to the proof of Theorem 1. □

4.4 Π^{s1}_{am} Compilation: Applying Multiple Actions at Once

To apply multiple actions at a time, we create macro-actions (Fikes et al. 1972) of consecutive actions in the input plan. Traditionally, generating macro-actions involves identifying sequences of actions that are usually applied together (Dawson and Siklóssy 1977; Fikes et al. 1972; Korf 1985). However, since we are interested in finding a plan reduction of an input plan, we know exactly which actions can be applied consecutively simply by looking at the different consecutive subsequences of the plan. Given a plan $\pi = \langle a_1, a_2, \ldots, a_n\rangle$, the number of consecutive subsequences of at least two actions is $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$. For example, the consecutive subsequences of π with $|\pi| = 3$ are $\langle a_1, a_2\rangle, \langle a_1, a_2, a_3\rangle, \langle a_2, a_3\rangle$. The macro-action generated from $\langle a_i, a_{i+1}, \ldots, a_j\rangle$ represents applying consecutive actions a_i to a_j in a single step.

The preconditions and effects of a macro-action a_{ij} generated from two consecutive actions $\langle a_i, a_j \rangle$ are defined as follows:

- $pre(a_{ij}) = pre(a_i) \cup (pre(a_j) \setminus \mathit{eff}(a_i))$
- $\mathit{eff}(a_{ij}) = \mathit{eff}(a_j) \cup \{v \mapsto d \mid v \mapsto d \in \mathit{eff}(a_i), v \notin vars(\mathit{eff}(a_j))\}$

Chaining two actions in this way is always well-defined for two actions that appear consecutively in the original plan. To generate macro-actions of more than two actions, this procedure can simply be applied repeatedly. Thus, to allow the application of multiple actions at a time, we simply add the appropriate macro-actions in Π^{s1}_{a1} that represent consecutive subsequences of actions in π to Π^{s1}_{a1}.

The added macro-actions increase the number of actions from $2n$ to $2n + \frac{n(n-1)}{2}$ compared to Π^{s1}_{a1}, and the branching factor increases to at most $n + 1$. The increased branching factor might slow down the expansion rate of the search. However, in cases where only a few or no actions can be removed from the original plan, Π^{s1}_{am} will find solutions at shallower depths of the search tree compared to Π^{s1}_{a1}.

Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, a plan $\pi = \langle a_1, a_2, \dots, a_n \rangle$, we define $\Pi^{s1}_{am} = \langle \mathcal{V}', \mathcal{A}'', \mathcal{I}', \mathcal{G}' \rangle$ in terms of $\Pi^{s1}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ by simply modifying the set of actions to include all possible macro-actions of at least two consecutive actions in π: $\mathcal{A}'' = \mathcal{A}' \cup \{macro(a_i, a_j) \mid 1 \leq i < j \leq n\}$, where $macro(a_i, a_j)$ produces the macro-action including all actions from a_i to a_j in succession.

Equivalent propositions to Proposition 1, Proposition 2 and Theorem 1 can be stated for Π^{s1}_{am}, but we omit them because they are practically identical to the propositions and proofs for Π^{s1}_{a1}.

4.5 Π^{sm}_{am} Compilation: Applying and Skipping Multiple Actions at Once

To completely explore the design space of compilations described in Section 4.1, we need to formulate a compilation that can apply and skip multiple actions at a time. To do this, we simply take Π^{sm}_{a1} and create macro-actions in the same way as it is done for Π^{s1}_{am}, and we call it Π^{sm}_{am}. Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ and a plan $\pi = \langle a_1, a_2, \dots, a_n \rangle$, we define $\Pi^{sm}_{am} = \langle \mathcal{V}', \mathcal{A}'', \mathcal{I}', \mathcal{G}' \rangle$ in terms of $\Pi^{sm}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ by modifying the set actions to include all possible macro-actions of at least two consecutive actions in π: $\mathcal{A}'' = \mathcal{A}' \cup \{macro(a_i, a_j) \mid a_i, a_j \in \mathcal{A}' \land 1 \leq i < j \leq n\}$.

Again, equivalent propositions to Proposition 3 and Theorem 2 can be stated for Π^{sm}_{am}, but we omit them because they are practically identical to the propositions and proofs for Π^{sm}_{a1}.

4.6 Comparison Between Π^{s1}_{a1}, Π^{sm}_{a1},Π^{s1}_{am} and Π^{sm}_{am}

Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ and a plan $\pi = \langle a_1, a_2, \dots, a_n \rangle$, we show how the different compilations compare against each other in Table 1. Π^{s1}_{a1} and Π^{s1}_{am} only require the additional *pos* variable, while Π^{sm}_{a1} and Π^{sm}_{am} require the n *active* variables. $|\mathcal{D}(pos)| = n + 1$, so the number of facts in Π^{s1}_{a1} and Π^{s1}_{am} increases only in that amount with respect to R_π, while $|\mathcal{D}(active_i)| = 2$ for each of the n active variables, increasing the number of facts by $2n$ for Π^{sm}_{a1} and Π^{sm}_{am}. Π^{s1}_{a1} has two actions for each action in π, while Π^{sm}_{a1} only has one action for each action in π. Similarly, Π^{s1}_{am} has the $2n$ actions from Π^{s1}_{a1} plus the macro-actions, while Π^{sm}_{am} has n actions plus the macro-actions.

The branching factor for Π^{s1}_{a1} is always 2 (skip or apply each action in π), while the branching factor for Π^{sm}_{a1} can be at most n (apply any of the active actions), for Π^{s1}_{am} the highest branching factor is $n + 1$ (apply any macro-action representing a subsequence starting from the first action or skip the first action), and for Π^{sm}_{am} it is $\frac{n(n+1)}{2}$ (apply any action or any macro-action).

Finally, the number of effects for actions created for both Π^{s1}_{a1} and Π^{s1}_{am} only increases by 1 (updating the *pos* variable), while each action in Π^{sm}_{a1} and Π^{sm}_{am} must deactivate itself and all preceding actions. We omit the size of the preconditions in the table since each compilation only adds one additional precondition to each action (*pos*

Table 1. Comparison of different characteristics of each compilation. For each compilation we show the number of resulting variables ($|\mathcal{V}'|$), the number of actions ($|\mathcal{A}'|$), the number of facts $|\mathcal{F}|$, the upper bound on the branching factor of the search tree induced by the compilation (BF), the number of effects of a reformulated action ($|\mathit{eff}(a')|$) in terms of the number of effects of the action it was created from, and the upper bound on the size of the state space induced by the compilation ($|S|$).

	Π^{s1}_{a1}	Π^{sm}_{a1}	Π^{s1}_{am}	Π^{sm}_{am}
$\lvert\mathcal{V}'\rvert$	$\lvert\mathcal{V}\rvert + 1$	$\lvert\mathcal{V}\rvert + n$	$\lvert\mathcal{V}\rvert + 1$	$\lvert\mathcal{V}\rvert + n$
$\lvert\mathcal{A}'\rvert$	$2n$	n	$2n + \frac{n(n-1)}{2}$	$n + \frac{n(n-1)}{2}$
$\lvert\mathcal{F}\rvert$	$\lvert R_\pi\rvert + n + 1$	$\lvert R_\pi\rvert + 2n$	$\lvert R_\pi\rvert + n + 1$	$\lvert R_\pi\rvert + 2n$
BF	2	n	$n + 1$	$\frac{n(n+1)}{2}$
$\lvert\mathit{eff}(a'_i)\rvert$	$\lvert\mathit{eff}(a_i)\rvert + i$	$\lvert\mathit{eff}(a_i)\rvert + i$	$\lvert\mathit{eff}(a_i)\rvert$	$\lvert\mathit{eff}(a_i)\rvert + i$
$\lvert S\rvert$	$2^{n+1} - 1$	2^n	$2^{n+1} - 1$	2^n

variable or *active* variable). Note that both the effects and preconditions of Π^{s1}_{am} and Π^{sm}_{am} depend on the number of effects and preconditions of the macro-action they represent, so they can have as many as $|\mathcal{V}|$ effects and preconditions.

Regarding the size of the state space induced by each compilation, it is clear that all compilations can generate any subsequence of actions of the input plan. To obtain an upper bound on the size $|S|$ of the reachable state space induced by each of the four compilations, we assume that S contains no duplicate states. Under this assumption, each subsequence of actions from the original plan is a different state. For a plan of length n, the number of subsequences is 2^n. Π^{sm}_{a1} and Π^{sm}_{am} implicitly eliminate actions, so they induce exactly as many states as there are subsequences of the plan (2^n). In contrast, Π^{s1}_{a1} explicitly eliminates actions. This translates into a branching factor of exactly 2 and a state tree of depth n (length of the input plan), so assuming there are no duplicates, the number of states is exactly $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$. The size of the state space for Π^{s1}_{am} is the same as for Π^{s1}_{a1}, since the macro-actions do not generate different states, just more edges.

5 Plan Action Landmarks

Even though identifying if a plan contains any redundant subsequence of actions is NP-complete in general, some actions can easily be detected as necessary. In this section, we discuss *plan action landmarks* (PALs), which are actions that must be part of any plan reduction of a given plan. The idea of identifying these type of actions to speed up action elimination methods was introduced by Med and Chrpa 2022. The authors proposed an algorithm to identify a specific type of plan action landmarks, which we call *trivial plan action landmarks* (TPALs). We use this name to distinguish them from *fix-point plan action landmarks* (FPALs), which extend the notion to a superset of trivial plan action landmarks. We use the name FPAL because these plan action landmarks are computed with a fix-point procedure. In the following, we define trivial and fix-point plan action landmarks, followed by a theoretical analysis of fix-point plan action landmarks.

To simplify the notation and without loss of generality, we describe plan action landmarks in terms of extended planning tasks with *virtual* initial and goal actions. These actions establish the initial state and require that all the goals are achieved, respectively.

Definition 9 (**extended task and plan**). *Given a planning task* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$*, the* extended task *is* $\Pi^e = \langle \mathcal{V}^e, \mathcal{A}^e, \mathcal{I}^e, \mathcal{G}^e \rangle$*, defined as follows.*

- $\mathcal{V}^e = \{\bar{v} \mid v \in \mathcal{V}\} \cup \{v_g\}$*, where* $\mathcal{D}(\bar{v}) = \mathcal{D}(v) \cup \{init\}$ *and* $\mathcal{D}(v_g) = \{init, goal\}$.

- $\mathcal{I}^e = \{v \mapsto init \mid v \in \mathcal{V}^e\}$.
- $\mathcal{A}^e = \{\langle pre(a) \cup \{v_g \mapsto init\}, eff(a)\rangle \mid a \in \mathcal{A}\} \cup \{a_{init}, a_{goal}\}$, *where* $a_{init} = \langle \mathcal{I}^e, \mathcal{I}\rangle$, *and* $a_{goal} = \langle \mathcal{G} \cup \{v_g \mapsto init\}, \{v_g \mapsto goal\}\rangle$.
- $\mathcal{G}^e = \{v_g \mapsto goal\}$.

Each plan $\pi = \langle a_1, a_2, \ldots, a_n\rangle$ *for* Π *corresponds to exactly one* extended plan $\pi^e = \langle a_0, a_1, a_2, \ldots, a_n, a_{n+1}\rangle$ *for* Π^e*, where* a_0 *and* a_{n+1} *are the virtual initial and goal actions* a_{init} *and* a_{goal}*, respectively.*

Definition 10 (**plan action landmark**). *Let* π *be a plan for an extended planning task* Π*. An action* $a \in \pi$ *is a* plan action landmark *of* π *iff* a *is part of all plan reductions of* π*.*

Identifying whether an action in a plan is a plan action landmark is co-NP-complete (Med and Chrpa 2022), but some plan action landmarks can be identified in polynomial time. Trivial plan action landmarks are one such example. Here, the idea is to identify if a necessary fact for the plan has only one achiever. For example, if only one action a in a plan achieves a goal fact, removing a would render the plan invalid. Furthermore, if some preconditions of a are also achieved by a single action a', then a' is also necessary. This is similar to the back-chaining method of fact landmark discovery (Hoffmann et al. 2004).

Definition 11 (**trivial plan action landmark, tpal**). *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ *be an extended planning task and* $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1}\rangle$ *be a plan for* Π*. Action* a_i *is a* trivial plan action landmark *for* π *iff: (1)* a_i *is the goal action* a_{n+1} *or (2) there is a trivial plan action landmark* a_j*,* $i < j$*, such that there is a fact* $p \in eff(a_i) \cap pre(a_j)$*, and there is no action* a_k*,* $k < j, k \neq i$*, such that* $p \in eff(a_k)$*.*

Figure 5a illustrates the concept of TPALs graphically. We now prove that trivial plan action landmarks are in fact plan action landmarks.[5]

Proposition 4. *Let* $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1}\rangle$ *be a plan for an extended planning task* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$*, and* a_i *an action in* π*. If* a_i *is a trivial plan action landmark for* π*, then* a_i *is a plan action landmark for* π*.*

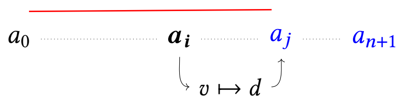
Proof by induction.

(1) Base case: both the virtual initial and goal actions are present exactly once in any plan, and they must always appear at the beginning and the end, respectively. If a_i is a TPAL because it is the virtual goal action a_{n+1}, it must be part of any plan reduction of π for Π, since it is the only action in the plan that achieves the goal fact $v_g \mapsto goal$. Therefore, a_i is a plan action landmark.

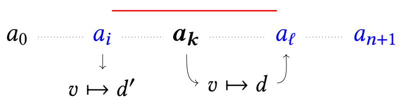
(2) Inductive step: assuming that a_j is a plan action landmark, we show that if a_i is a TPAL because it is the only action situated before a_j achieving a fact p in the precondition of a_j, then a_i is a plan action landmark. Since a_j is a plan action landmark, it is part of all plan reductions of π, and in consequence all facts in its precondition, including p, must be true in some state of the state sequence induced by applying the prefix up to a_j of any plan reduction. Since no other action in that prefix achieves p, all plan reductions must contain a_i. Therefore, a_i is a plan action landmark. □

The concept of *fix-point plan action landmarks* allows discovering additional plan action landmarks. They extend trivial plan action landmarks by exploiting the fact that, since plan action landmarks *must* be executed, their effects can be taken into account when identifying other plan action landmarks. While trivial plan action landmarks identify single achievers of a necessary fact (either because it is a goal fact or part of a precondition of a plan action landmark), fix-point plan action landmarks additionally identify single achievers of a necessary fact *after* a plan action landmark overwrote said fact.

[5]Proposition 4 is equivalent to Propositions 3 and 4 in the paper by Med and Chrpa 2022, but here we present it in a more compact way, by using the extended planning task.



(a) TPAL



(b) FPAL

Fig. 5. Visual representation of (a) a trivial plan action landmark and (b) a fix-point plan action landmark. Actions already known to be trivial and fix-point plan action landmarks, respectively, are marked in blue. In (a), $\boldsymbol{a_i}$ is a trivial plan action landmark because it is the only achiever of $p = v \mapsto d$, which is a precondition of the trivial plan action landmark a_j. The red line indicates the range of plan actions in which a_i is the only achiever. In (b), $\boldsymbol{a_k}$ is a fix-point plan action landmark because it is the only achiever of $v \mapsto d$, a precondition of the fix-point plan action landmark a_ℓ, situated before a_ℓ and after another previous fix-point plan action landmark action, a_i, generates a different value for the same variable v. Note that if there was another achiever of $v \mapsto d$ situated between a_i and a_ℓ, a_k would not be a FPAL.

Definition 12 (**fix-point plan action landmark, fpal**). *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be an extended planning task and $\pi = \langle a_0, a_1, \dots, a_n, a_{n+1} \rangle$ be a plan for Π. Action a_k is a* fix-point plan action landmark *iff: (1) a_k is a trivial plan action landmark or (2) there is a fix-point plan action landmark a_ℓ, $k < \ell$, such that there is a fact $v \mapsto d \in \mathit{eff}(a_k) \cap \mathit{pre}(a_\ell)$, and there is another fix-point plan action landmark a_i, $i < k$, with an effect $v \mapsto d' \in \mathit{eff}(a_i)$ where $d' \neq d$ and there is no other action a_j with $j \neq k$, $i < j < \ell$ such that $v \mapsto d \in \mathit{eff}(a_j)$.*

Figure 5b illustrates the concept of FPALs graphically. We now show that fix-point plan action landmarks are, indeed, plan action landmarks.

Proposition 5. *Let $\pi = \langle a_0, a_1, \dots, a_n, a_{n+1} \rangle$ be a plan for an extended planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, and a_k an action in π. If a_k is a fix-point plan action landmark for π, then a_k is a plan action landmark for π.*

Proof by induction.

(1) If an action a_k is a FPAL because it is a TPAL (case (1), Definition 12), it is a plan action landmark by Proposition 4.

(2) Assuming that a_ℓ and a_i are plan action landmarks, we show that if a_k is a FPAL because a_ℓ and a_i are FPALs with $i < k < \ell$ and a_k is the only achiever of a fact $f = v \mapsto d$, where $f \in \mathit{pre}(a_\ell) \cap \mathit{eff}(a_k)$ and $v \mapsto d' \in \mathit{eff}(a_i)$, with $d \neq d'$ (case (2), Definition 12), then a_k is a plan action landmark. Since a_ℓ is a plan action landmark, it belongs to all plan reductions and, consequently, all facts in its precondition must be true in some state of the state sequence induced by applying the prefix up to a_ℓ of any plan reduction. Analogously, given that a_i is a plan action landmark, it belongs to all plan reductions and, consequently, after the application of a_i in any plan reduction, the value of v is necessarily d'. Since a_k is the only achiever of the fact $v \mapsto d$, in the precondition of a_ℓ, after a_i overwrote it, it must be part of all plan reductions, and therefore it is a plan action landmark. □

5.1 Identifying Plan Action Landmarks

Our method to identify fix-point plan action landmarks (and the subsumed TPALs) is shown in Algorithm 1. The input is an extended planning task and a plan. Initially, the virtual goal action is marked as a PAL. Then, the procedure ComputeAchievers (called in line 3 and defined in lines 22–26) finds the achievers for each fact $v \mapsto d$, i.e., all actions $a_i \in \pi$ such that $v \mapsto d \in \mathit{eff}(a_i)$. For each fact, an achiever is a pair $\langle a_i, k \rangle$, where a_i is an action in π and $i < k$. In this pair, k represents that action a_i is an achiever of fact $v \mapsto d$ *until* step k of the plan. This means that action a_i can achieve a fact for the precondition of another action a_j only if $i < j \leq k$. Initially, for each effect of each action, we initialize k to the last plan position (line 26). Then, until no new plan action landmark is found (line 5), ComputePALs iterates backwards over all plan actions (line 6). If an action is a

Algorithm 1 Compute fix-point plan action landmarks.

```
 1: function ComputeFPALs($\Pi, \pi$)
 2:     $\langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle \leftarrow \pi$
 3:     $PALs \leftarrow \{a_{n+1}\}$
 4:     $achs \leftarrow$ ComputeAchievers($\pi$)                          ▷ Initial achievers
 5:     repeat
 6:         for $i = n + 1$ to 0 do
 7:             if $a_i \in PALs$ then
 8:                 for $v \mapsto d \in pre(a_i)$ do
 9:                     $A \leftarrow \{a_j | \langle a_j, k \rangle \in achs(v \mapsto d),\ j < i \le k\}$
10:                     if $A = \{a_j\}$ and $a_j \notin PALs$ then      ▷ Check if fact has a single achiever
11:                         $PALs \leftarrow PALs \cup \{a_j\}$
12:                         UpdateAchievers($achs, a_j$)
13:     until $PALs$ remains unchanged
14:     return $PALs$
15:
16: procedure UpdateAchievers($achs, a_j$)                              ▷ Update achievers
17:     for $v \mapsto d \in eff(a_j)$ do
18:         for $d' \in \mathcal{D}(v) \setminus \{d\}$ do
19:             for $(a_i, k) \in achs(v \mapsto d')$ with $i < j < k$ do
20:                 $achs(v \mapsto d') = (achs(v \mapsto d') \setminus \langle a_i, k \rangle) \cup \langle a_i, j \rangle$
21:
22: function ComputeAchievers($\pi$)
23:     $achs(v \mapsto d) = \emptyset$ for all $v \in \mathcal{V}, d \in \mathcal{D}(v)$
24:     for $a_i \in \pi$ do
25:         for $v \mapsto d \in eff(a_i)$ do
26:             $achs(v \mapsto d) = achs(v \mapsto d) \cup \langle a_i, |\pi| \rangle$
27:     return $achs$
```

plan action landmark (line 7), it checks its precondition (lines 8–10) to see if a new plan action landmark can be identified using Definition 12: is there a unique achiever a_j that was not marked as a plan action landmark before? If a new plan action landmark a_j is identified, it is marked as such (line 11) and the *until value* of actions whose effects are overwritten by a_j are updated (line 12). Procedure UpdateAchievers (lines 16–20) finds, for every $v \mapsto d \in eff(a_j)$, the actions a_i before a_j that set v to d', $d' \neq d$, and updates their *until value* to j.

With only a single iteration of the fix-point loop (lines 5–13) and without the Update call (line 11), Algorithm 1 finds the set of all trivial plan action landmarks, and we call it ComputeTPALs.

The following two propositions follow from the fact that ComputeTPALs and ComputeFPALs are direct implementations of Definitions 11 and 12.

Proposition 6. *Algorithm ComputeTPALs finds all trivial plan action landmarks.*

Proposition 7. *Algorithm 1 finds all fix-point plan action landmarks.*

The ComputeTPALs variant is equivalent to the algorithm proposed by Med and Chrpa 2022. The authors showed that it runs in linear time in the length of the plan if the size of action preconditions and effects is assumed to be constant. To find fix-point plan action landmarks, the fix-point loop in Algorithm 1 must be repeated every time a new plan action landmark is found. In the worst case, each iteration only discovers one plan action landmark, so the loop runs at most $|\pi|$ times. Making the same assumptions on the size of preconditions and effects, Algorithm 1 thus runs in $O(|\pi|^2)$.

5.2 Using Plan Action Landmarks

Since plan action landmarks must be part of any plan reduction, there is no need to consider skipping or eliminating them when looking for a minimal reduction. This reduces the branching factor and speeds up the search process for all compilations. We now show how to use information about plan action landmarks to shrink the search space and/or the task size of the four planning compilations.

Reducing the Number of Skip Actions in Π^{s1}_{a1} and Π^{s1}_{am}. Both Π^{s1}_{a1} and Π^{s1}_{am} have explicit *skip* actions that represent eliminating actions, making it easy to not consider removing plan action landmarks simply by omitting their corresponding skip actions. The resulting PAL-enhanced tasks for Π^{s1}_{a1} and Π^{s1}_{am} preserve $\mathcal{V}', \mathcal{I}', \mathcal{G}'$ exactly as in Section 4, and only creating skip actions for actions that are not known plan action landmarks. For this, $\mathcal{A}'$ is redefined as $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\} \cup \{skip_i \mid 1 \leq i \leq n \wedge \neg pal(a_i)\}$.

Reducing the Number of Macro-Actions in Π^{s1}_{am} and Π^{sm}_{am}. One clear detriment of Π^{s1}_{am} and Π^{sm}_{am} is the quadratic number of actions that are created from the input plan. By only creating macro-actions using actions that are not known to be plan action landmarks, the number of actions and the branching factor can be greatly reduced.

Reducing the Branching Factor in Π^{sm}_{a1} and Π^{sm}_{am}. Whenever an action a_i is applied, all variables $active_j$, with $j \leq i$ are set to false. This represents eliminating all actions a_j with $j < i$ for which $active_j$ is true before applying a_i. In Π^{sm}_{a1} and Π^{sm}_{am} actions are not explicitly removed. To ensure that no plan action landmark is removed, we will modify which actions are *active* in the initial state, and also change the effects of all actions by taking into account plan action landmarks. We describe these changes formally with the help of two functions: *firstPalAfter*(i) and *lastPalBefore*(i), which, for a given planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ and plan $\pi = \langle a_1, \ldots, a_n \rangle$, provide the index of the first plan action landmark with index higher than i (or n if there are none), and the index of the last plan action landmark with index smaller than i (or 0 if there are none), respectively:

$$firstPalAfter(i) = \begin{cases} n & \text{if } \neg pal(a_k) \text{ for all } k > i, \\ \min\{k \mid i < k \leq n \wedge pal(a_k)\} & \text{otherwise, and} \end{cases}$$

$$lastPalBefore(i) = \begin{cases} 0 & \text{if } \neg pal(a_k) \text{ for all } k < i, \\ \max\{k \mid 1 \leq k < i \wedge pal(a_k)\} & \text{otherwise.} \end{cases}$$

We show how to modify the task $\Pi^{sm}_{a1} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ obtained from Π and π, but the same changes apply to Π^{sm}_{am}. The initial state $\mathcal{I}'$ is modified so only actions up to the first plan action landmark in π are active:

$$\begin{aligned} \mathcal{I}' = \{\tau(v \mapsto d) \mid v \mapsto d \in \mathcal{I} \wedge v \in \mathcal{V}'\} \cup \{active_i \mid 1 \leq i \leq firstPalAfter(0)\} \cup \\ \{\neg active_i \mid firstPalAfter(0) < i \leq n\} \end{aligned}$$

Regarding the effects of the actions $a'_i \in \mathcal{A}'$, when a_i is a plan action landmark, all actions up to the next plan action landmark must be activated; and when an action is applied, only the preceding actions up to the previous plan action landmark are *deactivated.*

$$\begin{aligned} \mathit{eff}(a'_i) = &\{\tau(v \mapsto d) \mid v \mapsto d \in \mathit{eff}(a_i) \wedge v \in \mathcal{V}'\} \cup \\ &\{\mathit{active}_j \mid \mathit{pal}(a_i) \wedge i < j \leq \mathit{firstPalAfter}(i)\} \cup \\ &\{\neg \mathit{active}_j \mid \mathit{lastPalBefore}(i) < j \leq i\} \end{aligned}$$

Furthermore, since plan action landmarks must be executed, each action must only deactivate all preceding actions up to the last plan action landmark (which will have deactivated all preceding actions up to the plan action landmark before that, and so on). When dealing with particularly long plans, this can reduce the size of the effects of each action drastically.

Macro-Actions of Plan Action Landmarks. For all compilations, since plan action landmarks must be part of any plan reduction, consecutive subsequences of plan action landmarks can be replaced by a single macro-action. The effect it has on the search space is reducing the depth at which solutions are found. To maximize this effect, the longest possible subsequences must be chosen to be encapsulated as a macro-action. For a plan $\pi = \langle a_1, a_2, \ldots, a_n \rangle$, let *CPAL* be the set of the *longest* consecutive subsequences of plan action landmarks of π. For all subsequences in *CPAL*, their corresponding actions are replaced by the macro-action encapsulating all of them. The macro-actions are constructed as explained in Section 4.4.

6 Always Redundant Actions

Similarly to plan action landmarks, one might wonder if there are actions that can be identified as *always redundant*, and not be considered when searching for a plan reduction of a given plan. However, the notion is not as clear as plan action landmarks. An action can be redundant because another action achieves the same facts, but if the other action is removed, this action might no longer be redundant in the resulting plan reduction. In the context of minimal reduction, we now propose actions that can be identified as redundant in polynomial time.

In the minimal reduction problem, the goal is to find a perfectly justified plan reduction of least cost. So an action that is never part of a perfectly justified plan reduction cannot be part of a minimal reduction. We call this type of actions *always redundant actions.* As in the case of plan action landmarks, we use the extended planning task to simplify notation.

Definition 13 (**always redundant action**). *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *be an extended planning task and* $\pi = \langle a_0, a_1, a_2, \ldots, a_n, a_{n+1} \rangle$ *be a plan for* Π. *An action* $a_i \in \pi$ *is* always redundant *iff there is no perfectly justified plan reduction* ρ *of* π *such that* $a_i \in \rho$.

Given a plan π for a task Π, if an action $a_i \in \pi$ does not achieve any goal fact nor any precondition fact of another action, it can always be removed from any plan reduction ρ of π without affecting its validity. By Definition 6, ρ is not perfectly justified, so a_i is always redundant. Furthermore, if an action a_i only generates facts that are part of the precondition of always redundant actions a_j, following the same reasoning, a_i will not be part of any perfectly justified plan reduction. We call this type of actions *trivially redundant actions.* Note that, by Definition 9, virtual actions a_0 and a_{n+1} must be part of any plan for an extended task, so they are never redundant.

Definition 14 (**trivially redundant action**). *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *be an extended planning task and* $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ *be a plan for* Π. *Actions* a_0 *and* a_{n+1} *are never trivially redundant. Action* a_i, $1 \leq i \leq n$, *is a* trivially redundant action *for* π *iff: for all* a_j, $i < j \leq n+1$, $\mathit{eff}(a_i) \cap \mathit{pre}(a_j) = \emptyset$ *or* a_j *is trivially redundant.*

Proposition 8. *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *be an extended planning task and* $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ *be a plan for* Π. *If* a_i *is trivially redundant, then* a_i *is always redundant.*

Proof by induction.

(1) Base case: suppose a_i is trivially redundant because it does not achieve any precondition fact required by any subsequent action in the plan. That is, for all j, $i < j \leq n + 1$, $\mathit{eff}(a_i) \cap \mathit{pre}(a_j) = \emptyset$. Assume, for contradiction, that a_i is part of a perfectly justified plan reduction ρ of π. Let $\rho' = \rho \setminus \{a_i\}$, that is, the plan obtained by removing a_i from ρ. Since a_i's effects are not required by any subsequent action in π, removing a_i preserves applicability and the goal remains achieved. Therefore, ρ' is also a plan reduction of π, and $|\rho'| < |\rho|$. This contradicts the assumption that ρ is perfectly justified. Hence, a_i cannot be part of a perfectly justified plan reduction, and is therefore always redundant.

(2) Inductive step: assume for all k, $i < k \leq n + 1$, if a_k is trivially redundant, then a_k is always redundant. Let a_i be a trivially redundant action such that $\mathit{pre}(a_i) \cap \mathit{eff}(a_j) \neq \emptyset$ for some j, $i < j \leq n + 1$. Now suppose, for contradiction, that a_i is part of a perfectly justified plan reduction ρ of π. By the inductive hypothesis, all such a_j are always redundant and therefore cannot be included in any perfectly justified plan reduction of π. Hence, none of a_i's effects are required in ρ. This reduces to the base case and by the argument in the base case, a_i must then be always redundant—contradicting the assumption that it appears in a perfectly justified plan reduction. □

In a similar fashion as fix-point plan action landmarks, the effects of known plan action landmarks can be used to identify always redundant actions that are not trivially redundant. If an action a_i only achieves facts that are overwritten by a plan action landmark a_j before they are read by another action a_k, a_i can always be removed. However, in practice, this characterization is too restrictive to be useful: for a_i to be always redundant, there needs to be a plan action landmark $a_j \in \mathcal{A}$, with $i < j$, that sets the value of a variable v which does not occur in the precondition of a_j, that is, $v \in \mathit{vars}(\mathit{eff}(a_j)) \setminus \mathit{vars}(\mathit{pre}(a_j))$, and there must be no action a_k, with $i < k < j$, with a precondition involving variable v. This type of always redundant actions is absent from our benchmarks and plans.

Note that trivially redundant actions and actions that are not backward justified (Definition 3) are *not* equivalent. An action is not backward justified if there is no chain of causal links connecting it to the virtual goal action. However, this does not imply that the action can never be part of a perfectly justified plan reduction: causal links only consider the last achiever of a fact, so if multiple actions achieve the same fact, the action in question may still be included in a minimal reduction—provided all other achievers are removed. In contrast, always redundant actions never achieve any fact that could be useful in reaching the goals in any perfectly justified plan reduction. Thus, all trivially redundant actions are not backward justified, but not all actions that are not backward justified are trivially redundant.

Enhancing the performance of all planning compilations with information about always redundant actions is straightforward. Since always redundant actions are never part of a perfectly justified plan reduction, simply omitting the always redundant actions from the set of actions for the reformulated tasks is enough.

7 Evaluation

We now evaluate the different methods for finding minimal reductions experimentally. We solve the different planning compilations with the Scorpion planning system (Seipp, Keller, et al. 2020), which is an extension of Fast Downward (Helmert 2006). We use A^* with an admissible heuristic (described below) to find optimal plans, which is necessary to find minimal plan reductions with our approach. We use Python 3.10 to generate the planning task reformulations. To generate the WPMaxSAT tasks formulations we use the Java code by Balyo, Chrpa, et al. 2014. To solve the WPMaxSAT formulations, we use the Sat4J solver (Le Berre and Parrain 2010). All algorithm runs are limited to a runtime of 30 minutes and 8 GiB of memory. Code and data for the planning compilations are available online (Salerno et al. 2025).

Table 2. Per-domain benchmark analysis. For each domain, we show the number of plans (#), the length of the shortest plan ($min(L)$), the average number of actions in each plan ($\overline{L}$), and the length of the longest plan ($max(L)$). Additionally, for each planner, we show the average percentage of redundant actions in plans (number of redundant actions divided by the input plan length, multiplied by 100) and, in parentheses, the number of plans the planner found for each domain.

Domain	#	min(L)	$\overline{L}$	max(L)	Cerberus	Madagascar	LAMA	BFWS	YASHP3
Agricola	18	53	89.72	152	0.00 (2)	– (0)	0.00 (6)	0.00 (10)	– (0)
Barman	57	150	196.40	377	6.36 (17)	– (0)	17.82 (19)	4.35 (20)	47.21 (1)
ChildSnack	25	33	55.48	86	0.00 (1)	4.05 (19)	5.92 (5)	– (0)	– (0)
DataNetwork	34	18	95.82	402	3.57 (9)	57.49 (4)	15.90 (11)	7.40 (10)	– (0)
Floortile	47	36	83.11	127	7.99 (20)	4.05 (20)	5.13 (2)	3.50 (2)	22.15 (3)
GED	68	62	153.63	390	0.00 (20)	– (0)	0.00 (20)	0.00 (20)	4.50 (8)
Hiking	47	13	446.89	4058	6.54 (8)	5.38 (8)	0.67 (9)	0.00 (8)	55.33 (5)
OpenStacks	53	442	731.25	1115	0.00 (13)	– (0)	0.00 (20)	0.00 (20)	– (0)
OrgSynthesis	9	2	4.00	8	0.00 (3)	– (0)	0.00 (3)	0.00 (3)	– (0)
OrgSynthesisSplit	21	24	37.62	68	0.00 (4)	0.00 (4)	0.00 (8)	0.00 (5)	– (0)
Parking	45	70	103.64	147	0.00 (5)	– (0)	0.59 (20)	0.00 (20)	– (0)
Snake	26	34	91.27	270	0.00 (5)	– (0)	0.00 (3)	0.00 (18)	– (0)
Termes	31	100	732.00	2944	4.94 (11)	– (0)	9.77 (11)	40.33 (6)	– (0)
Tetris	37	23	71.59	157	0.28 (13)	10.11 (4)	6.67 (2)	10.53 (17)	4.76 (1)
Thoughtful	69	27	102.83	318	0.85 (18)	7.86 (5)	0.34 (16)	0.20 (20)	21.79 (10)
Transport	53	143	480.85	1358	0.00 (6)	– (0)	10.15 (7)	7.50 (20)	26.22 (20)
VisitAll	60	919	3046.97	4976	0.00 (4)	– (0)	0.72 (16)	0.00 (20)	0.03 (20)

7.1 Benchmark Analysis

As benchmarks, we use the same tasks and input plans as Med and Chrpa 2022.[6] We omit tasks with conditional effects since the WPMaxSAT approach does not support them.[7] The benchmark set consists of tasks from the Agile tracks of the International Planning Competitions (IPC) of 2014 and 2018, and plans found with the following planners: Cerberus (Katz 2018), Freelunch Madagascar (Balyo and Gocht 2018), LAMA 2011 (Richter, Westphal, and Helmert 2011), BFWS Preference (Francès et al. 2018) and YAHSP3 (Vidal 2014). We refer to LAMA 2011 simply as "LAMA", to Freelunch Madagascar as "Madagascar" and to BFWS Preference as "BFWS". All planners aim to find a single plan as fast as possible.

The number of actions in a plan directly influences the size of the state space induced by the reformulated task, and it also directly impacts the number of variables and clauses in the WPMaxSAT formulation. Table 2 shows the size of the input plans, as well as the ratio of redundant actions. In general, we are dealing with plans with hundreds of steps, while plans for VisitAll have thousands of steps. Particularly short plans are only found in OrgSynthesis and OrgSynthesisSplit. LAMA and BFWS produce the lowest ratio of redundant actions on average, close to 4% on average, while still producing a high number of redundant actions in some domains, like Barman for LAMA (17.82%) and Termes for BFWS (40.33%). On the other hand, Madagascar and YASHP3 produce highly redundant plans, with 12% and 22.75% of the actions being redundant on average for each plan, respectively.[8]

[6]While they find plan reductions of arbitrary cost, we solve the minimal reduction problem.

[7]Adding support for conditional effects to our planning compilations is straightforward and we have done so for our submission to the International Planning Competition (Salerno et al. 2023b).

[8]We obtained these statistics using the best configuration for Π^{s1}_{a1}, discussed in Section 7.4.

Table 3. Comparison of the total time needed to reformulate all tasks, the total time needed to solve all tasks, the sum of these two runtimes, the geometric mean of the number of expansions and the number of solved tasks. All values are computed only over the commonly solved tasks, except for coverage. Note that solving a task compilation implies finding a plan reduction, which includes the case of returning the input plan if it already is a minimal reduction.

	Π^{s1}_{a1}	Π^{sm}_{a1}	Π^{s1}_{am}	Π^{sm}_{am}
Reformulation Time (s)	**6.70**	35.84	1497.56	7345.67
Planning Time (s)	**16.22**	32.14	716.64	55934.31
Compilation + Planning Time (s)	**22.92**	67.98	2214.20	63279.98
Expansions	105.64	84.41	86.02	**68.58**
Coverage	**683**	670	551	414

7.2 Comparison of Planning Compilations

We begin by comparing the performance of the different planning compilations without using plan action landmarks. Table 3 shows an aggregated comparison of the performance of Π^{s1}_{a1}, Π^{sm}_{a1}, Π^{s1}_{am} and Π^{sm}_{am} when using $h^{\max}$ (Bonet and Geffner 2001) as the heuristic function. We use this simple heuristic to compare the proposed methods, and will evaluate the best configuration of each approach with more advanced heuristic functions in Section 7.4.

When it comes to time, Π^{s1}_{a1} is fastest overall, both regarding time needed to create the planning task and the time needed to solve it. Π^{sm}_{am} is noticeably slower compared to both Π^{sm}_{a1} and Π^{s1}_{am}.

The reformulation and solving times for Π^{s1}_{am} and Π^{sm}_{am} are extremely high when compared to the other reformulations (particularly for Π^{sm}_{am}). This follows from the fact that, as shown in Table 2, many plans have thousands of actions. The number of actions to be created by Π^{s1}_{am} and Π^{sm}_{am} is quadratic in the length of the input plan (see Table 1), which explains this increase both in reformulation and solving time.

For Π^{sm}_{a1}, the reformulation time is higher than Π^{s1}_{a1}, even though the number of actions is lower for the former ($|\pi|$ for Π^{sm}_{a1} and $2 \times |\pi|$ for Π^{s1}_{a1}). This counter-intuitive result comes from the fact that actions in Π^{sm}_{a1} have a large number of effects: each action must deactivate all preceding actions. Π^{sm}_{am} is affected by this large number of effects to an even greater extent because of its larger number of actions.

The number of expansions is lower for Π^{sm}_{a1}, Π^{s1}_{am} and Π^{sm}_{am} when compared to Π^{s1}_{a1} which is to be expected. Allowing to skip and apply multiple actions at a time means states found at a deeper level of the search tree generated by Π^{s1}_{a1} can be achieved in a single expansion by the other compilations. However, as can be inferred from Table 1, in general the overhead of the large number of effects and actions, as well as the higher branching factor, translates into a slower expansion rate for the other compilations compared to Π^{s1}_{a1}, and thus incurs a slower planning process.

A per-domain comparison of the four compilations can be found in Figure 6 and in Table 4. For the OpenStacks and VisitAll domains, both Π^{s1}_{am} and Π^{sm}_{am} fail to solve any tasks within the time and memory limits. In contrast, Π^{s1}_{a1} and Π^{sm}_{a1} solve all tasks for OpenStacks, and 58/60 and 53/60 for VisitAll, respectively. Π^{s1}_{a1} is particularly faster than Π^{sm}_{a1} for both domains. This is not reflected in Table 4, since we only include statistics for tasks solved by all compilations, but it can be appreciated in Figure 6(a). For Π^{s1}_{am} and Π^{sm}_{am}, the cause for the weak performance in these domains is the quadratic number of actions created from the input plans, in conjunction with the particularly long plans from those domains.[9]

[9]This problem will be alleviated by the use of plan action landmark information in the following section.

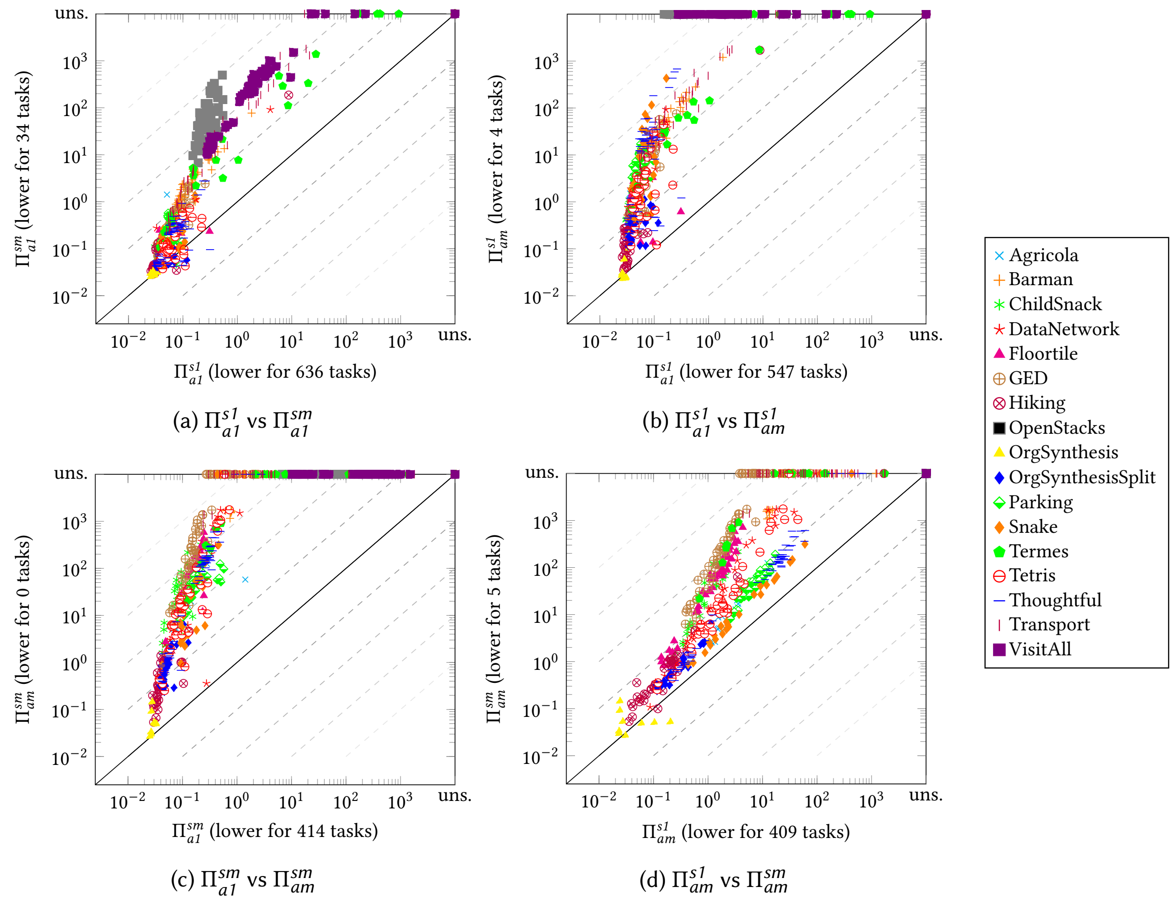


Fig. 6. Comparison of the compilation+planning time between the different planning formulations using the $h^{\max}$ heuristic.

Overall, Π^{s1}_{a1} outperforms the other three approaches, solving all tasks in all domains except for Hiking (37/47), Termes (26/31) and Visitall (58/60). In general, the solving time of Π^{s1}_{a1} is lower than the solving time of Π^{sm}_{a1}, Π^{s1}_{am} and Π^{sm}_{am}. However, there are cases where the opposite is true.

7.3 Plan Action Landmarks and Always Redundant Actions

In this section, we analyze how prevalent trivial and fix-point plan action landmarks are in our benchmarks, as well as always redundant actions. Additionally, we study the impact of enhancing the planning reformulations with fix-point plan action landmark information.

7.3.1 Prevalence of Trivial and Fix-Point Plan Action Landmarks. Table 5 shows that, on average, more than half of the actions of all plans are trivial plan action landmarks, while over 85% of actions are fix-point plan action landmarks. Particularly, in many domains **all** actions are fix-point plan action landmarks (OrgSynthesis, OrgSynthesisSplit, OpenStacks, Snake, Agricola). In these cases, the FPALs prove that the input plan is perfectly justified, and no planner call is needed. For other domains, a high number of plan action landmarks will greatly reduce the branching factor of the search when solving the reformulated planning tasks. For example, in the

Table 4. Per-domain statistics for our four compilations using $h^{\max}$: number of instances (#), coverage (C), time needed to reformulate all tasks (RT), time needed to solve all tasks (ST), and geometric mean of the number of expansions (E). All values are computed over the commonly solved tasks, except for coverage.

		Π^{s1}_{a1}				Π^{sm}_{a1}				Π^{s1}_{am}				Π^{sm}_{am}			
Domain	#	C	RT	ST	E	C	RT	ST	E	C	RT	ST	E	C	RT	ST	E
Agricola	18	18	**0.3**	**0.6**	86.7	18	2.6	0.8	86.7	18	57.8	9.7	**77.8**	18	277.3	147.4	**77.8**
Barman	57	**57**	**0.2**	**0.4**	1332.6	**57**	1.4	3.6	478.7	**57**	52.0	39.4	1329.5	7	484.6	9823.3	**475.5**
ChildSnack	25	25	**0.2**	**0.8**	68.5	25	0.7	1.4	61.9	25	20.1	5.4	67.3	25	65.9	1097.4	**60.6**
DataNetwork	34	**34**	**0.3**	**1.4**	162.4	**34**	1.8	4.2	131.0	33	52.5	86.8	153.4	30	439.8	7824.9	**125.3**
Floortile	47	47	**0.5**	**2.0**	197.3	47	2.4	4.6	156.3	47	45.4	24.9	195.1	47	349.9	5016.6	**154.2**
GED	68	**68**	**0.7**	**1.5**	109.6	**68**	3.8	2.9	102.7	**68**	59.5	18.5	98.9	44	741.2	13580.5	**92.3**
Hiking	47	**37**	**0.2**	**1.1**	42.8	**37**	0.4	1.5	37.8	36	3.8	5.3	38.7	36	12.3	74.3	**34.0**
OpenStacks	53	**53**	–	–	–	**53**	–	–	–	0	–	–	–	0	–	–	–
OrgSynthesis	9	9	**0.0**	**0.2**	4.3	9	**0.0**	**0.2**	4.3	9	0.1	0.5	**2.8**	9	0.1	0.5	**2.8**
OrgSynthesisSplit	21	21	**0.4**	0.8	37.6	21	0.6	**0.7**	37.1	21	6.4	1.7	7.7	21	19.3	6.1	**7.5**
Parking	45	45	**0.8**	**1.6**	104.9	45	6.2	2.8	103.7	45	257.6	62.7	100.0	45	1082.6	1151.7	**98.8**
Snake	26	**26**	**0.5**	**0.9**	70.4	**26**	2.4	1.2	70.1	**26**	161.7	60.6	43.2	23	564.6	340.5	**43.0**
Termes	31	**26**	**0.1**	**0.2**	643.2	20	0.8	1.0	230.7	16	7.6	5.7	640.6	6	263.5	2047.9	**228.3**
Tetris	37	**37**	**1.1**	**1.6**	138.1	**37**	3.1	3.0	107.5	**37**	91.2	88.9	134.2	36	386.0	5713.7	**104.2**
Thoughtful	69	**69**	**1.6**	**3.0**	98.9	**69**	8.9	3.9	83.3	**69**	672.4	303.0	94.0	64	2488.5	5657.6	**78.7**
Transport	53	**53**	**0.0**	**0.1**	181.4	**53**	0.6	0.3	159.9	43	9.6	3.7	173.2	3	170.4	3452.0	**151.7**
VisitAll	60	**58**	–	–	–	52	–	–	–	0	–	–	–	0	–	–	–
Overall	700	**683**	**6.7**	**16.2**	105.6	671	35.8	32.1	84.4	551	1497.6	716.6	86.0	414	7345.7	55934.3	**68.6**

VisitAll domain, if trivial plan action landmarks are used to enhance the search, the branching factor is reduced from 2 to 1.14 on average, and to 1.004 for fix-point plan action landmarks. From these results, we can expect a large reduction of planning time for domains with a high prevalence of plan action landmarks, like Visitall, while domains with low prevalence, like Termes, will remain challenging.

7.3.2 Prevalence of Always Redundant Actions. On average, only 0.7% of actions in plans in our benchmark set are trivially redundant, indicating that modern satisficing planners very rarely yield plans with actions that cannot contribute to the goals of the task, but this does not mean they cannot be more prevalent in different settings. A setting where the idea of always redundant actions can be helpful is plan reuse: imagine you have a plan for a planning task, but you are interested in achieving only a subset of the original goal facts (Fink and Yang 1992). If the effects of some actions are only related to goals that are not in the selected subset of goals, they will be identified as always redundant and can be safely removed, quickly yielding a better plan for this specific subset of goals. Table 6 shows the average ratio of trivially redundant actions for each domain in the benchmark set, where each task is modified by only keeping a subset of the original goals. We consider the original task (100% of the goals), and tasks with 75%, 50% and 25% of the goals. As the number of goals gradually decreases, trivially redundant actions become more and more common, showing they can be useful in the plan-reuse context.

Although few plans in our benchmark set contain trivially redundant actions, this does not mean they are never generated by modern planners. Take, for instance, the running example task from the Blocksworld domain shown in Figure 1 and the plan $\langle$*pick-up-b*, *stack-b-a*, *pick-up-c*, *stack-c-d*$\rangle$. The goal of the task is to stack B on top of A, so actions *stack-c-d* and *pick-up-c* are not causally related to the goals of the task. By Definition 14, both actions are trivially redundant: the effects of *stack-c-d* are not read by any action (the virtual goal action only has the precondition $\{\textit{b-pos} \mapsto a, \textit{a-pos} \mapsto \textit{table}\}$, and the effects of *pick-up-c* are only read by the trivially redundant

Table 5. For each domain, we show the minimum ratio (min), average ratio (avg) and maximum ratio (max) of trivial and fix-point plan action landmarks (number of PALs over plan length).

	TPAL			FPAL		
Domain	min	avg	max	min	avg	max
Agricola	0.687	0.745	0.804	1.000	1.000	1.000
Barman	0.095	0.195	0.247	0.377	0.830	1.000
ChildSnack	0.558	0.874	1.000	0.667	0.920	1.000
DataNetwork	0.161	0.579	0.903	0.161	0.794	1.000
Floortile	0.449	0.591	0.722	0.640	0.867	1.000
GED	0.031	0.436	0.864	0.059	0.966	1.000
Hiking	0.007	0.506	0.882	0.008	0.699	1.000
OpenStacks	0.758	0.822	0.925	1.000	1.000	1.000
OrgSynthesis	1.000	1.000	1.000	1.000	1.000	1.000
OrgSynthesisSplit	0.426	0.893	1.000	1.000	1.000	1.000
Parking	0.556	0.648	0.728	0.865	0.983	1.000
Snake	0.107	0.361	0.786	1.000	1.000	1.000
Termes	0.000	0.051	0.199	0.000	0.282	1.000
Tetris	0.000	0.133	0.391	0.000	0.629	1.000
Thoughtful	0.253	0.803	1.000	0.459	0.949	1.000
Transport	0.096	0.395	0.741	0.369	0.785	1.000
VisitAll	0.643	0.858	0.987	0.957	0.996	1.000
Overall (700)	0.000	0.582	1.000	0.000	0.865	1.000

action *stack-c-d*.) Plans for top-k planning tasks are bound to include such actions: in a Blocksworld task, a second cheapest plan can always be constructed by picking up any block not mentioned in the goal description after reaching a goal state, and such an action will be trivially redundant.

7.3.3 Using Plan Action Landmarks. We now analyze how enhancing each reformulation with fix-point plan action landmarks affects its performance.

Table 7 shows the impact plan action landmarks have on the performance of each individual compilation. The use of TPALs improves performance for all compilations, and FPALs similarly improve performance over only using TPALs. The coverage increases for all approaches: from 683 to 688 solved tasks for Π^{s1}_{a1}; 670 to 684 for Π^{sm}_{a1}; 550 to 676 for Π^{s1}_{am}; and 414 to 670 for Π^{sm}_{am}. Π^{s1}_{am} (Table 7c) and Π^{sm}_{am} (Table 7d) are the greatest beneficiaries of plan action landmarks. This result follows from the fact that, by using plan action landmarks, the number of actions created for Π^{s1}_{am} and Π^{sm}_{am} is lower, vastly reducing the time needed to create the task, which was the main culprit of its poor performance without plan action landmarks. The increase in reformulation time for Π^{s1}_{a1} (Table 7a) is due to the time needed to compute fix-point plan action landmarks, which is considered in the reformulation time. However, since the planning time is greatly reduced, the overall time is still lower than without plan action landmarks. The counter-intuitive decrease in reformulation time for Π^{sm}_{a1} (Table 7b) is due to the fact that the number of effects in the actions of Π^{sm}_{a1} shrinks drastically: each action must only "deactivate" all preceding actions up to the previous plan action landmark. In the best case, this reduces the number of effects of an action from thousands to 1, if the previous action is known to be a plan action landmark (only needs to

Table 6. Average ratio of trivially redundant actions for tasks modified by keeping only a fraction of goal facts. The absolute number of goals in each modified task is rounded down to the closest integer.

Domain	25%	50%	75%	100%
Agricola	1.000	0.949	0.949	0.000
Barman	0.162	0.083	0.035	0.000
ChildSnack	0.709	0.457	0.262	0.031
DataNetwork	0.411	0.216	0.111	0.057
Floortile	0.476	0.299	0.108	0.016
GED	0.930	0.869	0.024	0.000
Hiking	0.599	0.231	0.119	0.008
OpenStacks	0.013	0.002	0.001	0.000
OrgSynthesis	0.667	0.153	0.153	0.000
OrgSynthesisSplit	0.430	0.239	0.046	0.000
Parking	0.064	0.010	0.007	0.000
Snake	0.386	0.151	0.069	0.000
Termes	0.080	0.017	0.011	0.000
Tetris	0.682	0.156	0.102	0.010
Thoughtful	0.083	0.036	0.008	0.000
Transport	0.288	0.130	0.045	0.005
VisitAll	0.005	0.002	0.002	0.000
Overall (700)	0.411	0.235	0.121	0.007

Table 7. Impact of TPALs and FPALs on the performance of the different planning compilations. RT is the time needed to reformulate all tasks, PT is the time needed to find plans for all reformulated tasks, TT is the total time needed to solve all tasks (RT + PT), E is the geometric mean of the number of expansions and C is the number of solved tasks (coverage). All metrics (except for coverage) consider commonly solved tasks by the three configurations in the respective table.

	Π^{s1}_{a1}	Π^{s1}_{a1}-TPAL	Π^{s1}_{a1}-FPAL
RT (s)	**38.59**	59.85	306.86
PT (s)	3260.66	1142.96	**740.24**
TT (s)	3299.24	1202.80	**1047.10**
E	283.25	85.54	**10.41**
C	683	686	**688**

(a) Impact of TPALs and FPALs on Π^{s1}_{a1}.

	Π^{sm}_{a1}	Π^{sm}_{a1}-TPAL	Π^{sm}_{a1}-FPAL
RT (s)	7836.76	**114.45**	302.96
PT (s)	26382.58	1067.05	**828.21**
TT (s)	34219.34	1181.50	**1131.17**
E	183.16	63.06	**7.31**
C	670	682	**684**

(b) Impact of TPALs and FPALs on Π^{sm}_{a1}.

	Π^{s1}_{am}	Π^{s1}_{am}-TPAL	Π^{s1}_{am}-FPAL
RT (s)	4479.09	718.11	**140.62**
PT (s)	19228.46	1684.42	**1200.30**
TT (s)	23707.55	2402.54	**1340.92**
E	140.63	51.00	**9.27**
C	551	675	**676**

(c) Impact of TPALs and FPALs on Π^{s1}_{am}.

	Π^{sm}_{am}	Π^{sm}_{am}-TPAL	Π^{sm}_{am}-FPAL
RT (s)	7345.67	518.83	**141.74**
PT (s)	55934.31	4162.95	**2239.28**
TT (s)	63279.98	4681.78	**2381.02**
E	68.58	28.18	**4.02**
C	414	658	**670**

(d) Impact of TPALs and FPALs on Π^{sm}_{am}.

Table 8. Time needed to reformulate all tasks, time needed to solve all tasks, sum of reformulation and planning time, geometric mean of number of expansions and coverage. All approaches are enhanced using fix-point plan action landmarks and macro-actions of consecutive plan action landmarks. The heuristic used to solve all problems is $h^{\max}$.

	Π^{s1}_{a1}	Π^{sm}_{a1}	Π^{s1}_{am}	Π^{sm}_{am}
Reformulation Time (s)	**7.85**	8.98	45.84	153.14
Planning Time (s)	**12.9**	14.36	65.76	2329.78
Total Time (s)	**20.75**	23.34	111.6	2482.92
Expansions	4.73	4.04	4.7	**4.02**
Coverage	**688**	684	676	670

deactivate itself). Using PALs reduces the number of expansions for all compilations, as expected, due to the reduced branching factor.

Table 8 offers a direct comparison of performance between the different compilations enhanced with FPALs (the same statistics as Table 3, but now all compilations use fix-point plan action landmarks as explained in Section 5.2). Note that the commonly solved tasks for Table 3 and Table 8 differ, so times and expansions in those tables are not comparable. Since using FPALs is always beneficial, and Π^{s1}_{a1} is preferable to the other three compilations, we use Π^{s1}_{a1} enhanced with FPALs for all experiments below.

7.4 Using Different Heuristics

Beyond $h^{\max}$, we evaluated several heuristics spanning a diverse range of approaches to heuristic search. These include the blind heuristic (h^0), the $h^{\text{LM-cut}}$ heuristic (Helmert and Domshlak 2009) and the admissible landmark-based h^{BJOLP} heuristic (Domshlak et al. 2011). As a representative of a state-of-the-art heuristic for optimal classical planning, we considered saturated cost partitioning (Seipp, Keller, et al. 2020) over pattern database heuristics (Edelkamp 2001) and Cartesian abstractions (Seipp and Helmert 2018), referred to as h^{SCP}.

The evaluated heuristics can be categorized based on their trade-off between informativeness and computational cost. The blind heuristic (h^0) is fast to evaluate but wholly uninformed. The $h^{\max}$ heuristic strikes a balance, being semi-informed and moderately fast to evaluate. In contrast, h^{SCP}, $h^{\text{LM-cut}}$ and h^{BJOLP} are highly informed but require significantly greater computational effort.

For the Π^{s1}_{a1} reformulation, augmented with fix-point plan action landmarks, h^{SCP} achieves the highest coverage in all domains among all heuristics. However, $h^{\max}$ solves only one fewer task than h^{SCP} (in the Termes domain) while requiring nearly two orders of magnitude less time for the tasks they both solve. To provide a closer comparison, Table 9 highlights results for three representative heuristics: h^0, $h^{\max}$ and h^{SCP}. We do not include $h^{\text{LM-cut}}$ and h^{BJOLP} in the table, as neither solves any task not solved by $h^{\max}$ and h^{SCP}. The results reveal a trade-off between planning time and heuristic informativeness. While h^0 requires the least planning time for the commonly solved tasks, h^{SCP} significantly reduces the number of node expansions. This reduction, however, comes at the cost of the high computational overhead required to compute and evaluate h^{SCP}. For simpler problems, the additional computation time of informed heuristics like h^{SCP} may not be justified. However, for more complex problems, only the additional information provided by such heuristics leads to finding a solution.

7.5 Comparison to WPMaxSAT

We now compare the Π^{s1}_{a1} planning compilation to the WPMaxSAT approach (Balyo, Chrpa, et al. 2014), described in Section 3.2. We also highlight a particular plan characteristic where the WPMaxSAT approach struggles, and

Table 9. Per-domain statistics for Π^{s1}_{a1} enhanced with fix-point plan action landmarks with three different heuristics: h^0, $h^{\max}$ and h^{SCP}. For each heuristic, we show coverage (C), time needed to solve all tasks (T), geometric mean of evaluated nodes per second (ER), and geometric mean of expanded nodes (E). For domains where all instances need at most 100 evaluations, we do not show ER. All results are computed over all commonly solved tasks.

	h^0				$h^{\max}$				h^{SCP}			
Domain	C	T	ER	E	C	T	ER	E	C	T	ER	E
Agricola	18	**0.47**	–	2.00	18	0.50	–	2.00	18	8.92	–	2.00
Barman	57	**2.07**	**99568.9**	104.77	57	2.09	34606.4	45.11	57	78.64	11784.9	**31.42**
ChildSnack	25	**0.63**	–	6.92	25	0.68	–	6.49	25	28.48	–	**5.90**
DataNetwork	34	8.94	**288314.3**	100.41	34	**1.86**	57717.5	24.09	34	47.42	29565.8	**23.21**
Floortile	47	**1.31**	–	32.07	47	1.45	–	18.18	47	55.34	–	**15.33**
GED	68	**1.82**	**490636.2**	3.13	68	2.64	19100.0	2.55	68	74.58	10064.2	**2.53**
Hiking	37	6.59	**516668.7**	10.15	**38**	**2.38**	114388.3	5.80	**38**	43.52	49863.5	**5.29**
OpenStacks	53	**1.50**	–	2.00	53	1.79	–	2.00	53	92.87	–	2.00
OrgSynthesis	9	0.25	–	2.00	9	**0.24**	–	2.00	9	6.71	–	2.00
OrgSynthesisSplit	21	**0.56**	–	2.00	21	0.59	–	2.00	21	16.16	–	2.00
Parking	45	1.22	–	3.57	45	**1.17**	–	3.14	45	53.39	–	**3.10**
Snake	26	**0.66**	–	2.00	26	0.82	–	2.00	26	27.59	–	2.00
Termes	20	3.21	**463916.1**	1709.28	28	**1.31**	80664.3	244.19	**29**	20.65	14374.6	**148.08**
Tetris	36	41.83	**350780.8**	76.11	**37**	**0.95**	29209.9	**10.04**	**37**	45.14	47046.2	19.75
Thoughtful	69	**2.25**	**296896.2**	6.28	69	2.39	38804.6	4.84	69	82.53	11331.0	**4.50**
Transport	53	5.62	**332304.4**	381.02	53	**1.75**	47618.0	53.52	53	63.14	7729.2	**33.32**
VisitAll	60	12.45	**78784.9**	15.67	60	**6.51**	11441.9	**5.74**	60	774.82	38027.8	14.84
Overall	678	91.39	**275706.3**	14.25	688	**29.13**	39176.6	7.46	**689**	1519.89	19425.2	**7.38**

confirm it empirically using a slightly modified VisitAll domain. Throughout this section we use Π^{s1}_{a1} enhanced with fix-point plan action landmarks and the $h^{\max}$ heuristic.

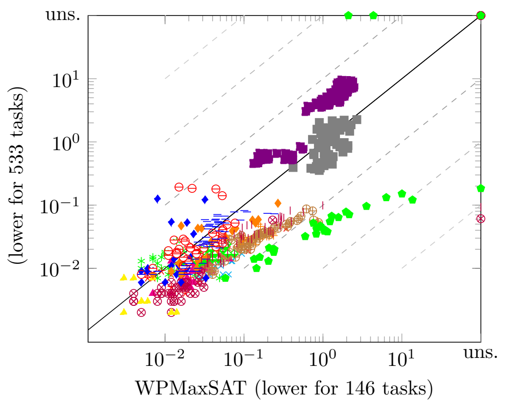
7.5.1 *Solving Time and Coverage.*

Figure 7 compares the time it takes Π^{s1}_{a1} and WPMaxSAT to solve the minimal reduction tasks. Figure 7a compares the time needed to create the tasks for both approaches, Figure 7b compares the time needed to solve the tasks, and Figure 7c compares the sum of task creation and solving times. We see that Π^{s1}_{a1} is faster in most cases, though there are cases where WPMaxSAT manages to create the task faster. These cases consist mainly of tasks where the input planning task has a high number of fix-point plan action landmarks (like in VisitAll and OpenStacks), since the time needed to compute them is accounted for in the reformulation time for Π^{s1}_{a1}.

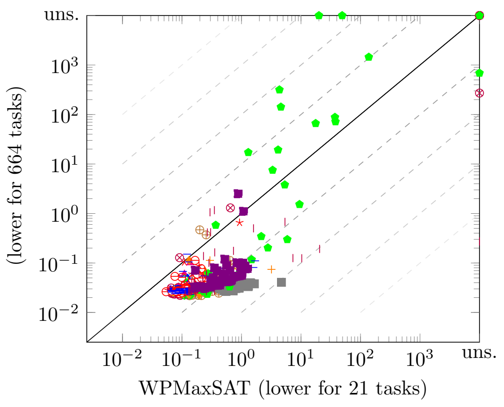
Regarding the number of solved tasks, both approaches perform similarly. Π^{s1}_{a1} solves 688 out of 700 tasks, while WPMaxSAT solves 687. Π^{s1}_{a1} solves one more task in the Transport and Hiking domains, while WPMaxSAT solves one more task in the Termes domain.

7.5.2 *Modified VisitAll.*

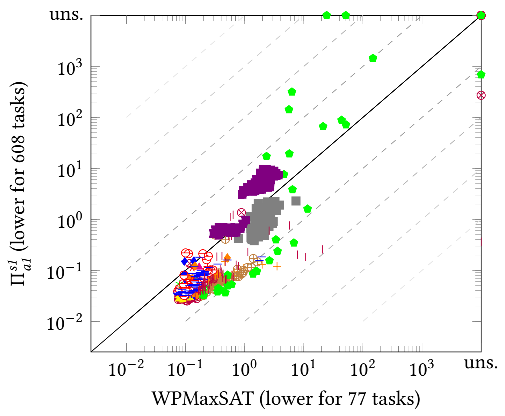
We now present a family of minimal reduction tasks where the CNF formula created by the WPMaxSAT approach grows drastically, while the planning approaches maintain the same number of actions.



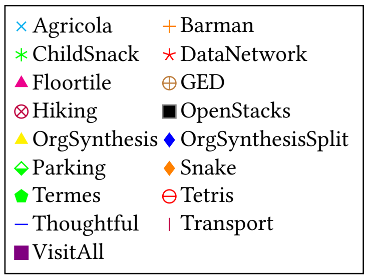
(a) Reformulation Time (s)



(b) Solving Time (s)



(c) Total Time (s)



(d) Legend

Fig. 7. Scatter plots comparing time needed to: (a) create the minimal reduction task, (b) solve the task and (c) total time (sum of task creation and solving time).

In Appendix A, we show how the CNF formula proposed by Balyo, Chrpa, et al. 2014 is created. The number of variables is at most quadratic in the plan length n, while the number of clauses is at most cubic in n (Balyo, Chrpa, et al. 2014). The number of clauses depends both on the plan length and the number of achievers and opposers of the required facts. For the number of clauses to grow to the worst case, the input plan needs to contain n different options to generate each required fact (all actions in the plan are achievers) and n opposing actions for each fact (all actions in the plan are opposers). Therefore, the formulas generated for input plans where multiple actions require to use an exclusive fact, or consume a fact that needs to be restored with a different action (or actions) will be particularly large. Formula (1) shows that if a fact needs to be true at a given step of the plan, at least one of the actions that achieve said fact before the time step must be true. In the worst case, all actions achieve a specific fact, and this clause can have as many as n variables, with n being the plan length. Formula (2) shows that, in order for a fact to be achieved by a specific action at a certain time step i, none of the actions that oppose that fact must be part of the plan reduction between the action and the time step i. In the worst case, all actions oppose a fact achieved by an action, so this clause can have as many as n variables. Formula (4) declares that, if an action at time step i is part of the plan reduction, then all of its preconditions must be true at time i. For this, for each precondition p of the action, Formula (1) is added, and for each achiever of p Formula (2) is also added. If all actions are achievers and opposers of every fact, in the worst case, $n + n^2$ clauses are added to the CNF for each precondition of every action. Assuming the number of preconditions and effects is constant in the plan length, this translates into a cubic number of clauses, as noted by Balyo, Chrpa, et al. 2014. For plans with thousands of actions, this worst-case scenario translates into billions of clauses. In contrast, the size of the Π^{s1}_{a1} compilation does not depend on the number of achievers or opposers of facts in the plan; it is only dependent (linear) on the size of the input plan.

To simulate a similar situation to this worst case scenario, we slightly modify the VisitAll domain. In the original domain, an agent is placed in a grid and must visit all cells in the grid. The actions in the domain allow the agent to move from its current position to any adjacent cell. Our modification consists of adding a resource (fuel) that is consumed after each move: each move consumes the fuel and a refuel action can be executed at any point in the grid. With this modification, all move actions will be opposers of all other (they all consume the available fuel, which is needed to move). Also, for the fuel precondition there will be as many achievers as there are previous actions in the plan (a valid plan needs to refuel after every move action). We modify the input plans by adding a refuel action between move actions to create valid plans for this modified domain (this also makes the plans double in size) and run both Π^{s1}_{a1} using FPALs, macro-actions of FPALs and $h^{\max}$ and the WPMaxSAT approach. Π^{s1}_{a1} shows a similar behavior as for the original plans and solves all tasks, while WPMaxSAT runs out of memory for each task while creating the CNF. This indicates that, for plans where each fact has many achieving and opposing actions, using the planning reformulation might be preferable over the WPMaxSAT when searching for a minimal reduction.

8 Conclusions

In this work, we presented several approaches to find minimal plan reductions using classical planning, as well as a review of existing work related to redundant action detection in automated planning. Similarly, a theoretical analysis of trivial and fix-point plan action landmarks was presented, showing that both must be part of any plan reduction of a plan. Additionally, we introduced the concept of always redundant actions, which are actions that are not part of any perfectly justified plan reduction.

Experimentally, we compared all planning compilations and found that, in general, Π^{s1}_{a1} is preferable to the other approaches, though there are cases where Π^{sm}_{a1} and Π^{s1}_{am} outperform it. We looked at the prevalence of trivial and fix-point plan action landmarks, and found that they are highly prevalent in our benchmark set. This indicates that, in many cases, one can identify most actions that must be part of a plan reduction in polynomial time. The same study was done for always redundant actions, and we found that they are not present in our benchmark set, but we illustrated settings where they occur. The performance of different heuristics when solving the minimal reduction problem with the different planning compilations indicates that, in the vast majority of cases, a simple heuristic is preferable over a more informative one.

A comparison to the WPMaxSAT approach shows that Π^{s1}_{a1} is faster in most cases, while solving roughly as many tasks. Finally, we proposed a simple modification to the VisitAlldomain to empirically show a particular type of tasks where the WPMaxSAT approach struggles. The experiments using this domain showed that, under these conditions, the formula created by the WPMaxSAT grows too fast to be able to find minimal reductions of long plans, while the planning approaches are able to solve them.

In the future, we plan to study the impact of using our action elimination techniques inside of anytime planners that iteratively reduce the last found plan before a new search. Furthermore, since our techniques are general, they can be applied to other planning settings where the solutions are sequences of actions, such as temporal, numeric or conformant planning. While we focused on obtaining *minimum* plan reductions, our techniques can also be applied to find *any* plan reduction, simply by running a satisficing instead of an optimal planner on the reformulations. Finally, to solve the minimal reduction problem, we maintain the relative order of actions. However, it is easy to lift this restriction to solve the problem of finding the cheapest plan that also uses only actions from the input plan, but possibly in a different order.

Acknowledgments

This work was partially funded by grant PID2021-127647NB-C21 from MCIN/AEI/10.13039/501100011033, by the ERDF "A way of making Europe", and by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation). Furthermore, this work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

C. Bäckström and B. Nebel. 1995. "Complexity Results for SAS$^+$ Planning." 11, 4, 625–655.

T. Balyo, L. Chrpa, and A. Kilani. 2014. "On Different Strategies for Eliminating Redundant Actions from Plans." In: *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*. Ed. by S. Edelkamp and R. Barták. AAAI Press, 10–18.

T. Balyo and S. Gocht. 2018. "The Freelunch Planning System Entering IPC 2018." In: *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 51–54.

P. Bercher, P. Haslum, and C. Muise. 2024. "A Survey on Plan Optimization." In: *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*. Ed. by K. Larson. IJCAI, 7941–7950.

B. Bonet and H. Geffner. 2001. "Planning as Heuristic Search." 129, 1, 5–33.

B. Bonet and H. Geffner. 2000. "Planning with Incomplete Information as Heuristic Search in Belief Space." In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*. Ed. by S. Chien, S. Kambhampati, and C. A. Knoblock. AAAI Press, 52–61.

T. Bylander. 1994. "The Computational Complexity of Propositional STRIPS Planning." 69, 1–2, 165–204.

L. Chrpa, T. L. McCluskey, and H. Osborne. 2012a. "Determining Redundant Actions in Sequential Plans." In: *Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012)*. IEEE, 484–491.

L. Chrpa, T. L. McCluskey, and H. Osborne. 2012b. "Optimizing Plans through Analysis of Action Dependencies and Independencies." In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. Ed. by L. McCluskey, B. Williams, J. R. Silva, and B. Bonet. AAAI Press, 338–342.

C. Dawson and L. Siklóssy. 1977. "The Role of Preprocessing in Problem Solving Systems." In: *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI 1977)*. Ed. by R. Reddy. William Kaufmann, 465–471.

C. Domshlak, M. Helmert, E. Karpas, E. Keyder, S. Richter, G. Röger, J. Seipp, and M. Westphal. 2011. "BJOLP: The Big Joint Optimal Landmarks Planner." In: *IPC 2011 Planner Abstracts*, 91–95.

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

R. E. Fikes, P. E. Hart, and N. J. Nilsson. 1972. "Learning and Executing Generalized Robot Plans." 3, 251–288.

E. Fink and Q. Yang. 1992. "Formalizing Plan Justifications." In: *Proceedings of the Ninth Conference of the Society for Computational Studies of Intelligence*, 9–14.

G. Francès, H. Geffner, N. Lipovetzky, and M. Ramiréz. 2018. "Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants." In: *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 23–27.

M. Helmert. 2006. "The Fast Downward Planning System." 26, 191–246.

M. Helmert and C. Domshlak. 2009. "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*. Ed. by A. Gerevini, A. Howe, A. Cesta, and I. Refanidis. AAAI Press, 162–169.

J. Hoffmann, J. Porteous, and L. Sebastia. 2004. "Ordered Landmarks in Planning." 22, 215–278.

S. Jiménez Celorrio, P. Haslum, and S. Thiébaux. 2013. “Pruning bad quality causal links in sequential satisfying planning.” In: *ICAPS 2013 Workshop on Planning and Learning*, 45–52.

E. Karpas and C. Domshlak. 2011. “Living on the Edge: Safe Search with Unsafe Heuristics.” In: *ICAPS 2011 Workshop on Heuristics for Domain-Independent Planning (HDIP)*, 53–58.

M. Katz. 2018. “Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection.” In: *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 47–51.

M. Katz and S. Sohrabi. 2022. “Who Needs These Operators Anyway: Top Quality Planning with Operator Subset Criteria.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*. Ed. by S. Thiébaux and W. Yeoh. AAAI Press.

M. Katz, S. Sohrabi, and O. Udrea. 2022. “Bounding Quality in Diverse Planning.” In: *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*. Ed. by V. Honavar and M. Spaan. AAAI Press.

M. Katz, S. Sohrabi, and O. Udrea. 2020. “Top-Quality Planning: Finding Practically Useful Sets of Best Plans.” In: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. Ed. by V. Conitzer and F. Sha. AAAI Press, 9900–9907.

M. Katz, S. Sohrabi, O. Udrea, and D. Winterer. 2018. “A Novel Iterative Approach to Top-k Planning.” In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. Ed. by M. de Weerdt, S. Koenig, G. Röger, and M. Spaan. AAAI Press, 132–140.

R. E. Korf. 1985. “Depth-First Iterative-Deepening: An Optimal Admissible Tree Search.” 27, 1, 97–109.

D. Le Berre and A. Parrain. 2010. “The Sat4j library, release 2.2.” *Journal on Satisfiability, Boolean Modeling and Computation*, 7, 2–3, 59–64.

N. Lipovetzky and H. Geffner. 2017. “Best-First Width Search: Exploration and Exploitation in Classical Planning.” In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*. Ed. by S. Singh and S. Markovitch. AAAI Press, 3590–3596.

D. A. McAllester and D. Rosenblitt. 1991. “Systematic Nonlinear Planning.” In: *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*. Vol. 2. AAAI Press, 634–639.

J. Med and L. Chrpa. 2022. “On Speeding Up Methods for Identifying Redundant Actions in Plans.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*. Ed. by S. Thiébaux and W. Yeoh. AAAI Press, 252–260.

C. Muise, J. C. Beck, and S. A. McIlraith. 2016. “Optimal Partial-Order Plan Relaxation via MaxSAT.” 57, 113–149.

H. Nakhost and M. Müller. 2010. “Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement.” In: *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*. Ed. by R. Brafman, H. Geffner, J. Hoffmann, and H. Kautz. AAAI Press, 121–128.

B. Nebel, Y. Dimopoulos, and J. Koehler. 1997. “Ignoring Irrelevant Facts and Operators in Plan Generation.” In: *Recent Advances in AI Planning. 4th European Conference on Planning (ECP 1997)*. Ed. by S. Steel and R. Alami. Vol. 1348. Springer-Verlag, 338–350.

C. Olz and P. Bercher. 2019. “Eliminating Redundant Actions in Partially Ordered Plans – A Complexity Analysis.” In: *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*. Ed. by N. Lipovetzky, E. Onaindia, and D. E. Smith. AAAI Press, 310–319.

S. Richter and M. Westphal. 2010. “The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks.” 39, 127–177.

S. Richter, M. Westphal, and M. Helmert. 2011. “LAMA 2008 and 2011 (planner abstract).” In: *IPC 2011 Planner Abstracts*, 50–54.

M. Salerno, R. Fuentetaja, and J. Seipp. 2025. *Code and Data for the Article “Finding Minimal Plan Reductions Using Classical Planning”*. https://doi.org/10.5281/zenodo.14065819. (2025).

M. Salerno, R. Fuentetaja, and J. Seipp. 2023a. “Eliminating Redundant Actions from Plans using Classical Planning.” In: *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. Ed. by P. Marquis, T. C. Son, and G. Kern-Isberner. IJCAI Organization, 774–778.

M. Salerno, R. Fuentetaja, and J. Seipp. 2023b. “Spock: Fast Downward Stone Soup with Redundant Action Elimination.” In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

B. Say, A. A. Cire, and J. C. Beck. 2016. “Mathematical Programming Models for Optimizing Partial-Order Plan Flexibility.” In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. Ed. by G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen. IOS Press, 1044–1052.

J. Seipp and M. Helmert. 2018. “Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning.” 62, 535–577.

J. Seipp, T. Keller, and M. Helmert. 2020. “Saturated Cost Partitioning for Optimal Classical Planning.” 67, 129–167.

F. H. Siddiqui and P. Haslum. 2015. “Continuing Plan Quality Optimisation.” 54, 369–435.

J. Slaney and S. Thiébaux. 2001. “Blocks World revisited.” 125, 1–2, 119–153.

D. Speck, R. Mattmüller, and B. Nebel. 2020. “Symbolic Top-k Planning.” In: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. Ed. by V. Conitzer and F. Sha. AAAI Press, 9967–9974.

S. Sreedharan, C. Muise, and S. Kambhampati. 2023. “Generalizing Action Justification and Causal Links to Policies.” In: *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. Ed. by S. Koenig, R. Stern, and M. Vallati. AAAI Press, 417–426.

B. Srivastava, T. A. Nguyen, A. Gerevini, S. Kambhampati, M. B. Do, and I. Serina. 2007. "Domain Independent Approaches for Finding Diverse Plans." In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*. Ed. by M. M. Veloso, 2016–2022.

V. Vidal. 2014. "YAHSP3 and YAHSP3-MT in the 8th International Planning Competition." In: *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 64–65.

M. Waters, L. Padgham, and S. Sardina. 2020. "Optimising Partial-Order Plans Via Action Reinstantiation." In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*. IJCAI, 4143–4151.

A WPMaxSAT Formulation

Below, we summarize how the WPMaxSAT formula by Balyo, Chrpa, et al. 2014 is constructed, in order to show how the characteristics of the planning task affect it. To encode the minimal reduction problem as a SAT problem, the following variables are defined:

- a_i for $1 \leq i \leq n$: a_i is used in the plan reduction.
- $y_{p,i,k}$ for $1 \leq k < i \leq n$: plan action a_k is responsible for achieving fact p at step i in the plan reduction.
- $y_{p,i,0}$: the initial state is responsible for achieving fact p at step i in the plan reduction.

Using these variables, the formula $F_{\Pi,\pi}$ is constructed as a conjunction of clauses representing which actions from the input plan are applied. For each action in the input plan, clauses are introduced to guarantee that all their preconditions hold when the action is applied. To formulate $F_{\Pi,\pi}$, the authors consider *achievers* and *opposers*. An action is an *achiever* of a fact f if $f \in \mathit{eff}(a)$. Similarly, an action is an *opposer* of a fact $v \mapsto d$ if $v \mapsto d' \in \mathit{eff}(a)$, with $d' \neq d$. With these notions, the sets of achievers and opposers can be defined as:

- $\mathit{Achs}(p, i)$: set of plan positions j of the achievers of fact p with $j < i$.
- $\mathit{Opps}(p, i, j)$: set of plan positions k of opposers of fact p with $i \leq k \leq j$.

Using the achievers and opposers, the authors define the following formulas:

- $F_{p,i}$: if a fact p is required to be true at step i in the plan reduction, then it must be generated by the initial state or by at least one of its achievers with plan positions smaller than i:

$$F_{p,i} = y_{p,i,0} \bigvee_{j \in \mathit{Achs}(p,i)} y_{p,i,j} \tag{1}$$

- $F_{p,i,j}$: for $1 \leq j \leq n$ encodes that, if a plan action a_j is responsible for achieving fact p at time i in the plan reduction, then a_j must belong to the plan reduction, and that none of the opposing actions between j and i belong to the plan reduction:

$$F_{p,i,j} = (\neg y_{p,i,j} \vee a_j) \bigwedge_{k \in \mathit{Opps}(p,j,i)} (\neg y_{p,i,j} \vee \neg a_k) \tag{2}$$

- $F_{p,i,0}$: the initial state is responsible for achieving fact p at time i:

$$F_{p,i,0} = \bigwedge_{k \in \mathit{Opps}(p,0,i)} (\neg y_{p,i,0} \vee \neg a_k) \tag{3}$$

- F_{a_i}: if a_i is an action of the plan reduction, then all its preconditions are required to be true at time i:

$$F_{a_i} = \bigwedge_{p \in pre(a_i)} \left((\neg a_i \vee F_{p,i}) \wedge F_{p,i,0} \bigwedge_{j \in \mathit{Achs}(p,i)} F_{p,i,j} \right) \tag{4}$$

- F_G: all goal conditions must be true in the end of the plan:

$$F_G = \bigwedge_{p \in \mathcal{G}} \left(F_{p,i} \wedge F_{p,i,0} \bigwedge_{j \in \mathit{Achs}(p,n)} F_{p,n,j} \right) \tag{5}$$

Then $F_{\Pi,\pi}$ is defined as:

$$F_{\Pi,\pi} = F_G \bigwedge_{i=1,\ldots,|\pi|} F_{a_i} \tag{6}$$

Clearly, every satisfying assignment ϕ of $F_{\Pi,\pi}$ constitutes a plan reduction π_ϕ of π, where an action a_i is present in the plan reduction if $\phi(a_i) = \top$.

To solve the minimal reduction problem using $F_{\Pi,\pi}$, the authors use a weighted partial MaxSAT (WPMaxSAT) formula and a WPMaxSAT solver. WPMaxSAT problems have *hard* and *soft* clauses. In a solution to a WPMaxSAT problem, all hard clauses must be satisfied, while soft clauses can be true or false. Each soft clause has a weight associated with it, and the goal is to satisfy the hard clauses while maximizing the summed weight of the satisfied soft clauses. The hard clauses are defined as $H_{\Pi,\pi} = F_{\Pi,\pi}$; while the soft clauses are unit clauses with negated action variables, $S_{\Pi,\pi} = \bigwedge_{a_i \in \Pi} \neg a_i$. The weight of each soft clause is the cost of the corresponding action. Thus, maximizing the weight of the satisfied soft clauses is equivalent to removing actions with maximal cost. To deal with potential redundant zero-cost actions remaining after solving the WPMaxSAT, a minimal *length* reduction (same scheme but with unitary action costs) is performed on the resulting plan reduction.

Received 07 June 2025; accepted 04 September 2025