Eliminating Redundant Actions from Plans using Classical Planning

Mauricio Salerno[1], **Raquel Fuentetaja**[1], **Jendrik Seipp**[2]
[1]Universidad Carlos III de Madrid, Leganés, Madrid, Spain
[2]Linköping University, Linköping, Sweden
msalerno@pa.uc3m.es, rfuentet@inf.uc3m.es, jendrik.seipp@liu.se

Abstract

Even though automated planning is PSPACE-complete in general, satisficing planners are able to solve large planning tasks quickly. However, the found plans are often far from optimal and may even contain actions that can be removed while maintaining a valid plan. The problem of finding and eliminating the most expensive set of such redundant actions in a plan is NP-complete and there is a compilation to MaxSAT that solves it. Here, we introduce a simple and natural formulation of the problem as a planning task. Solving it with an optimal planner guarantees finding a minimal reduction. Our experiments show that this is competitive with the previous state of the art for optimal action elimination.

1 Introduction

Today's satisficing planners are able to solve large planning tasks efficiently (Richter and Westphal 2010; Lipovetzky and Geffner 2017). However, their solutions often include *redundant actions*. A subsequence of actions in a plan is redundant if it can be removed without invalidating the plan, and a plan without redundant subsequences is called *perfectly justified* (Fink and Yang 1992; Nebel, Dimopoulos, and Koehler 1997). Finding justified plans is important in settings such as top-k planning (Katz et al. 2018; Katz, Sohrabi, and Udrea 2020; Speck, Mattmüller, and Nebel 2020), especially when diversity is required (Srivastava et al. 2007; Katz, Sohrabi, and Udrea 2022). 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 useful plans. Filtering redundant actions from plans can also help anytime planners to generate better solutions earlier. However, checking whether a plan is perfectly justified is NP-complete (Fink and Yang 1992; Nakhost and Müller 2010). Thus, it is important to develop efficient filtering techniques.

In spirit of previous work, we focus on filtering redundant actions from plans in a post-planning step, while preserving the order of the remaining actions (Fink and Yang 1992; Nakhost and Müller 2010; Chrpa, McCluskey, and Osborne 2012b,a; Balyo, Chrpa, and Kilani 2014). Given a plan for a planning task, we propose an automatic reformulation to a new planning task whose optimal solution is a *minimal reduction* of the original plan. A minimal reduction is a perfectly justified subplan that additionally has the lowest possible cost. Our paper is closely related to the work by Balyo, Chrpa, and Kilani (2014), who solve the same problem with a weighted MaxSAT encoding. Other works focused on suboptimal action elimination (Chrpa, McCluskey, and Osborne 2012b,a; Med and Chrpa 2022). Action elimination also reduces the plan length or cost, which is a welcome byproduct, but our main motivation is to find useful plans without redundant actions. 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 (Siddiqui and Haslum 2015; Muise, Beck, and McIlraith 2016; Say, Cire, and Beck 2016; Olz and Bercher 2019; Waters, Padgham, and Sardina 2021).

2 Classical Planning and Plan Justification

We consider classical planning tasks in the SAS$^+$ formalism (Bäckström and Nebel 1995). 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*, $\mathcal{A}$ is a finite set of *actions*, $\mathcal{I}$ is an *initial state*, and $\mathcal{G}$ is a *goal description*. Each variable $v \in \mathcal{V}$ has a finite domain $\mathcal{D}(v)$ of values. Each pair $\langle v, d \rangle$, where $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$, is a *fact*. A *partial state* s is a mapping of a subset of variables $vars(s) \subseteq \mathcal{V}$ to values in their domains. We write $s[v] \in \mathcal{D}(v)$ to denote the value of variable $v \in vars(s)$ in the partial state s. Where convenient, we treat partial states as sets of facts. A *state* s is a partial state that maps *all* variables to a value, i.e., $vars(s) = \mathcal{V}$. $\mathcal{I}$ is a state and $\mathcal{G}$ is a partial state. *Goal states* are those states s for which $\mathcal{G} \subseteq s$. 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 a state s if $pre(a) \subseteq s$. The successor state $s[\![a]\!]$ that results from applying a in s is defined as $s[\![a]\!][v] = eff(a)[v]$ if $v \in vars(eff(a))$ and as $s[\![a]\!][v] = s[v]$ otherwise. A *solution* or *plan* for Π is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that 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 $|\pi| = n$. Each action $a \in \mathcal{A}$ has a cost $c(a) \in \mathbb{N}_0$, so that 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.

The notion of plan *justification* can be traced back to the early 1990s. Fink and Yang (1992) define three types of plan justifications: *backward* justification, *well*-justification and *perfect* justification. Perfectly-justified plans are those for which no subsequence of actions can be removed from the plan without invalidating it. We consider this latter variant, and now introduce it formally.

Definition 1 (Plan Reduction). *Let π be a plan for a planning task Π and ρ be a subsequence of π with $|\rho| < |\pi|$. ρ is a plan reduction of π if and only if ρ is also a plan for Π.*

Definition 2 (Perfectly-Justified Plan). *A plan π for the planning task Π is perfectly justified if and only if there is no plan reduction of π.*

Thus, if there is at least one plan reduction, the plan is not perfectly justified. Given a task Π and a plan π for Π, the task of finding the cheapest perfectly justified plan reduction of π is called Minimal Reduction (MR) (Nakhost and Müller 2010; Balyo, Chrpa, and Kilani 2014).

3 Minimal Reduction as Planning

We now introduce a compilation that encodes the MR problem as a planning task. Given a planning task and a plan, we define a new planning task that can eliminate subsequences of redundant actions from the plan. For that, we encode the new task in a way that allows to either keep or skip each plan action while preserving the order of the actions. If we were only interested in finding a cheaper plan, the action order could be omitted. However, since we are interested in MR, maintaining the action order is crucial.

In the following, we consider the same action occurring at different positions in an action sequence as different actions. For this, we slightly abuse notation and let $a_i \in \pi$ represent that action a_i is at position i in π.

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_1, \dots, a_n \rangle$ be a plan for Π. We define F_π as the set of facts that appear in a precondition of the actions in the plan π or the goal: $F_\pi = \bigcup_{a_i \in \pi} pre(a_i) \cup \mathcal{G}$. Then, the new planning task $\Pi^{skip} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ is defined as:

- $\mathcal{V}' = \{v \in \mathcal{V} \mid |\mathcal{D}'(v)| > 1\} \cup \{pos\}$, where we keep variables v from the original task but with a redefined domain $\mathcal{D}'(v)$, and only when $\mathcal{D}'(v)$ contains more than one value; and there is an additional variable pos with $\mathcal{D}(pos) = \{0, \dots, n\}$ to track the current position in the original plan. For a variable v, $\mathcal{D}'(v)$ contains only the values of v that appear in F_π and an additional value θ, representing an *irrelevant value*. The value of a variable is irrelevant when it is set by either the initial state or the effect of some action in the plan, but the corresponding fact is not in F_π. Thus, $\mathcal{D}'(v) = \{d \in \mathcal{D}(v) \mid \langle v, d \rangle \in F_\pi\} \cup \Theta$, where $\Theta = \{\theta\}$ if $\langle v, d \rangle \notin F_\pi$ but $\langle v, d \rangle \in \mathcal{I}$ or there exists $a_i \in \pi$ with $\langle v, d \rangle \in \mathit{eff}(a_i)$, and $\Theta = \emptyset$ otherwise. The irrelevant value is used when the value of a variable is changed to a value which is not relevant to the plan. Just removing such effects is not enough because the variable does not maintain its previous value after the application of the action. This potentially reduces the size of $\mathcal{D}'(v)$ compared to $\mathcal{D}(v)$ since all facts in the effects of any action in π but not in F_π are represented by the single fact $\langle v, \theta \rangle$.
- $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\} \cup \{skip_i \mid 1 \leq i \leq n\}$, where there is a new action a'_i for every action a_i in the plan and a skip action for every plan position. Actions a'_i are defined as: $pre(a'_i) = pre(a_i) \cup \{\langle pos, i-1 \rangle\}$ and $\mathit{eff}(a'_i) = \{\tau(\langle v, d \rangle) \mid \langle v, d \rangle \in \mathit{eff}(a_i), v \in \mathcal{V}'\} \cup \{\langle pos, i \rangle\}$, where the effects of the new action are the same as those of the original action, but changing the variable value to the irrelevant one for those facts not used in action preconditions or goals. Formally, $\tau(\langle v, d \rangle)$ is $\langle v, d \rangle$ if $\langle v, d \rangle \in F_\pi$ and $\langle v, \theta \rangle$ otherwise. The $skip_i$ actions just increase the value of pos from $i-1$ to i. They have zero cost, while the a'_i actions maintain the cost $c(a_i)$.
- $\mathcal{I}' = (\mathcal{I} \cap F_\pi) \cup \{\langle v, \theta \rangle \mid v \in \mathcal{V}', \langle v, d \rangle \in (\mathcal{I} \setminus F_\pi)\} \cup \{\langle pos, 0 \rangle\}$ contains the facts from the original initial state that are relevant for the plan, and relevant variables with an irrelevant initial value are set to the *irrelevant value*. The pos variable is initialized to zero.
- $\mathcal{G}' = \mathcal{G} \cup \{\langle pos, n \rangle\}$ contains the original goals and requires the pos variable to be at the end of the original plan (this could be omitted but it can be useful for heuristics).

Plans for Π^{skip} 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 $i-1$). Therefore, there is a one-to-one correspondence between the actions in a plan π' for Π^{skip} and the actions in the plan π for Π, defined by the action positions. Consequently, it is straightforward to transform a plan for Π^{skip} into a plan for Π, i.e., by removing skip actions and replacing the remaining actions by their counterparts in Π.

The plan obtained from solving Π^{skip} can contain redundant zero-cost actions. 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 $mincost = 0$, f is undefined but also unneeded. This factor satisfies $f \cdot m < mincost$, which guarantees that removing any action with a cost greater than zero will be more beneficial than removing any set of zero-cost actions. With this modification, an optimal plan for Π^{skip} is a MR for π.

4 Plan Action Landmarks

Identifying if *all* subsequences of actions in a plan are necessary is NP-hard (Fink and Yang 1992). However, this does not mean that we cannot identify *some* actions as necessary in polynomial time. Med and Chrpa (2022) defined *plan action landmarks* as actions that must be part of any plan reduction of a given plan. For example, if only a single action a 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. They proposed an algorithm to compute this specific type of plan actions landmarks in linear time, and we call them *trivial plan action landmarks* (TPAL). To simplify the formal definition of trivial plan action landmarks, we extend a given plan with *virtual initial* and *goal* actions, defined as $a_0 = \langle \emptyset, \mathcal{I} \rangle$ and $a_{n+1} = \langle \mathcal{G}, \emptyset \rangle$, respectively. (Action a_{n+1} does not need effects for our purposes.)

Definition 3 (Trivial Plan Action Landmark, TPAL). *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ be a plan for Π extended with virtual initial and goal actions. Action a_i is a trivial plan action landmark iff: (1) $i = n+1$ (goal action) or (2) there is a trivial plan action landmark a_j, $i < j$, such that there is a fact $p \in \mathit{eff}(a_i) \cap \mathit{pre}(a_j)$, and there is no action a_k, $k < j$, $k \neq i$, such that $p \in \mathit{eff}(a_k)$.*

We identify the set A_s of actions that can potentially be skipped as $A_s = \{a_i \in \pi \mid a_i \textit{ is not a TPAL}\}$. Then, we create an enhanced task $\Pi^{skip}_{TPAL} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G} \rangle$, where $\mathcal{V}', \mathcal{I}', \mathcal{G}$ are defined exactly as for Π^{skip}, but the set of actions $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\} \cup \{skip_i \mid a_i \in A_s\}$ only has $skip_i$ actions for actions a_i that are not TPAL.

Since a TPAL *must* be executed, we can extend our definition to take their effects into account, giving rise to the notion of *fix-point plan action landmarks* (FPAL). In essence, an action is an *FPAL* if it is the only achiever of a fact that is needed (either because it is a goal fact or part of a precondition of a FPAL) after another FPAL overwrote that fact.

Definition 4 (Fix-point Plan Action Landmark, FPAL). *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ be a plan for Π extended with virtual initial and goal actions. Action a_i is a fix-point plan action landmark iff: (1) a_i is a trivial plan action landmark or (2) there is a fix-point plan action landmark a_j, $i < j$, such that there is a fact $\langle v, d \rangle \in \mathit{eff}(a_i) \cap \mathit{pre}(a_j)$, and there is another fix-point plan action landmark a_k, $k < i$, with an effect $\langle v, d' \rangle \in \mathit{eff}(a_k)$ where $d' \neq d$ and there is no other action a_ℓ with $\ell \neq i$, $k < \ell < j$ such that $\langle v, d \rangle \in \mathit{eff}(a_\ell)$.*

Algorithm 1 computes the set of FPALs. The procedure ComputeAchievers (line 3) finds the achievers for each variable-value pair $\langle v, d \rangle$, i.e., all actions $a_i \in \pi$ such that $\langle v, d \rangle \in \mathit{eff}(a_i)$. Each achiever is a pair $\langle a_i, k \rangle$, with $i < k$, where k represents *until* which step of the plan a_i is an achiever of $\langle v, d \rangle$. For this, we initialize k to the last plan position. Then, until no new FPAL is found, ComputeFPALs iterates backwards over all plan actions. If one action is an FPAL, it checks for every precondition if a new FPAL can be identified using Definition 4: is there a unique achiever a_j that was not marked as an FPAL before? If a new FPAL a_j is identified, it is marked as such and the *until value* for the achievers of its effects are updated, since a_j overwrites them (line 11). The procedure Update finds for every $\langle v, d \rangle \in \mathit{eff}(a_j)$ the actions a_i before a_j, that set v to d', $d' \neq d$, and update their *until value* to j. The algorithm runs in polynomial time (quadratic in the number of plan actions, assuming the size of action preconditions and effects is constant wrt. the plan length). With only a single iteration of the fixpoint loop and without the Update call (line 11), it finds TPALs instead of FPALs.

Consecutive subsequences of FPALs can be safely encapsulated in a single macro action (Fikes, Hart, and Nilsson 1972). For two actions a_i and a_j, we build the macro a_{ij} as follows: $\mathit{pre}(a_{ij}) = \mathit{pre}(a_i) \cup (\mathit{pre}(a_j) \setminus \mathit{eff}(a_i))$, and $\mathit{eff}(a_{ij}) = \mathit{eff}(a_j) \cup \mathit{eff}|_{a_j}(a_i)$, where $\mathit{eff}|_{a_j}(a_i)$ represents the effects of a_i restricted to those facts whose variable does not appear in the effects of a_j: $\mathit{eff}|_{a_j}(a_i) = \{\langle v, d \rangle \mid \langle v, d \rangle \in \mathit{eff}(a_i), v \notin \mathit{vars}(\mathit{eff}(a_j))\}$. Longer sequences of macros result from iteratively applying this definition. For a macro to be *sound*, no variable in the effects of a_i can be assigned a value different than the one required for that variable in the precondition of a_j. Soundness for macros of FPALs is guaranteed since they only contain consecutive plan actions.

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:     $FPALs \leftarrow \{a_{n+1}\}$                ▷ Virtual goal action is FPAL
 4:     $achs \leftarrow$ ComputeAchievers($\pi$)         ▷ Initial achievers
 5:     repeat
 6:         for $i = n+1$ to 1 do
 7:             if $a_i \in FPALs$ then
 8:                 for $\langle v, d \rangle \in pre(a_i)$ do
 9:                     $A \leftarrow \{a_j | \langle a_j, k \rangle \in achs[v][d],\ j < i \leq k\}$
10:                     if $A = \{a_j\}$ and $a_j \notin FPALs$ then
11:                         $FPALs \leftarrow FPALs \cup \{a_j\}$
12:                         Update($achs, a_j$)
13:     until FPALs remains unchanged
14:     return $FPALs$
15:
16: procedure Update($achs, a_j$)                 ▷ Update achievers
17:     for $\langle v, d \rangle \in eff(a_j)$ do
18:         for $d' \in \mathcal{D}(v) \setminus \{d\}$ do
19:             for $(a_i, k) \in achs[v][d]$ with $i < j < k$ do
20:                 $achs[v][d'] = (achs[v][d'] \setminus \langle a_i, k \rangle) \cup \langle a_i, j \rangle$
```

5 Evaluation

We evaluate the proposed approach for MR on a set of tasks from the agile tracks of IPC 2014 and 2018 and plans found with state-of-the-art planners (700 tasks in total).[1] Executions were done on an Intel(R) Xeon(R) X3470 2.93 GHz CPU, with a time limit of 30 minutes and 8 GiB of RAM. All benchmarks, code and data are available online (Salerno, Fuentetaja, and Seipp 2023).

Table 1 shows the results of an ablation study comparing the weighted partial MaxSAT approach by Balyo, Chrpa, and Kilani (2014) to our base compilation (Π^{skip}), the compilation enhanced with TPALs (Π^{skip}_{TPAL}), FPALs (Π^{skip}_{FPAL}) and FPALs plus macros (Π^{skip}_{FPAL+M}). To solve the resulting planning tasks, we use the Fast Downward planning system with an optimal configuration (Helmert 2006), and we use Sat4J as the MaxSAT solver (Le Berre and Parrain 2010). All planner configurations use A^* with the h^{max} heuristic (Bonet and Geffner 2001), except for the last column which uses A^* with *saturated cost partitioning* (SCP) over pattern database heuristics (Seipp, Keller, and Helmert 2020). The algorithms

[1]We use the same benchmarks and input plans as Med and Chrpa (2022), removing those with conditional effects.

Domain	#	Reduction	MaxSAT	Π^{skip}	Π^{skip}_{TPAL}	Π^{skip}_{FPAL}	Π^{skip}_{FPAL+M}	Π^{skip}_{FPAL+M}SCP
Agricola	18	0% (0)	2.0 (18)	0.6 (18)	0.6 (18)	0.7 (18)	**0.4** (18)	8.2 (18)
Barman	57	7.8% (47)	18.9 (57)	14.5 (57)	4.4 (57)	4.2 (57)	**3.0** (57)	60.1 (57)
Childsnack	25	5.1% (20)	1.6 (25)	**0.5** (25)	**0.5** (25)	**0.5** (25)	0.6 (25)	25.8 (25)
Datanetwork	34	10.6% (24)	3.7 (34)	6.6 (34)	2.2 (34)	2.0 (34)	**1.8** (34)	39.0 (34)
Floortile	47	6.8% (44)	3.8 (47)	1.3 (47)	1.2 (47)	1.2 (47)	**1.0** (47)	48.3 (47)
Ged	68	9.9% (8)	22.1 (68)	3.3 (68)	3.2 (68)	3.6 (68)	2.2 (68)	72.4 (68)
Hiking	47	14.3% (14)	3.2 (37)	14.7 (37)	2.4 (**38**)	2.5 (**38**)	**2.3** (**38**)	39.2 (**38**)
Openstacks	53	0% (0)	164.1 (53)	17.3 (53)	**13.4** (53)	16.5 (53)	59.1 (53)	97.1 (53)
Org. Syn.	9	0% (0)	0.4 (9)	**0.1** (9)	**0.1** (9)	**0.1** (9)	**0.1** (9)	6.4 (9)
Org. Syn. Split	21	0% (0)	1.2 (21)	**0.7** (21)	0.9 (21)	0.9 (21)	0.8 (21)	15.4 (21)
Parking	45	2% (6)	4.6 (45)	1.7 (45)	1.8 (45)	1.9 (45)	**1.4** (45)	46.6 (45)
Snake	26	0% (0)	3.3 (26)	1.2 (26)	1.2 (26)	1.4 (26)	**0.9** (26)	27.2 (26)
Termes	31	8.7% (17)	368.1 (**29**)	3644.2 (26)	1351.4 (27)	996.2 (27)	977.8 (27)	**322.4** (**29**)
Tetris	37	8% (21)	3.2 (37)	**2.1** (37)	2.4 (37)	2.4 (37)	2.3 (37)	101.1 (37)
Thoughtful	69	6.6% (22)	8.7 (69)	4.3 (69)	3.8 (69)	3.9 (69)	**3.5** (69)	72.7 (69)
Transport	53	3.8% (43)	47.6 (52)	137.0 (**53**)	22.0 (**53**)	15.5 (53)	**11.0** (**53**)	58.9 (**53**)
Visitall	60	0.5% (17)	188.2 (60)	1024.7 (58)	622.7 (58)	304.9 (60)	**135.2** (60)	592.0 (60)
Total	700	5.4% (283)	**844.7** (687)	4875.1 (683)	2034.2 (685)	1358.5 (687)	1203.3 (687)	1632.7 (**689**)

Table 1: Comparison of algorithms for the MR problem. "#" shows the number of tasks per domain. "Reduction" is the geometric mean of the plan cost reduction ratios for plans where costs were reduced. The number of such reduced plans is given in parentheses. For each MR method, we show the sum of times needed to solve all commonly solved task per domain, and in parentheses the number of instances solved within the time and memory limits.

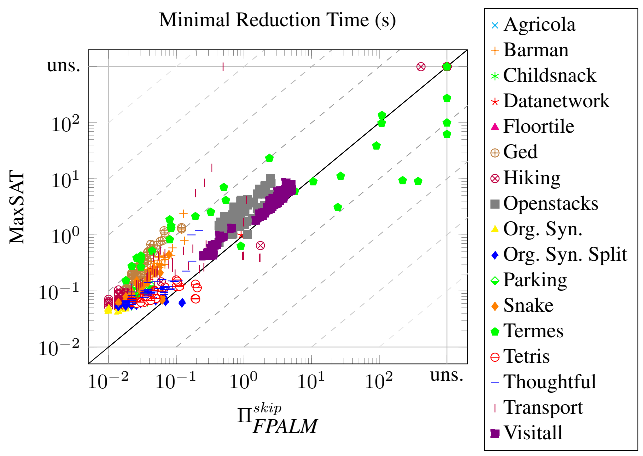

Figure 1: Scatter plot comparing the solving time of MaxSAT (y-axis) and Π^{skip}_{FPAL+M} with h^{max} as the heuristic function (x-axis). uns. means unsolved.

in Table 1 use the encoding with the *pos* variable in the goal description. Leaving it out yields almost identical results.

Π^{skip} performs considerably worse than all other approaches, as expected. The use of TPALs, FPALs and macro actions improves performance, leading to lower runtimes overall and increased coverage. Π^{skip}_{FPAL+M} yields the best results in general, except for *Termes* where it is considerably slower than MaxSAT. The use of more advanced heuristics (*SCP*) increases coverage but also increases solving time for most domains (we let the SCP planner take one second for finding pattern databases). The results for *Visitall* and *Termes* are particularly interesting: FPALs greatly reduce the solving time and increase the coverage for Visitall, but they are not as impactful on Termes. On the other hand, using SCP greatly reduces the solving time for Termes, but it has the opposite effect on Visitall. This behavior can be explained by the amount of FPALs in each domain: for Termes there are plans with more than 2000 actions that have only 5 FPALs, translating to a search tree with an average branching factor of 2 and the solutions at depths in the thousands, justifying the use of more accurate heuristics. In contrast, a large percentage of actions in plans for Visitall are FPALs. In multiple cases, all actions are identified as FPALs for plans with more than 4000 actions. This translates to average branching factors close to 1, making the tasks to solve quite easy in comparison. Figure 1 shows a comparison of the solving times for MaxSAT and Π^{skip}_{FPAL+M}. The planning approach is faster for most tasks (663 vs. 21).

6 Conclusions and Future Work

In this work, we presented an approach for eliminating redundant actions from plans based on classical planning. The experiments for solving the MR problem using an optimal planner show that the results are comparable in time with the state-of-the-art compilation to MaxSAT. The proposed approach even outperforms the MaxSAT approach in most cases, but there are domains where the MaxSAT variant is preferable. We believe this work opens interesting research avenues on investigating the use of automated planning, including different heuristics, for solving the MR problem, for which planning has not been used before; and for the use of action elimination to find perfectly-justified and cheaper plans in different settings. In the short-term future, we plan to compare anytime planners to action elimination approaches and their potential synergies.

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). Also, this work was supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

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

Balyo, T.; Chrpa, L.; and Kilani, A. 2014. On different strategies for eliminating redundant actions from plans. In *Proc. SoCS 2014*, 10–18.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1):5–33.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012a. Determining redundant actions in sequential plans. In *Proc. ICTAI 2012*, 484–491. IEEE.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012b. Optimizing plans through analysis of action dependencies and independencies. In *Proc. ICAPS 2012*, 338–342.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

Fink, E., and Yang, Q. 1992. Formalizing plan justifications. In *Proc. CSCSI 1992"*, 9–14.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *Proc. ICAPS 2018*, 132–140.

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-quality planning: Finding practically useful sets of best plans. In *Proc. AAAI 2020*, 9900–9907.

Katz, M.; Sohrabi, S.; and Udrea, O. 2022. Bounding quality in diverse planning. In *Proc. AAAI 2022*, 9805–9812.

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

Lipovetzky, N., and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *Proc. AAAI 2017*, 3590–3596.

Med, J., and Chrpa, L. 2022. On speeding up methods for identifying redundant actions in plans. In *Proc. ICAPS 2022*, 252–260.

Muise, C.; Beck, J. C.; and McIlraith, S. A. 2016. Optimal partial-order plan relaxation via MaxSAT. *JAIR* 57:113–149.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proc. ICAPS 2010*, 121–128.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. ECP 1997*, 338–350.

Olz, C., and Bercher, P. 2019. Eliminating redundant actions in partially ordered plans — a complexity analysis. In *Proc. ICAPS 2019*, 310–319.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Salerno, M.; Fuentetaja, R.; and Seipp, J. 2023. Code and data for the KR 2023 paper: "Eliminating Redundant Actions from Plans using Classical Planning". *https://doi.org/10.5281/zenodo.8018193*.

Say, B.; Cire, A. A.; and Beck, J. C. 2016. Mathematical programming models for optimizing partial-order plan flexibility. In *Proc. ECAI 2016*, 1044–1052.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated cost partitioning for optimal classical planning. *JAIR* 67:129–167.

Siddiqui, F. H., and Haslum, P. 2015. Continuing plan quality optimisation. *JAIR* 54:369–435.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic top-k planning. In *Proc. AAAI 2020*, 9967–9974.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Proc. IJCAI 2007*, 2016–2022.

Waters, M.; Padgham, L.; and Sardina, S. 2021. Optimising partial-order plans via action reinstantiation. In *Proc. IJCAI 2020*, 4143–4151.