Elimination of Unnecessary Actions from Plans Using Automated Planning

Mauricio Salerno, Raquel Fuentetaja

Department of Computer Science and Engineering
Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain
msalerno@pa.uc3m.es, rfuentet@inf.uc3m.es

Abstract

Finding solutions in Automated Planning has been proven to be intractable. However, satisficing planning approaches are capable of finding solutions to large problems relatively fast, in contrast to optimal planning. The solutions provided by satisficing planners are often far from optimal, and can even contain actions that can be removed while the plan maintains its validity. These unnecessary actions not only increase the cost of a solution, but also can render a plan irrelevant in certain settings. The problem of eliminating unnecessary or redundant actions from plans is known to be NP-complete. There are both sub-optimal (greedy) and optimal approaches to solve it. In this paper we introduce two compilations to solve this problem in a post-planning optimization step that encode the problem as an Automated Planning task. Using an optimal planner to solve the corresponding task guarantees finding the *best* set of unnecessary actions, while using a satisficing planner might find good solutions fast. The proposed approaches are empirically compared to existing approaches.

Introduction

One of the objectives in Automated Planning (AP) is to achieve a good trade-off between the time to generate plans and the quality of them. Quality of plans is typically measured using a cost function. However, quality can be also defined from the point of view of "plan relevance" or "justified plans" (Fink and Yang 1992). From a domain independent perspective, relevant plans can be understood as those that do not contain loops and do not contain irrelevant or unnecessary actions (Fink and Yang 1992; Nebel, Dimopoulos, and Koehler 1997). From a domain specific point of view more subtle notions of relevancy could be considered.

The notion of relevance was previously considered from two points of view: external and internal (Nebel, Dimopoulos, and Koehler 1997). The external point of view refers to the definition of the planning task, which may contain information as type information, initial facts, objects, and/or operators that are not needed for a solution. The internal point of view refers to the internal process of planning where ground operators or ground facts can be relevant or irrelevant. Also, different degrees of irrelevance can be distinguished, from completely irrelevant information (information that is not part of any solution), to solution irrelevance (information that may appear in solutions but it is not necessary). Deciding about these "semantic" notions of relevance was showed to be computationally as hard as planning itself (Nebel, Dimopoulos, and Koehler 1997), and guaranteeing a plan does not contain redundant actions is NP-complete (Fink and Yang 1992; Nakhost and Müller 2010), where redundant actions are those that can be removed from a plan while maintaining its validity.

In large-scale real-world applications with varying and diverse goals it is common to include in the domain many operators, static facts and objects which might be irrelevant for a specific goal and initial situation. This can occur for instance in robotic domains where the robot can perform multiple tasks. It is true that current planners which are guided by powerful heuristics will probably not include (many) irrelevant operators in the output plan. However, there are settings where this is not enough, so that the problem is still interesting. An example is top-k planning (Yen 1971; Katz et al. 2018; Katz, Sohrabi, and Udrea 2020; Speck, Mattmüller, and Nebel 2020), where the objective is to derive a set of plans instead of just one. Iterative approaches to top-k planning (Katz et al. 2018) include more irrelevant information in the plans as more iterations are carried out, to the point that truly alternative plans are not distinguished from those that were extended with unnecessary operators.

In this work, we focus on filtering unnecessary actions in plan post-optimization, in the same line of some previous works (Nakhost and Müller 2010; Chrpa, McCluskey, and Osborne 2012b,a; Balyo, Chrpa, and Kilani 2014). Specifically, we contribute with several compilations to encode the problem as an Automated Planning Task, and then solve it using an off-the-shelf automated planner. Using an optimal planner will guarantee to find the "best" set of unnecessary actions, while a satisficing planner is expected to find solutions faster but without that guarantees.

The rest of the paper is organized as follows. Next section includes a summary of related work. Section introduces basic notions of classical planning and plan optimization. In Section we define the planning compilations. Section contains an empirical evaluation comparing our approach to existing post planning optimization methods. Finally there is a discussion and an outline future lines of research.

Related Work

Most previous works related to action and fact irrelevance focused on filtering irrelevant actions in a plan post-optimization step. Fink and Yang [1992] formalized different notions of plan justifications and provided complexity results for them. Specifically, they defined greedily justified actions as those that make the plan invalid when they are removed from it, and perfectly-justified plans as those with no redundant actions. Nakhost and Müller [2010] proposed Action Elimination, an algorithm based on greedy justification, and an additional technique based on plan neighborhood graph search. Siddiqui and Haslum [2015] proposed a method to improve a plan based on neighbourhood search, where the substitution of subplans with improved subplans define a plans' neighbourhood. The solutions found by this approach are not necessarily subsets of the input plan. Chrpa et al. proposed methods based on identifying redundant actions and non-optimal sub-plans by analyzing action dependencies and independencies (Chrpa, McCluskey, and Osborne 2012b), and by checking pairs of inverse actions (Chrpa, McCluskey, and Osborne 2012a). Balyo, Chrpa, and Kilani [2014] introduced an approach for determining the "best" set of redundant actions encoding the problem as a weighted MaxSAT problem. Similarly, Muise, Beck, and McIlraith [2016] propose an encoding as a weighted MaxSAT problem centered on improving the flexibility of partial-order plans, that also supports redundant action elimination. Say, Cire, and Beck [2016] also propose an approach to improve the flexibility and eliminate actions from a partial-order plan, but based on mixed-integer lineal programming models. Olz and Bercher [2019] provide a complexity analysis for eliminating redundant actions for partial-order plans, finding that this problem is harder than the one regarding totally ordered plans. Waters, Padgham, and Sardina [2021] optimize a partial-order plan flexibility via action reinstantiation using a MaxSAT approach. The work in this paper is closely related with all these works with the difference that we approach the problem using AP.

There are also techniques that remove irrelevant information at preprocessing. For instance, Nebel, Dimopoulos, and Koehler 1997 proposed heuristics for selecting relevant information based on minimizing the number of initial facts by computing a fact generation tree going backwards from the goals; and a recent approach (Silver et al. 2020) learns convolutional graph neural networks to predict subsets of objects that are sufficient for solving the planning task. Approaching the problem at preprocessing has the additional advantage that it can make easier the planning process. This is specially interesting when the number of objects is very large. In this case, most modern heuristic planners that ground the actions over objects during preprocessing scale poorly. This is also one of the motivations for recent research on lifted planning (Corrêa et al. 2020), abstractions that simplify the problem (Fuentetaja and de la Rosa 2016) and some approaches based on generalized planning, as the aforementioned work of Silver et al.

Irrelevant information may have an important negative impact when the objective is to derive a set of plans instead of just one, though it has been little studied in this context. However, it is an important setting since there are many applications where deriving a set of plans is necessary, as malware detection (Boddy et al. 2005), model-based goal and plan recognition (Ramírez and Geffner 2009, 2010; Sohrabi, Riabov, and Udrea 2016; Pereira 2016), diverse planning (Srivastava et al. 2007; Roberts, Howe, and Ray 2014), plan monitoring (Fox et al. 2006) and explainable AI (Chakraborti et al. 2018; Eifler et al. 2020). When it is required to derive a set of plans, those plans could be asked to fulfill some extra constraints as being the best *k* solutions or being diverse. In this sense several approaches have been defined for generating sets of plans (Yen 1971; Katz et al. 2018; Katz, Sohrabi, and Udrea 2020; Speck, Mattmüller, and Nebel 2020), and for considering different diversity metrics (Fox et al. 2006; Nguyen et al. 2012; Roberts, Howe, and Ray 2014).

Background

This section contains some basic definitions on AP and plan justifications that will be used throughout the rest of the paper.

Classical Planning

In this subsection the STRIPS formalism (Fikes and Nilsson 1971) for Classical Automated Planning is introduced.

A classical planning task is defined as a tuple $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, where $\mathcal{F}$ is set of propositions; $\mathcal{A}$ is set of actions; $\mathcal{I} \subseteq \mathcal{F}$ is the initial situation, encoding what propositions are true initially; and $\mathcal{G} \subseteq \mathcal{F}$ is a set of goal propositions. Every $a \in \mathcal{A}$ has preconditions, denoted as $pre(a) \subseteq \mathcal{F}$, added effects $add(a) \subseteq \mathcal{F}$ and negative effects $del(a) \subseteq \mathcal{F}$.

A planning task Π defines a state model which states $s \in S$ are subsets of $\mathcal{F}$ and are represented by the fluents that are true in the corresponding state. In this model, the initial state is $s_i = \mathcal{I}$, and the goal states are those s_g that include the goals $\mathcal{G} \subseteq s_g$. The actions $a \in \mathcal{A}$ that are applicable in a state s, denoted as $A(s)$, are those for which $pre(a) \subseteq s$. The transition function is γ, where $\gamma(s, a) = (s \setminus del(a)) \cup add(a)$ represents the state s' that results from the application of the action a in state s.

A solution or valid 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 $s_0 = \mathcal{I}$ and, for each i such that $1 \leq i \leq n$, a_i is applicable in s_{i-1} and $s_i = \gamma(s_{i-1}, a_i)$. A plan π solves Π if and only if $\mathcal{G} \subseteq s_n$. Each action $a \in \mathcal{A}$ is assumed to have a non-negative cost $c(a)$, so that the cost of a plan is $c(\pi) = \sum c(a_i)$. A plan is optimal if it has minimum cost.

Plan Justification

This subsection introduces the notion of justified plans. Causal links are used in some types of plan justifications, so they are introduced now. Given a planning task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ and a plan $\pi = (a_1, ..., a_n)$, a causal link between two actions a_i, a_j $(i < j)$ in π represents a causal dependency between both actions through a proposition p. Specifically, the triple $cl = \langle a_i, a_j, p \rangle$ forms a causal link if

a_i adds p, p is a precondition of a_j, and p is neither added nor deleted by any action between a_i and a_j. Given the planning task Π and a plan π, the causal links can be extracted following the approach presented in (Celorrio et al. 2013) in polynomial time. Causal links were called *establishments* in previous work (Fink and Yang 1992).

The notion of plan justification can be traced back to the early 1990s (Fink and Yang 1992). In that work, Fink and Yang define three types of plan justifications: **backward** justification, **well** justification and **perfect** justification. Given $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, a plan $\pi = (a_1, ..., a_n)$ and the set of causal links of π_{links}, an action $a_i \in \pi$ is backward justified if $\exists p \in add(a_i)$ such that $p \in \mathcal{G}$ or $\langle a_i, a_j, p\rangle \in \pi_{links}$ and a_j is backward justified. This means that a_i is backward justified if there is a sequence of causal links rooting in a_i that ends up related to a goal proposition. A plan π is backward justified if all of its actions are backward justified. Well justified actions are those that cannot be removed from the plan without affecting the applicability of other actions. Perfectly-justified plans are those for which no subset of actions can be removed from the plan without invalidating plan. We also apply this idea.

AP compilations for Action Elimination

This section introduces some formal definitions used in the rest of the paper, and explains the proposed AP compilations to eliminate unnecessary actions from plans.

Formal Definitions

We use the notion of *perfectly-justified* introduced by Fink and Yang. Thus, a plan is perfectly-justified if no actions can be *skipped* or eliminated while maintaining the plans' validity. We introduce it using the following definitions.

Definition 1 (Reduced Plan). *Given a plan $\pi = \langle a_1, \ldots, a_n\rangle$ for Π and a subset of its actions $A_\pi \subseteq \pi$, the reduced plan $\pi_{\setminus A_\pi}$ is the action sequence resulting from eliminating the actions $a_i \in A_\pi$ from π.*

Definition 2 (Well-justified action set). *A subset of plan actions $A_\pi \subseteq \pi$, $A_\pi \neq \emptyset$, is well justified if the corresponding reduced plan $\pi_{\setminus A_\pi}$ is not a valid plan for Π.*

The previous definition just extends the notion of well-justified actions to well-justified subsets of actions. [1]

Now, we consider a plan π to be perfectly-justified if all of its subsets of actions are well justified, i.e. all the plans reduced by those subsets are invalid, so that there is no way of reducing the plan while maintaining its validity.

Definition 3 (Perfectly-justified plan). *A plan π is perfectly-justified iff all non-empty subsets of its actions, $A_\pi \subseteq \pi$, are well-justified.*

Then, if there is at least a subset of actions which is not well-justified, the plan is not perfectly-justified. In that case will say the actions in that subset are **unnecessary**.

Given Π and a plan π, the task of finding the smallest perfectly-justified plan by eliminating actions from π is called Minimal Length Reduction (MLR) (Balyo, Chrpa, and Kilani 2014). Balyo, Chrpa, and Kilani also define the Minimal Reduction (MR) task. The aim of this task is to find a plan with the smallest possible cost by eliminating actions from π. We are particularly interested in MR and MLR. In the following section we propose different ways of compiling the problem into an AP one so that it can be solved with an off-the-shelf planner.

Perfect Justification as Planning

In this section we propose two alternative compilations to represent the perfect justification problem as a planning problem.

Compilation with skip actions The idea is to define a classical planning task that, given a planning task and a solution plan, can identify and eliminate sets of unnecessary actions from plans. Then, it is necessary to encode a task where any number of actions can be skipped (eliminated) from the original plan π while maintaining the order of the actions in the sequence (i.e. if $j > i$, a_i cannot be applied after a_j).

Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be planning task and $\pi = \langle a_1, ..., a_n\rangle$ a valid plan for Π. We define a new planning task $\Pi^{skip} = (\mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G})$ that encodes the action elimination problem as follows:

- $\mathcal{F}' = \mathcal{F} \cup \mathcal{F}_{last} \cup \mathcal{F}_{next} \cup \mathcal{F}_{planact}$, where:
  - $\mathcal{F}_{last} = \{last_i \,|\, 0 \leq i \leq n\}$ facts represent the last position considered. There is a position (order in the sequence) for every action in the original plan plus an additional zero position,
  - $\mathcal{F}_{next} = \{next_{i,i+1} \,|\, 0 \leq i < n\}$ are static facts to encode that position i is immediately before position j, and
  - $\mathcal{F}_{planact} = \{planact\_a_i \,|\, 1 \leq i \leq n\}$ are static facts to represent the action a appears in the plan π at position i.
- $\mathcal{A}' = \{a_{i,i+1} \,|\, a \in \mathcal{A}, 0 \leq i < n\} \cup \{skip_j \,|\, 1 \leq j \leq n\}$, where:
  - There is an a_{ij} action for every action a in the original task and consecutive positions i, $j = i + 1$, which is defined as follows:

$$\begin{aligned} pre(a_{ij}) &= pre(a) \cup \\ &\quad \{last_i, next_{i,j}, planact\_a_j\} \\ add(a_{ij}) &= add(a) \cup \{last_j\} \\ del(a_{ij}) &= del(a) \cup \{last_i\} \end{aligned}$$

  - There is a $skip_j$ action for every possible plan position j. These actions allow to skip the action in position j from the original plan. $skip_j$ are defined as follows:

$$\begin{aligned} pre(skip_j) &= \{last_i, next_{i,j}\} \\ add_(skip_j) &= \{last_j\} \\ del(skip_j) &= \{last_i\} \end{aligned}$$

- $\mathcal{I}' = \mathcal{I} \cup \{last_0\} \cup \{next_{i,i+1} | 0 \leq i < n\} \cup \{planact\_a_i \,|\, a_i \in \pi\}$

[1] This notion is also defined by Balyo, Chrpa, and Kilani [2014] as plan reductions.

The $skip_j$ actions have zero cost. The cost of the other actions depends on the type of action elimination to be performed. This will be addressed later in the paper.

Actions a_{ij} in $\mathcal{A}'$ will only be applicable if there is an occurrence of action $a \in \mathcal{A}$ in the original plan π at position j. For that, facts of type $planact\_a_i$ are included in $\mathcal{I}'$, representing the plan π. There is one of such facts per plan action, indicating π contains an occurrence action $a \in \mathcal{A}$ at position i. Thus, when Π^{skip} is solved, the resulting plan will only contain the actions in the original plan that have not been skipped without altering the order.

Solutions plans for Π^{skip} are not directly plans for Π, due to the possible occurrence of skip actions and additional action parameters related to the positions i and j in actions a_{ij}. However, it is quite straightforward to transform them into valid plans for Π, just by removing those skip actions from the plan and those parameters from the remaining actions. The resulting plan of eliminating skip actions and extra parameters from will be denoted as π'.

Definition 4. *Let π' be a valid plan of Π^{skip}. Then, the compiled back plan for Π is $\pi'' = \{a \,|\, a \in \pi \wedge a_{ij} \in \pi'\}$.*

This means that every a_{ij} action is replaced by its original action a and skip actions are not included.

Proposition 1. *The plan π'' obtained from any valid plan π' for Π^{skip} is a valid plan for Π.*

Proof sketch. A plan π that induces the state sequence $\mathcal{S}_\pi = \langle s_0, \ldots, s_n \rangle$ is valid if all of its actions are applicable in the state they are applied and $\mathcal{G} \subseteq s_n$. Since the goals of both Π and Π^{skip} are the same, if π' is valid for Π^{skip}, then it will also achieve all the goals in Π by definition. Since $pre(a) \subset pre(a_{i,j})$ for every $a_{i,j} \in \mathcal{A}'$, if $a_{i,j}$ is applicable in a state, then the corresponding action $a \in \mathcal{A}$ is also applicable in that state. We know that π' is valid, so, starting from $\mathcal{I}$ all of its actions are applicable. Since $\mathcal{I} \subset \mathcal{I}'$, all actions in π'' are in turn applicable starting from $\mathcal{I}$. The goals are met and all actions are applicable, so π'' is a valid plan for Π.

Enhanced compilation with skip actions Identifying if all actions in a plan are necessary is NP-hard. This does not mean that *some* actions cannot be easily identified as necessary (their elimination would render the plan invalid). Actions that have a goal proposition in their *add* list are paramount for the plans' success. In particular, if a goal proposition is achieved by a single action in a plan, we can assure that the removal of that action would generate an invalid plan (goals not achieved). Even more, if some preconditions of that action are also achieved by single actions, then those actions are also clearly necessary. With this idea, we define *trivially necessary actions*.

Definition 5 (Trivially necessary actions). *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task and $\pi = (a_1, ..., a_n)$ a valid a plan. An action $a_i \in \pi$ is trivially necessary if: (a) $\exists g \in \mathcal{G}$ such that $g \in add(a_i)$ and $\nexists a_j \in \pi, j \neq i$ such that $g \in add(a_j)$; or (b) there is a necessary action $a_j, i < j$, such that $\exists p \in pre(a_j) \cap add(a_i)$ and $\nexists a_k \in \pi, k < j, k \neq i$, such that $p \in add(a_k)$.*

Trivially necessary actions cannot be skipped from the plan without affecting its' validity. We denote the remaining actions which can potentially be skipped as $A_s = \{i \,|\, a_i \textit{ is not trivially necessary}\}$. The proposed AP compilation (Π^{skip}) is enhanced to consider this information. For that, we introduce the additional (static) facts $\mathcal{F}_{skippable} = \{skippable_i \,|\, i \in A_s\}$ into $\mathcal{F}'$. Then, the *skip* actions and the initial state are modified as follows:

- Skip actions have an additional precondition to be applicable only in the case that the action is *skippable*:

$$\begin{aligned} pre(skip_j) &= \{last_i, next_{i,j}, skippable_j\} \\ add_(skip_j) &= \{last_j\} \\ del(skip_j) &= \{last_i\} \end{aligned}$$

- The initial state contains all propositions that identify *skippable* actions: $\mathcal{I}' = \mathcal{I} \cup \{last_0\} \cup \{next_{i,i+1} \,|\, 0 \leq i < n\} \cup \{planact\_a_i \,|\, a_i \in \pi\} \cup \{skippable_i | i \in A_s\}$

With the compilation enhanced with skip actions the number of applicable skip actions can be reduced, which should speed up the planning process.

Compilation without skip actions There is another compilation which does not require the use of skip actions. It consist of encoding the planning task in a way that allows to include any action occurring in the original plan after the last action that was previously included. We achieve this by creating an order relation between the actions in π. More formally, given the planning task Π and a solution plan π we define $\Pi^{order} = (\mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G})$ as follows:

- $\mathcal{F}' = \mathcal{F} \cup \mathcal{F}_{last} \cup \mathcal{F}_{order} \cup \mathcal{F}_{planact}$, where $\mathcal{F}_{last}$ and $\mathcal{F}_{planact}$ have the same definition as before and $\mathcal{F}_{order} = \{order_{i,j} \,|\, 0 \leq i \leq n, 1 \leq j \leq n\}$ are static facts to encode that position i is before position j.
- $\mathcal{A}' = \{a_{i,j} \,|\, a \in \mathcal{A}, 0 \leq i \leq n, 1 \leq j \leq n\}$, where there is an $a_{i,j}$ action for every action a in the original task and combination of positions i, j, defined as follows:

$$\begin{aligned} pre(a_{ij}) &= pre(a) \cup \\ &\quad \{last_i, order_{i,j}, planact\_a_j\} \\ add(a_{ij}) &= add(a) \cup \{last_j\} \\ del(a_{ij}) &= del(a) \cup \{last_i\} \end{aligned}$$

- $\mathcal{I}' = \mathcal{I} \cup \{last_0\} \cup \{order_{i,j} \,|\, 0 \leq i, j < n, i < j\} \cup \{planact\_a_i \,|\, a_i \in \pi\}$.

There are only a few differences between this and the previous compilation. One of them is that $next_{i,j}$ propositions have been replaced by $order_{i,j}$ propositions. $next$ represented pairs of consecutive positions, while $order$ represents pairs of positions such that one is before the other, but they are not necessarily consecutive. Another difference is that now $last$ refers always to a position that correspond to the last included action. Thus, the preconditions of $a_{i,j}$ actions check that the action being included at position j, occurs in the original plan before the last included action with position i. Finally, $\mathcal{I}'$ sets the relation order between the actions in π, where there is a fact $order_{0,j}$ for every position j in the plan, which allows the application of any action of the plan.

Solution plans π' for Π^{order} are compiled back as in Definition 4. This transformation also yields valid plans for the original task.

Proposition 2. *The plan π'' obtained from any valid plan π' for Π^{order} is a valid plan for Π.*

The proof is analogous to the proof of Proposition 2, but without considering skip actions. It is omitted to avoid unnecessary repetition.

Theoretical Properties

This section shows the type of plans that are obtained from solving the compiled planning tasks. In particular, we show that the set of valid plans for the compilations is the set of valid reduced plans of the original task.

Let P_Π be the set of valid plans for a planning task Π. Two planning tasks Π, Π' are *equivalent* if they have the same sets valid plans, $P_\Pi = P_{\Pi'}$. For two plans π^-, π we say that π^- is a subset of π, $\pi^- \subset \pi$ if π^- can be generated from eliminating actions from π. Since both compilations **only** allow for the execution of actions in the original plan or for their (implicit or explicit) elimination, the set of valid plans for both Π^{skip}, Π^{order} contain exactly all the valid plans that can be generated from eliminating subsets of actions from the original task if their corresponding transformations as defined in 4 are considered. More formally:

Proposition 3. *Let Π be a planning task and $\pi = \langle a_1, ..., a_n \rangle$ a valid plan for Π. Let P_{skip} be the set of compiled-back valid plans for Π^{skip}. Then $P_{skip} = \{\pi^- \subseteq \pi \,|\, \pi^- \in P_\Pi\}$.*

Proof sketch. We have to show that (i) any plan in P_{skip} is in $\{\pi^- \subseteq \pi \,|\, \pi^- \in P_\Pi\}$, and that (ii) any plan in $\{\pi^- \subseteq \pi \,|\, \pi^- \in P_\Pi\}$ is in P_{skip}:
(i) Let $\pi' = \langle b_1, ..., b_m \rangle \in P_{skip}$ be any compiled-back valid plan for Π^{skip}. Since all a_{ij} belonging to Π^{skip} have in their precondition $planact\_a_j$, only actions in π can be in π'. It is trivial to show that only one $last_i$ proposition is true in each state. In $\mathcal{I}$ only $last_0$ is true. Because of Π^{skip} encoding, only a_1 or $skip$ are applicable initially, so the first action in the plan can be either applied or eliminated. Both of them delete $last_0$ and add $last_1$. Continuing this process until $last_n$ is true it is easy to see that π' can only have actions in π respecting their order, where some of them might be skipped. By Proposition 2, $\pi' \in P_\Pi$. Therefore, it is proven that $(\forall\, \pi', \pi' \in P_{skip} \implies \pi' \subseteq \pi \wedge \pi' \in P_\Pi)$.
(ii) Let $\pi = \langle b_1, ..., b_m \rangle \in \{\pi^- \subset \pi \,|\, \pi^- \in P_\Pi\}$. Following similar reasoning we can prove that $(\forall \pi, \pi \in \{\pi^- \subseteq \pi \,|\, \pi^- \in P_\Pi\} \implies \pi \in P_{skip})$. This is omitted for space reasons. Therefore $P_{skip} = \{\pi^- \subseteq \pi \,|\, \pi^- \in P_\Pi\}$. □

Proposition 4. *Let Π be a planning task and $\pi = \langle a_1, ..., a_n \rangle$ a valid plan for Π. Let P_{order} be the set of compiled-back valid plans for Π^{order}. Then $P_{order} = \{\pi^- \subseteq \pi \,|\, \pi^- \in P_\Pi\}$.*

The proof is similar to the one of Proposition 3.

When comparing the two compilations, the clear difference is the $skip$ actions. In the first compilation, Π^{skip}, to eliminate m consecutive actions, m consecutive *skip* actions must be applied. In cases where a plan has large sets of unnecessary actions, this might affect the performance of the planning engine. By contrast, in the second compilation, Π^{order}, the elimination of actions occurs without the need of any virtual action, and any number of actions can be eliminated with a single action. On the other hand, the first compilation has a branching factor of exactly 2 in each step: either the current action or the skip action can be applied. On the second compilation, the branching factor can be as high as $|\pi| = n$, and $(\sum_{i=1}^{n-1} i)$ *before* propositions must be created. When original plans are particularly long, this might become an issue.

Minimal and Minimal Length Reductions

We want to solve MR and MLR (Balyo, Chrpa, and Kilani 2014) using the planning compilations presented in this work. We have already shown in Proposition 3 that the compilations are equivalent, and that their set of valid plans is the set of valid plans that can be obtained from eliminating actions from the original plan.

To solve MR with Π^{skip} and Π^{order} all that should be done is to set the cost of all the actions in the compilations to their corresponding cost in the original task. In Π^{skip} the cost of $skip$ actions is set to 0. To solve MLR, all actions costs should be 1, and the cost of $skip$ actions are also 0.

With these costs defined, it becomes clear that using an optimal planner guarantees finding a least-cost plan in the case of MR, and a plan of minimum length in the case of MLR.

Evaluation

In this section we present an empirical comparison and analysis of different techniques used to remove unnecessary actions from plans. We generate plans from instances of the satisficing track of the International Planning Competition (IPC) using the Fast-Downward (FD) planning system (Helmert 2006), which is a state-of-the-art planner. We compared the results of using the planning compilations to solve both MR and MLR with SAT based approaches and the greedy approach proposed by Balyo, Chrpa, and Kilani [2014].[2]

Evaluation Setting

To test the algorithms, we need as input a set of planning tasks and plans that solve them. To generate them, we configured FD to find one plan as fast possible on several tasks from the IPC. In this stage, we set a time limit of 10 minutes to find plans.

We tested the following methods for eliminating unnecessary actions:

- Our implementation of the Π^{skip} compilation. This implementation takes as input a planning task (in PDDL format) and a plan and generates the Π^{skip} task (also in a PDDL format). FD was configured to find an optimal solution for this task using A* as a search algorithm and the h_{max} heuristic (Bonet and Geffner 2001). We set a time limit of 10 minutes and a memory limit of 8GB for each instance. With these configurations, we solve

[2]Our code will be made publicly available upon publication.

Domain	Plans			Π^{order}				ALL			Π^{skip}	$\Pi^{skipE-BLIND}$	Π^{skipE}	SAT-MR
	#	Cost	Avg	#	diff	Ratio	Time(s)	#	diff	Ratio	Time(s)	Time(s)	Time(s)	Time(s)
elevator	19	25934	1365	19	1404	6.28%	11.2(5.4)±13.7	19	1404	6.28%	**0.2(0.0)± 0.1**	**0.2(0.0)± 0.1**	**0.2(0.0)± 0.1**	0.3±0.1
barman	20	7721	386	20	903	10.86%	11.3(6.4)±9.6	20	903	10.86%	0.3(0.1)±0.1	0.3(0.1)±0.1	**0.2(0.0)± 0.0**	0.4±0.1
floor-tile	10	1347	135	10	113	8.63%	0.4(0.0)±0.2	10	113	8.63%	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1± 0.0**
no-mystery	20	715	36	20	0	0.00%	0.6(0.0)±0.4	20	0	0.00%	0.5(0.0)±0.3	0.5(0.0)±0.3	0.6(0.0)±0.4	**0.1± 0.0**
parking	20	1537	77	20	23	1.50%	0.6(0.0)±0.1	20	23	1.50%	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	0.3±0.0
peg-solitaire	20	291	15	20	0	0.00%	0.1(0.0)±0.0	20	0	0.00%	0.1(0.0)±0.0	0.1(0.0)±0.0	0.1(0.0)±0.0	0.1±0.0
scanalyzer-3d	20	1837	92	20	60	3.35%	0.3(0.0)±0.2	20	60	3.35%	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	0.3±0.2
sokoban	17	1079	63	17	24	1.68%	10.0(2.1)±9.5	17	24	1.68%	**0.2(0.0)± 0.1**	2.3(2.1)±3.9	**0.2(0.0)± 0.1**	0.3±0.2
visit-all	19	27036	1423	13	120	0.92%	251.4(49.5)±385.6	19	216	0.87%	4.0(3.3)±7.7	**1.9(1.2)± 4.0**	2.5(1.9)±4.3	8.2±9.2
woodworking	20	35100	1755	20	2120	4.95%	9.3(8.1)±30.1	20	2120	4.95%	0.2(0.0)± 0.1	4.5(4.3)±18.8	**0.2(0.0)± 0.0**	0.3±0.1

Table 1: Results of different methods for the minimal reduction (MR) problem

Domain	Plans			Π^{order}				ALL			Π^{skip}	$\Pi^{skipE-BLIND}$	Π^{skipE}	SAT-MLR
	#	Cost	Avg	#	diff	Ratio	Time(s)	#	diff	Ratio	Time(s)	Time(s)	Time(s)	Time(s)
elevator	19	25934	1365	19	97	2.2%	11.4(5.4)±13.0	19	97	2.2%	**0.2(0.0)± 0.1**	**0.2(0.0)± 0.1**	**0.2(0.0)± 0.1**	0.3±0.1
barman	20	7721	386	20	606	15.3%	11.8(6.6)±8.9	20	606	15.3%	0.3(0.1)±0.1	0.3(0.1)±0.1	**0.2(0.0)± 0.0**	0.4±0.1
floor-tile	10	1347	135	10	41	7.4%	0.4(0.0)±0.2	10	41	7.4%	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1± 0.0**
no-mystery	20	715	36	20	0	0.0%	0.6(0.0)±0.4	20	0	0.0%	0.6(0.0)±0.3	0.5(0.0)±0.3	0.6(0.0)±0.4	**0.1± 0.0**
parking	20	1537	77	20	23	1.5%	0.6(0.0)±0.1	20	23	1.5%	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	0.3±0.0
peg-solitaire	20	291	15	20	0	0.0%	0.1(0.0)±0.0	20	0	0.0%	0.1(0.0)±0.0	0.1(0.0)±0.0	0.1(0.0)±0.0	0.1±0.0
scanalyzer-3d	20	1837	92	20	24	3.2%	0.3(0.0)±0.2	20	24	3.2%	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	**0.1(0.0)± 0.0**	0.2±0.2
sokoban	17	1079	63	17	116	2.0%	10.4(2.0)±9.6	17	116	2.0%	**0.2(0.0)± 0.1**	2.6(2.4)±4.4	**0.2(0.0)± 0.1**	0.4±0.2
visit-all	19	27036	1423	11	88	1.0%	110.9(59.2)±168.1	19	216	0.9%	4.2(3.5)±8.1	**1.5(0.9)± 2.8**	2.4(1.8)±4.2	7.7±8.3
woodworking	20	35100	1755	20	100	5.2%	6.7(5.6)±19.5	20	100	5.2%	0.2(0.0)±0.1	4.7(4.5)±19.3	**0.2(0.0)± 0.0**	0.3±0.1

Table 2: Results of different methods for the minimal length reduction (MLR) problem.

the minimal reduction and the minimal length reduction problems.

- Our implementation of the Π^{order} compilation, following the same procedure.
- Our implementation of the Π^{skipE} compilation. For this compilation we also report results obtained running blind search.
- The SAT compilations of MR and MLR implemented by Balyo, Chrpa, and Kilani [2014]. This compilation is written in Java, and the CNF formula can be solved using any MaxSAT solver. We used the Sat4J (Le Berre and Parrain 2010) MaxSAT solver. We are aware that there are faster MaxSAT solvers, but we used SAT4J for its availability and convenience, since it is implemented in the same language as the SAT compilations. Our intention is to check to what extent the different approaches are comparable.
- Finally, we compare sub-optimal results obtained from solving Π^{skip} with a satisficing configuration of FD and compare them with sub-optimal results obtained from the Greedy Action Elimination (GAE) algorithm (Balyo, Chrpa, and Kilani 2014). [3]

All the experiments were executed on a computer with 16GB of ram and an 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz processor.

Results

Table 1 shows the results to the MR (Minimal Reduction) problem. First three columns in this table represent the number of input plans (#), their cost and average cost. For Π^{order} we show the number of solved instances (10m time limit), the sum of the cost reductions obtained (diff), the average ratio of reduction and the average/stdev time to solve. For Π^{skip}, Π^{skipE} (enhanced Π^{skip}) and SAT-MR we group the cost reduction and ratios in a single column because they get the same results (except for the time needed to solve). Π^{order} results are shown separately because it is the only method that did not manage to solve all instances in the time limit. In the Time(s) column we show the average time to get a solution, and in parentheses, for AP-based approaches, the actual time the planner spent doing search (the rest of the time is for preprocessing). As it can be seen, the preprocessing time of Π^{order} proved to be too costly, taking more time than the actual search in many cases. This is due to the instantiation, and might be alleviated if the compilation generates instantiated actions instead of PDDL files.

In general, Π^{order} performed worse than the other approaches, even failing to solve instances of the *visit-all* domain. Plans in this domain can have thousands of actions, and the number of *before* propositions has a quadratic growth w.r.t the plan length. Π^{skip} and SAT-MR manage to solve most instances in tenths of a second, except for the visit-all domain where Π^{skip} and its' enhancement show better results. It is interesting to note that Π^{skipE} shows competitive results even when the planner performs blind search. This confirms the intuition that the introduction of the *skippable* predicates greatly reduces the size of the search space.

MLR (Minimal Length Reduction) results are on Table 2. Here, the *diff* column is the total length reduction instead of the cost reduction. Again, Π^{order} has the worst results in general, with a similar behaviour to the MR results. Π^{skip}, Π^{skipE} and SAT-MR show similar results to the ones for MR.

[3] We use directly the author's implementation, provided by personal communication.

Domain	Plans			GAE				Satisficing Π^{skipE}			
	#	Cost	Avg	#	diff	Ratio	Time(s)	#	diff	Ratio	Time(s)
elevator	19	25934	1365	19	1404	6.28%	0.2±0.1	19	1404	6.28%	0.2(0.0)±0.1
barman	20	7721	386	20	703	8.65%	0.4±0.1	20	**903**	**10.86%**	**0.2(0.0)± 0.0**
floor-tile	10	1347	135	10	**113**	**8.63%**	0.1±0.0	10	95	7.28%	0.1(0.0)±0.0
no-mystery	20	715	36	20	0	0.00%	**0.1± 0.0**	20	0	0.00%	0.5(0.0)±0.3
parking	20	1537	77	20	23	1.50%	0.3±0.1	20	23	1.50%	**0.1(0.0)± 0.0**
peg-solitaire	20	291	15	20	0	0.00%	**0.0± 0.0**	20	0	0.00%	0.1(0.0)±0.0
scanalyzer-3d	20	1837	92	20	60	3.35%	0.2±0.2	20	60	3.35%	**0.1(0.0)± 0.0**
sokoban	17	1079	63	17	12	0.81%	0.2±0.1	17	**24**	**1.68%**	0.2(0.0)±0.1
visit-all	19	27036	1423	19	**216**	**0.87%**	7.7±8.5	19	214	0.86%	**0.7(0.1)**±0.6
woodworking	20	35100	1755	20	2050	4.77%	0.3±0.1	20	**2100**	**4.89%**	**0.2(0.0)**±0.0

Table 3: Results for two sub-optimal action elimination methods.

Table 3 shows results for GAE and Π^{skipE} with a satisficing configuration (weighted A* with w=3 and h_{max}). Time results are low and similar in both cases. The reduction ratio is similar for both approaches, coming close to the optimal solution in many instances. For really large plans, the AP approach with a satisficing planner finds good solutions really fast. For the visit-all domain, while GAE takes on average 7 seconds to solve an instance, Π^{skipE} takes less than a second.

Conclusions and Future Work

In this work we proposed two Automated Planning approaches to find an eliminate unnecessary actions from plans. In particular, given a plan we show how to create planning tasks to solve the MR problem and the MLR problem. The experiments done show that the results of solving MR and MLR with these compilations are comparable in time to other state-of-the algorithms to eliminate unnecessary actions. When dealing with particularly large plans, the proposed approaches outperform their SAT-based counterparts under the specified configurations (the usage of different SAT solvers or planners might affect this result).

In future work, we plan to experiment with a wider range of domains and instances. In particular, we wish to identify situations where the proposed AP approaches to action elimination might outperform SAT-based approaches and viceversa. We also want to incorporate unnecessary action elimination on settings where it might be important, such as top-k planning and plan recognition.

Acknowledgments

This work has been partially funded by FEDER/Ministerio de Ciencia, Innovación y Universidades - Agencia Estatal de Investigación/TIN2017-88476-C2-2-R, RTC-2016-5407-4, and the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17), and in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

References

Balyo, T.; Chrpa, L.; and Kilani, A. 2014. On different strategies for eliminating redundant actions from plans. In *Seventh Annual Symposium on Combinatorial Search*.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of Action Generation for Cyber Security Using Classical Planning. In *ICAPS 2005*, 12–21.

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

Celorrio, S. J.; Haslum, P.; Thiebaux, S.; et al. 2013. Pruning bad quality causal links in sequential satisfying planning.

Chakraborti, T.; Fadnis, K. P.; Talamadupula, K.; Dholakia, M.; Srivastava, B.; Kephart, J. O.; and Bellamy, R. K. E. 2018. Visualizations for an Explainable Planning Agent. In *IJCAI 2018*, 5820–5822.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012a. Determining redundant actions in sequential plans. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, volume 1, 484–491. IEEE.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012b. Optimizing plans through analysis of action dependencies and independencies. In *Twenty-Second International Conference on Automated Planning and Scheduling*.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2020. Lifted successor generation using query optimization techniques. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 80–89.

Eifler, R.; Steinmetz, M.; Torralba, Á.; and Hoffmann, J. 2020. Plan-Space Explanation via Plan-Property Dependencies: Faster Algorithms & More Powerful Properties. In *IJCAI 2020*, 4091–4097.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Fink, E.; and Yang, Q. 1992. Formalizing plan justifications.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan Stability: Replanning versus Plan Repair. In *ICAPS 2006 - Proceedings, Sixteenth International Conference on Automated Planning and Scheduling*, volume 2006, 212–221.

Fuentetaja, R.; and de la Rosa, T. 2016. Compiling irrelevant objects to counters. special case of creation planning. *AI Communications*, 29(3): 435–467.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-Quality Planning: Finding Practically Useful Sets of Best Plans. In *AAAI 2020*, 9900–9907.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A Novel Iterative Approach to Top-k Planning. In *ICAPS 2018*, 132–140.

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.

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

Nakhost, H.; and Müller, M. 2010. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 121–128. AAAI.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *European Conference on Planning*, 338–350. Springer.

Nguyen, T. A.; Do, M. B.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating Diverse Plans to Handle Unknown and Partially Known User Preferences. *Artif. Intell.*, 190: 1–31.

Olz, C.; and Bercher, P. 2019. Eliminating redundant actions in partially ordered plans—a complexity analysis. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 310–319.

Pereira, R. F. 2016. Landmark-Based Approaches for Plan Recognition Tasks. *M.Sc. Dissertation - Pontificia Universidade Católica do Rio Grande do Sul (PUCRS)*.

Ramírez, M.; and Geffner, H. 2009. Plan recognition as planning. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Ramírez, M.; and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Roberts, M.; Howe, A. E.; and Ray, I. 2014. Evaluating diversity in classical planning. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Say, B.; Cire, A. A.; and Beck, J. C. 2016. Mathematical programming models for optimizing partial-order plan flexibility. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 1044–1052.

Siddiqui, F. H.; and Haslum, P. 2015. Continuing plan quality optimisation. *Journal of Artificial Intelligence Research*, 54: 369–435.

Silver, T.; Chitnis, R.; Curtis, A.; Tenenbaum, J.; Lozano-Perez, T.; and Kaelbling, L. P. 2020. Planning with learned object importance in large problem instances using graph neural networks. *arXiv preprint arXiv:2009.05613*.

Sohrabi, S.; Riabov, A.; and Udrea, O. 2016. Plan Recognition as Planning Revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *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 *IJCAI*, 2016–2022.

Waters, M.; Padgham, L.; and Sardina, S. 2021. Optimising partial-order plans via action reinstantiation. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 4143–4151.

Yen, J. Y. 1971. Finding the k shortest loopless paths in a network. *management Science*, 17(11): 712–716.