On Speeding Up Methods for Identifying Redundant Actions in Plans

Jakub Med, Lukáš Chrpa

Faculty of Electrical Engineering, Czech Technical University in Prague
{medjaku1,chrpaluk}@fel.cvut.cz

Abstract

Satisficing planning aims at generating plans that are not necessarily optimal. Often, minimising plan generation time negatively affects quality of generated plans. Acquiring plans quickly might be of critical importance in decision-making systems that operate nearly in realtime. However, (very) suboptimal plans might be expensive to execute and more prone to failures. Optimising plans after they are generated, in a spare time, can improve their quality. This paper focuses on speeding up the *(Greedy) Action Elimination* methods, which are used for identifying and removing *redundant actions* from plans in polynomial time. We present two enhancements of these methods: *Plan Action Landmarks*, actions that are not redundant in a given plan, and *Action Cycles* which are subsequences of actions which if removed do not affect the state trajectory after the last action of the cycle. We evaluate the introduced methods on benchmark problems from the Agile tracks of the International Planning Competition and on plans generated by several state-of-the-art planners, successful in the recent editions of the competition.

Introduction

Classical planning assumes a static, deterministic and fully observable environment, where a solution plan amounts to a sequence of actions transforming the environment from the given initial state to a state satisfying the given goal.

In general, finding a solution plan is PSPACE-complete (Bylander 1994). Bylander (1994) and Helmert (2003) have shown that for some classes of classical planning problems, finding optimal plans is in a higher complexity class than finding satisficing plans for the same problems. Hence many planning engines generate satisficing (suboptimal) plans. Some planners (e.g. LAMA (Richter and Westphal 2010)) incrementally improve solution plans as long as they have time. Other planners (e.g. YAHSP (Vidal 2014)) aim to generate any solution plan as quickly as possible.

Minimising plan generation time, which might be essential in (almost) real-time decision-making systems, often comes with low quality of generated plans. Executing (very) low quality plans might be time and energy consuming and prone to failures. A possible remedy for low quality plans is to optimise them in a post-processing step in a spare time. One class of optimisation techniques aims at identifying redundant actions that can be safely removed from plans. Another class of optimisation techniques aims at (local) optimisation of action subsequences in plans. As demonstrated by the ARAS system (Nakhost and Müller 2010a), it is effective to initially remove redundant actions, which is computationally cheap, before trying more sophisticated and more computationally demanding (local) optimisation techniques. BDPO2 (Siddiqui and Haslum 2015) has shown that specialised post-planning plan optimisation techniques are more effective than optimisation provided by anytime satisficing planners (e.g. LAMA (Richter and Westphal 2010)).

This paper focuses on plan optimisation by identifying and removing sets of redundant actions. Specifically, this paper aims at improving performance of the Action Elimination (AE) (Nakhost and Müller 2010a) and the Greedy Action Elimination (GAE) (Balyo, Chrpa, and Kilani 2014) algorithms that look for redundant actions “by trial and error”. In a nutshell, they iterate through the plan trying to remove the current action and its dependants which became inapplicable – if the goal is still achieved without these actions, they are redundant and can be removed from the plan. In this paper, we propose two enhancements of the (G)AE algorithms. Firstly, we identify actions that cannot be part of any set of redundant actions – we call such actions *Plan Action Landmarks*. Hence, if (G)AE tries to remove Plan Action Landmark, we can immediately infer that the goal will not be achieved if such an action is removed and it is then unnecessary to go through the rest of the plan. Secondly, we identify whether redundant action candidates form *Action Cycles*, that is, the state in the state trajectory positioned after the last of the actions will be the same even after these actions are removed. If such a situation occurs, we know that the goal will be achieved after removing the actions forming an Action Cycle without having to go through the rest of the plan. Both enhancements can be embedded into (G)AE, aiming at improving its performance while maintaining the same “power” (i.e., identify the same sets of redundant actions as the original (G)AE).

We evaluate the (G)AE enhancements on benchmarks used in the Agile Tracks of the International Planning Competition (IPC) 2014 and 2018. For a plan generation, we used five state-of-the-art planners, successful in the recent IPCs.

Preliminaries

In the upcoming paragraphs, we formally define *classical planning* in the *SAS*$^+$ formalism introduced by Bäckström and Nebel (1995) while also taking conditional effects into consideration.

Let V be a set of ***variables*** where each variable $v \in V$ is associated with its domain $D(v)$. An ***assignment*** of a variable $v \in V$ is a pair (v, val), where $val \in D(v)$ is its value. Hereinafter, an assignment of a variable is also denoted as a ***fact***. A (partial) ***variable assignment*** p over V is a set of assignments of individual variables from V, where $vars(p)$ is a set of all variables in p and $p[v]$ represents the value of v in p. A ***state*** is a complete variable assignment (over V). We say that a (partial) variable assignment q ***holds*** in a (partial) variable assignment p, denoted as $p \models q$, if and only if $vars(q) \subseteq vars(p)$ and for each $v \in vars(q)$ it is the case that $q[v] = p[v]$.

An ***action*** is a triplet $a = (pre(a), \mathit{eff}_{cond}(a), C(a))$, where $pre(a)$ is a partial variable assignment representing a's ***precondition***, $\mathit{eff}_{cond}(a)$ represents a's ***conditional effects*** and $C(a)$ denotes the non-negative ***cost*** of a. In particular, $\mathit{eff}_{cond}(a)$ is a set of pairs (c, e), where e is a partial assignment which stands for an ***effect*** and c is a partial assignment representing e's ***condition***.

We say that an action a is ***applicable*** in a state s if and only if $s \models pre(a)$. Furthermore, we say that an effect $e : (c, e) \in \mathit{eff}_{cond}(a)$ is ***active*** in s if and only if $s \models c$. We assume that each action a is well defined, i.e., for each state s in which a is applicable, it is the case that there are no two active effects for a and s assigning a different value to the same variable. The set of (unconditional) ***effects*** of an action a in state s, denoted as $\mathit{eff}(a, s)$, is a partial variable assignment such that for each variable v, $\mathit{eff}(a, s)[v] = e[v]$ iff e is an active effect for a and s and $v \in vars(e)$, or $\mathit{eff}(a, s)[v]$ is undefined otherwise. The ***result*** of applying a in s, denoted as $\gamma(s, a)$, is a state s' such that for each variable $v \in V$, $s'[v] = \mathit{eff}(a, s)[v]$ if $v \in vars(\mathit{eff}(a, s))$ while $s'[v] = s[v]$ otherwise. If a is not applicable in s, $\gamma(s, a)$ is undefined. The notion of action application can be extended to sequences of actions, i.e., $\gamma(s, \langle a_1, \dots, a_n \rangle) = \gamma(\gamma(s, a_1), \langle a_2, \dots, a_n \rangle)$ and $\gamma(s, \langle\rangle) = s$.

A ***classical planning task*** is a quadruple $T = (V, A, I, G)$, where V is a set of variables, A is a set of actions, I is a complete variable assignment representing the ***initial state*** and G is a partial variable assignment representing the ***goal***. We say that a sequence of actions $\Pi = \langle a_1, \dots, a_n \rangle$ is ***applicable*** in a state s if and only if $\gamma(s, \Pi)$ is defined. A ***plan*** is an applicable sequence of actions Π such that $\gamma(I, \Pi) \models G$. The sequence of states visited during the execution of a plan is called ***state trajectory***. Each state trajectory begins with the initial state I and continues with the n states $s_i = \gamma(I, \langle a_1, \dots, a_i \rangle)$, $i \leq n$. Each plan $\Pi = \langle a_1, \dots, a_n \rangle$ has a ***cost*** $C(\Pi)$ which is defined as $C(\Pi) = \sum_{i=1}^{n} C(a_i)$. We say that a plan Π is ***optimal*** if and only if for each plan Π' (for the same planning task) it holds that $C(\Pi) \leq C(\Pi')$.

Redundant Actions

Informally speaking, if a plan of some planning task contains a set of *redundant* actions, a shorter action sequence obtained by removing these actions from the former plan is also a plan for that planning task. The shorter plan is called *reduction* (Nakhost and Müller 2010b).

Definition 1. *Let T be a planning task and Π be a plan for T. We say that actions $a_{x_1}, \dots, a_{x_k} \in \Pi$ represent a* **set of redundant actions** *in Π if and only if $\Pi' = \Pi \setminus \{a_{x_1}, \dots, a_{x_k}\}$ is a plan for T. We also say that Π' is a* **reduction** *of* Π.

Note that in Definition 1 we abuse the notation for set operations as we actually operate with action indices referring to their positions in plans rather than the actions themselves. Hence, for example, the set subtraction operation will not remove all occurrences of the actions from the plan but only the actions on specific positions in the plan.

Also, we would like to emphasise that a subset of a set of redundant actions is not necessarily a set of redundant actions itself.

Consequently, for each plan, there is a finite amount of reductions. Therefore we can determine the "best" one. A plan Π' for some task T is called a ***minimal reduction*** of Π, also a plan for T, if and only if Π' is a reduction of Π and for each reduction Π'' of Π it holds that $C(\Pi') \leq C(\Pi'')$.

It has been proven that finding a minimal reduction is *NP-complete* (Nakhost and Müller 2010b) and can be found by a technique based on MAXSAT (Balyo, Chrpa, and Kilani 2014). On the other hand, some reductions, albeit not necessarily minimal, can be found in a polynomial time, which is the aim of our paper.

(Greedy) Action Elimination

The idea for the Action Elimination (AE) algorithm has been introduced by Fink and Yang (1992) and later rediscovered and formulated by Nakhost and Müller (2010a). As shown in Algorithm 1, AE is trying to remove actions from the plan one by one (from a_1 to a_n – see the for loop in Lines 3–17). Then, the rest of the plan is checked such that actions that become inapplicable are marked for removal while actions that are still applicable are applied (see the for loop in Lines 8–13). If the resulting state (after processing the rest of the plan) is a goal state, then the actions marked as "to be removed" form a set of redundant actions and can be safely removed from the plan.

For each action in the plan, the algorithm processes at most n following actions, hence the asymptotic time complexity of AE is $O(n^2)$.

The Greedy Action Elimination algorithm (Balyo, Chrpa, and Kilani 2014) extends AE in such a way that by a modification of AE it finds the most expensive set of redundant actions, removes it from the plan and repeats the process until the modification of AE no longer finds a set of redundant actions. The AE algorithm is modified such that Line 15 of Algorithm 1 stores only the currently most expensive set of redundant actions. The modification of AE can be called at most n times and hence GAE runs in $O(n^3)$.

Algorithm 1: Action Elimination

```
Input  : Planning task $T = (V, A, I, G)$, a plan
         $\Pi = \langle a_1, \dots, a_n \rangle$ for $T$
Output: A reduction $\Pi'$
1  $removed \leftarrow \emptyset$;
2  $s \leftarrow I$;
3  for $i \leftarrow 1$ to $n$ do
4      $marks \leftarrow \emptyset$;
5      if $i \notin removed$ then
6          $marks \leftarrow marks \cup \{i\}$;
7          $s' \leftarrow s$;
8          for $j \leftarrow i + 1$ to $n$ do
9              if $j \notin removed$ then
10                 if $a_j$ is not applicable in $s'$ then
11                     $marks \leftarrow marks \cup \{j\}$;
12                 else
13                     $s' \leftarrow \gamma(s', a_j)$;
14         if $s' \models G$ then
15             $removed \leftarrow removed \cup marks$;
16         else
17             $s \leftarrow \gamma(s, a_i)$;
18 $\Pi' \leftarrow \langle a_i \mid a_i \in \Pi, i \notin removed \rangle$;
19 return $\Pi'$;
```

Plan Action Landmarks

In the literature, *landmarks* are statements over facts or actions that must be true for each plan (Hoffmann, Porteous, and Sebastia 2004). It has been shown that landmarks can be leveraged for computing heuristics (Richter and Westphal 2010; Helmert and Domshlak 2009).

Inspired by the concept of Action Landmarks, we define a notion *Plan Action Landmark* that represents an action that is not part of any set of redundant actions in a given plan. Hence, similarly to Action Landmarks, which are actions that must be present in every plan, Plan Action Landmarks cannot be removed from a given plan.

Definition 2. *Let* Π *be a plan for a planning task* T*. Action* $a \in \Pi$ *is called a* **Plan Action Landmark** *for* Π *and* T *if and only if there does not exist a set of redundant actions* m *in* Π *such that* $a \in m$*.*

In general, deciding whether an action a is a part of some set of redundant actions in a plan Π is NP-complete. We can non-deterministically find a reduction of Π without a and validate its correctness in polynomial time, thus the problem is in NP. NP-hardness can be straightforwardly derived from the proof of Theorem 2 of Fink and Yang (1992) in which they reduce the 3-SAT problem to the problem of existence of a proper reduction of a plan. Hence, deciding whether an action is a Plan Action Landmark is intractable (co-NP-compete). On the other hand, identifying some Plan Action Landmarks can be done in polynomial time. Our method for Plan Action Landmark discovery is inspired by the back-chaining method of (fact) Landmark discovery (Hoffmann, Porteous, and Sebastia 2004). Our method is based on two claims. Firstly, actions that are the only achievers of goal facts are Plan Action Landmarks. Secondly, if an action is the only achiever of a fact for a Plan Action Landmark, the action is also Plan Action Landmark. The claims are proven in the following propositions. We denote as $\mathit{eff}_{max}(a)$ the set of facts that can possibly be achieved by an action a, i.e., $\mathit{eff}_{max}(a) = \{(v, val) \mid (c, e) \in \mathit{eff}_{cond}(a), e[v] = val\}$. Also, (partial) variable assignments will be, in this section, represented as sets of facts.

Algorithm 2: Plan Action Landmark Discovery

```
Input  : Planning task $T = (V, A, I, G)$, a plan
         $\Pi = \langle a_1, \dots, a_n \rangle$ for $T$
Output: Plan Action Landmarks
1  for $p \in \{(v, val) \mid v \in V, val \in D(v)\}$ do
2      $achievers[p] \leftarrow \emptyset$;
3  for $p \in I$ do
4      $achievers[p] \leftarrow achievers[p] \cup \{0\}$;
5  for $i \leftarrow 1$ to $n$ do
6      for $p \in \mathit{eff}_{max}(a_i)$ do
7          $achievers[p] \leftarrow achievers[p] \cup \{i\}$;
8  $landmarks \leftarrow \emptyset$;
9  for $p \in G$ do
10     if $|achievers[p]| = 1$ then
11         $landmarks \leftarrow landmarks \cup achievers[p]$;
12 for $i \leftarrow n$ to $1$ do
13     for $p \in \mathit{eff}_{max}(a_i)$ do
14         $achievers[p] \leftarrow achievers[p] \setminus \{i\}$;
15     if $i \in landmarks$ then
16         for $p \in pre(a_i)$ do
17             if $|achievers[p]| = 1$ then
18                 $landmarks \leftarrow$
                     $landmarks \cup achievers[p]$;
19 return $landmarks \setminus \{0\}$;
```

Proposition 3. *Let* $\Pi = \langle a_1, \dots, a_n \rangle$ *be a plan for a planning task* $T = (V, A, I, G)$ *and* a_i *be an action from* Π*. If there exists* $p \in (G \cap \mathit{eff}_{max}(a_i))$ *such that* $p \notin (I \cup \bigcup_{k=1}^{i-1} \mathit{eff}_{max}(a_k) \cup \bigcup_{l=i+1}^{n} \mathit{eff}_{max}(a_l))$*, then* a_i *is a Plan Action Landmark for* Π *and* T*.*

Proof. Since $p \in G$, it has to become true at some point in the state trajectory of Π. From the assumption, p is not true in the initial state nor can be achieved by any action from Π other than a_i. Hence removing a set of actions including a_i from Π cannot result in a valid plan as p will no longer become true (and the goal will not be achieved). Consequently, a_i is a Plan Action Landmark for Π and T. ☐

Proposition 4. *Let* $\Pi = \langle a_1, \dots, a_n \rangle$ *be a plan for a planning task* $T = (V, A, I, G)$ *and* a_i *be an action from* Π*. If there exist a Plan Action Landmark* a_j *with* $j > i$ *and* $p \in (pre(a_j) \cap \mathit{eff}_{max}(a_i))$ *such that* $p \notin (I \cup \bigcup_{k=1}^{i-1} \mathit{eff}_{max}(a_k) \cup$

$\bigcup_{l=i+1}^{j-1} \mathit{eff}_{max}(a_l)$), *then* a_i *is a Plan Action Landmark for* Π *and* T.

Proof. Since $p \in \mathit{pre}(a_j)$ and a_j is a Plan Action Landmark, p has to become true at some point in the state trajectory of $\langle a_1, \ldots, a_{j-1}\rangle$. From the assumption, p is not true in the initial state nor can be achieved by any action from $\langle a_1, \ldots, a_{j-1}\rangle$ other than a_i. Analogously to the proof of Proposition 3, we can imply that removing a_i would always invalidate precondition of a_j. Because a_j cannot be removed from Π (it is a Plan Action Landmark), a_i is a Plan Action Landmark for Π and T too. □

Our method for Plan Action Landmarks discovery is depicted in Algorithm 2. Firstly, for each fact, we identify what actions (including the initial state) can possibly achieve it (Lines 3–7). Then, we look for actions that are the only achievers of some goal fact (Lines 9–11). These actions are according to Proposition 3 Plan Action Landmarks. Then, we iterate backwards through Π while looking for actions that are the only achievers of some precondition for a Plan Action Landmark (Lines 12–18). According to Proposition 4 these actions are also Plan Action Landmarks. Algorithm 2 runs in linear time (assuming that the size of action preconditions and effects is constant with respect to the plan length).

Proposition 5. *The algorithm 2 returns a set of Plan Action Landmarks.*

Proof. Lines 3–7 fill the *achievers* map such that at the end, for each fact p, *achievers*$[p]$ will contain the set of actions from Π (including the initial state) that possibly achieves p. Note that using $\mathit{eff}_{max}(a)$ for each action a means that the sets *achievers*$[p]$ are overapproximations of possible achievers of p (i.e., *achievers*$[p]$ is a superset of a set of actual achievers of p in Π).

Lines 9–11 identify Plan Action Landmarks as actions that are the only possible achievers of some goal fact (according to Proposition 3), i.e., for a goal fact p if *achievers*$[p]$ contains only one element, the action listed in *achievers*$[p]$ is a Plan Action Landmark.

The for loop in Lines 12–18 goes backwards through Π. After the for loop in Lines 13–14, the *achievers* map considers only achievers of facts up to a_{j-1} (including the initial state). Lines 15–18 identify Plan Action Landmarks as actions that are the only possible achievers of some precondition of a_j which is a Plan Action Landmark (according to Proposition 4). Again, if for some fact $p \in \mathit{pre}(a_j)$, where a_j is a Plan Action Landmark, *achievers*$[p]$ contains only one element, the action listed in *achievers*$[p]$ is Plan Action Landmark.

So far, the initial state was handled by algorithm as possible achiever (and as an action). Hence the initial state (if present in *landmarks*) is removed in Line 18, so the returned set of Plan Action Landmarks does not contain it (the initial state is not an action and hence cannot technically be a Plan Action Landmark). □

Algorithm 3: Action Elimination with Plan Action Landmarks and Action Cycles

```
Input : Planning task $T = (V, A, I, G)$, a plan $\Pi = \langle a_1, \ldots, a_n\rangle$ for $T$
Output: A reduction $\Pi'$
1  $removed \leftarrow \emptyset$;
2  $marks \leftarrow \emptyset$;
3  $landmarks \leftarrow$ Alg. $2(T, \Pi)$;
4  $s \leftarrow I$;
5  for $i \leftarrow 1$ to $n$ do
6      if $i \notin removed$ and $i \notin landmarks$ then
7          $marks \leftarrow marks \cup \{i\}$;
8          $s' \leftarrow s$;
9          $cycle \leftarrow false$;
10         $violated \leftarrow false$;
11         $x \leftarrow eff(a_i, s')$;
12         for $j \leftarrow i + 1$ to $n$ do
13             if $j \notin removed$ then
14                 if not $violated$ and $\exists(c, e) \in eff_{cond}(a_j) : vars(c) \cap vars(x) \neq \emptyset$ then
15                     $violated = true$;
16                 if $a_j$ is not applicable in $s'$ then
17                     if $j \notin landmarks$ then
18                         $marks \leftarrow marks \cup \{j\}$;
19                         if not $violated$ then
20                             $\forall v \in vars(eff(a_j, s')) : x[v] \leftarrow eff(a_j, s')[v]$;
21                             if $s' \models x$ then
22                                 $cycle \leftarrow true$;
23                                 break;
24                     else
25                         break;
26                 else
27                     if not $violated$ and $\exists(v, val) \in eff(a_j, s') : v \in vars(x), (v, val) \notin x$ then
28                         $violated \leftarrow true$;
29                     $s' \leftarrow \gamma(s', a_j)$;
30         if $cycle$ or $s' \models G$ then
31             $removed \leftarrow removed \cup marks$;
32         else
33             $s \leftarrow \gamma(s, a_i)$;
34         $marks \leftarrow \emptyset$;
35     $s \leftarrow \gamma(s, a_i)$;
36 $\Pi' \leftarrow \langle a_i \mid a_i \in \Pi, i \notin removed\rangle$;
37 return $\Pi'$;
```

Algorithm 2 is performed before AE or GAE. To leverage the information about Plan Action Landmarks, AE and GAE can be easily modified. The condition in Line 5 of Algorithm 1 would include also a check whether a_i is not Plan Action Landmark. Line 11 of Algorithm 1 would consist of a check whether a_j is a Plan Action Landmark and if so, the for loop in Lines 8–13 will be terminated. Integration of Plan Action Landmarks in AE and GAE does not change their asymptotic time complexity.

Action Cycles

Chrpa, McCluskey, and Osborne (2012a) and (2012b) studied under which conditions pairs of inverse actions (i.e., actions that if consecutively applied results in the same state as before their application) can be removed from plans while maintaining their validity. Informally speaking, if actions placed in between the inverse actions do not require facts provided by the first inverse action or do not interfere with effects of the second inverse action, the pair of inverse actions is redundant.

In this paper, we generalise this idea to any action sequence. An action sequence whose application leads to the same state is redundant (form a set of redundant actions) if being part of a plan. The AIRS (Anytime Iterative Refinement of a Solution) system (Estrem and Krebsbach 2012), for instance, is able to identify such situations. However, it might be the case that such action sequences are interleaved by other actions in plans. We introduce the notion of an *Action Cycle* which represents a subsequence of actions in a plan (not necessarily consecutive) that if removed from the plan the part of the state trajectory after the last action from that subsequence remains the same as for the original plan.

Definition 6. *Let $\Pi = \langle a_1, \ldots, a_n\rangle$ be a plan for a planning task $T = (V, A, I, G)$ and $\langle I, s_1, \ldots, s_n\rangle$ be the Π's state trajectory. An* **Action Cycle***, $C = \langle a_{\iota_1}, \ldots, a_{\iota_m}\rangle$, is a subsequence of actions from the plan Π (not necessarily consecutive) such that $\gamma(s_{\iota_1}, \langle a_{\iota_1}, a_{\iota_1+1}, \ldots, a_{\iota_m-1}, a_{\iota_m}\rangle \setminus C) = s_{\iota_m}$.*

Definition 6 provides a blueprint on how Action Cycles can be identified within (G)AE. Instead of keeping only the state s' (see Algorithm 1), we will also keep another state s''. Before the j for loop of Alg. 1, we insert $s'' = \gamma(s, a_i)$. Then, we insert $s'' = \gamma(s'', a_j)$ in between Lines 9 and 10. Finally, we modify the condition on Line 14 to "$s' \models G$ or $s'' = s'$". By such an approach, we can identify all Action Cycles (G)AE comes across. However, such an approach is computationally demanding as we have to keep two states (one for each trajectory) and compare them in each step.

We have, hence, proposed an "incomplete" approach that tracks only variables that are modified by marked actions. If the conditions of conditional effects of processed actions (both actions forming an Action Cycle or interleaving actions) do not interfere with the variables modified by the actions in the Action Cycle, then the actions forming an Action Cycle are redundant and can be safely removed from the plan. The idea is formalised in the following proposition.

Proposition 7. *Let $\Pi = \langle a_1, \ldots, a_n\rangle$ be a plan for a planning task $T = (V, A, I, G)$ and $\langle I, s_1, \ldots, s_n\rangle$ the Π's state trajectory. Let $C = \langle a_{\iota_1}, \ldots, a_{\iota_m}\rangle \subseteq \Pi$ be a subsequence of actions from Π (not necessarily consecutive). Let x^l ($1 \leq l \leq m$) be a (partial) variable assignment such that $x^l[v] = \mathit{eff}(a_{\iota_l}, s_{\iota_l-1})[v]$ iff $v \in vars(\mathit{eff}(a_{\iota_l}, s_{\iota_l-1}))$, $x^l[v] = x^{l-1}[v]$ iff $l > 1$ and $v \in vars(x^{l-1}) \setminus vars(\mathit{eff}(a_{\iota_l}, s_{\iota_l-1}))$, or $x^l[v]$ is undefined otherwise. We assume the following conditions hold:*

- *For each action a_k with $\iota_1 < k < \iota_m$ and $k \neq \iota_2, \ldots, k \neq \iota_{m-1}$, it holds that a_k is applicable in $\gamma(I, \langle a_1, \ldots, a_{k-1}\rangle \setminus C)$*
- *For each i, j with $1 \leq i < m$ and $\iota_i < j \leq \iota_{i+1}$, it holds that $\forall (c, e) \in \mathit{eff}_{cond}(a_j) : vars(c) \cap vars(x^i) = \emptyset$*

If $\gamma(I, \langle a_1, \ldots, a_{\iota_m}\rangle \setminus C) \models x^m$, then C is an Action Cycle and also a set of redundant actions.

Proof sketch. The first assumption ensures that actions interleaving with the actions forming the Action Cycle C remain applicable even if C is removed from the plan Π. The (partial) variable assignment x^i contains the latest values of the variables the sequence of actions $\langle a_{\iota_1}, \ldots, a_{\iota_i}\rangle$ achieves. Consequently, x^i is defined only for variables that $\langle a_{\iota_1}, \ldots, a_{\iota_i}\rangle$ might modify. The second assumption ensures that conditional effects of all actions from $\langle a_{\iota_i+1}, \ldots, a_{\iota_{i+1}}\rangle$ are not affected by possibly different values of variables from x^i. That said, removing C from Π does not change the outcome of conditional effects of the interleaving actions.

The (partial) variable assignment x^m contains the latest values of the variables the actions from C achieve. Let $s' = \gamma(I, \langle a_1, \ldots, a_{\iota_m}\rangle \setminus C)$ be the state obtained by applying actions from Π while skipping the actions from C until the last action from C is considered. If $s' \models x^m$, then the actions from C cannot change the value of any variable from s'. Hence, $s' = \gamma(I, \langle a_1, \ldots, a_{\iota_m}\rangle)$, which means that C is an Action Cycle (note that it is guaranteed that all actions interleaving with C have the same effects along both trajectories; otherwise this would not necessary hold). Since applying the rest of the plan Π in s' straightforwardly results in a goal state, C is also a set of redundant actions.

$\square$

Proposition 7 provides guidelines for how Action Cycle detection can be incorporated into the (G)AE algorithm. Algorithm 3 presents an enhanced AE algorithm by considering Action Cycles and Plan Action Landmarks. As the latter enhancement is straightforward (as mentioned in the previous section) we will elaborate in detail only for the Action Cycle enhancement.

If marked actions form an Action Cycle during the iteration of AE, we know that these marked actions can be safely removed without having to go through the rest of the plan. Throughout the action marking process, we keep track whether actions follow the "conditional effect interference" condition (i.e., variables for conditions are distinct from those modified by the marked actions) – see Line 14. Note that considering this condition also for marked actions (the a_{ι_i} actions from Proposition 7) is important as in the algorithm we can use the state trajectory that does not consider marked actions rather than the original state trajectory (as considered in Proposition 7) since these state trajectories can differ only in values of variables from x. As long as the condition is not violated we keep updating the (partial) variable assignment x consisting of changes to the values the marked actions would have done so far (Lines 19–20). If x is entailed by the current state (without considering the marked actions), we have found an Action Cycle (Lines 21–22) and currently marked actions can be removed without the necessity to go through the rest of the plan (Lines 30–31). Additionally, we require that interleaving actions do not change the value of any variable from x (Line 27). Such a condition is not required to ensure soundness of the Action Cycle detection process, on the other hand, its rationale is based on our observation that actions interleaving Action Cycles rarely change values of variables from x. Note that Algorithm 3, in case either the "conditional effect interference" or the "variable modification" condition is violated, continues like the standard AE (with Plan Action Landmarks).

Note that we do not explicitly check whether some marked action invalidates a precondition of another action (possibly interleaving an Action Cycle). This is done implicitly by AE as it marks every action whose precondition is not met in the current state. So, such an action is considered as a part of an Action Cycle until either it is confirmed that the action is actually a part of an Action Cycle, or it is disproved.

It can be observed that integration of Action Cycle detection in AE and GAE does not change their asymptotic time complexity.

Example

Let us consider a simple example from the well known Logistics domain. We have two trucks – trk1, trk2, two packages – pkg1, pkg2 and three locations – A, B, C. Initially, both trucks are at location A, packages pkg1, pkg2 are at locations A and B, respectively. The goal is to deliver both packages to C. We can observe that an action sequence $\langle$drive(trk1,A,B), drive(trk2,A,B), load(trk2,pkg2,B), drive(trk1,B,C), drive(trk2,B,C), drive(trk1,C,A), load(trk1,pkg1,A), drive(trk1,A,C), unload(trk1,pkg1,C), unload(trk2,pkg2,C)$\rangle$ is a plan. We can observe that the load and unload actions are Plan Action Landmarks since the unload actions are the only achievers for the goals and the load actions are the only achievers for the respective unload actions. Also, the drive actions concerning trk2 are Plan Action Landmarks as are the only achievers for the respective load and unload actions (concerning the same truck). On the other hand, the drive actions concerning trk1 are not Plan Action Landmarks because they are not the only achievers for the respective load and unload actions. The action subsequence $\langle$drive(trk1,A,B), drive(trk1,B,C), drive(trk1,C,A)$\rangle$ forms an Action Cycle and is redundant because trk1 after applying the actions returns back to A and the interleaving actions concerning trk2 are not influenced by them (in terms of Proposition 7).

Experiments

The aim of the experiments is to demonstrate that the proposed enhancements generally improve performance of the AE and GAE algorithms.

The proposed algorithms were implemented in *C++* and the translation from PDDL to the SAS (or FDR) representation was done by the Fast Downward translator (Helmert 2006). Experiments ran on a computer with processor Intel® Core™ i7-7700HQ CPU (2.8 GHz, L2 Cache 1 MB, L3 Cache 6 MB) and 8 GB RAM (2133 MHz)[1].

We used five state-of-the-art planners that were successful in the recent IPCs. The considered planners are *Cerberus* (Katz 2018), *Freelunch-Madagascar* (Balyo and Gocht 2018), *LAMA 2011* (Richter and Westphal 2010), *BFWS-Preference* (Francès et al. 2018) and *YAHSP3* (Vidal 2014). For generating plans, we have adopted the settings from the Agile Tracks of the IPC, that is, one CPU core, 8 GB of RAM and 300 seconds of CPU time.

We considered all benchmarks from the Agile Tracks of IPC 2014 and 2018 (800 planning tasks in total). The planners managed to generate 938 plans in total.

Results

The per domain results of the introduced enhancements of AE and GAE (as well as their original versions) are presented in Table 1. In 7 out of 26 domains, no redundant actions were identified at all, in 4 domains (G)AE optimised plans by more than 10% on average (in City Car by nearly $1/3$), yielding overall optimisation rate of about 5% on average. Note that GAE outperforms AE in terms of "optimisation power" rather marginally and the results support the observation of Balyo, Chrpa, and Kilani (2014). Note that comparing AE with GAE is not our primary concern in this paper. On the other hand, the results of GAE can highlight the "power" of the proposed enhancements more profoundly than the results of AE.

Plan Action Landmarks were identified in every domain, ranging from 5.1% (Termes) to 100% (Organic Synthesis) of actions in all plans. In 9 domains more than 80% of actions were identified as Plan Action Landmarks and on average, roughly 60% of actions in all plans were Plan Action Landmarks. Note that if all actions in a plan are Plan Action Landmarks (like in Organic Synthesis), then the plan is a minimal reduction of the corresponding planning task. It can be observed that Plan Action Landmarks can be identified in the order of milliseconds at most (see Table 1). The results also show that the use of Plan Action Landmarks improves runtimes of both AE and GAE considerably (Plan Action Landmark identification is included), in a few cases even by a few orders of magnitude (e.g. Visitall, Openstacks). On top of that, our results show that sometimes even small percentage of found Plan Action Landmarks may result in considerable performance improvement (e.g. Barman or Settlers).

Action Cycles were detected on plans from 13 domains (out of 26 domains we considered). This is one of the reasons why the results are rather mixed and overall slightly worse when compared to the original AE and GAE, respectively. Higher percentage of identified Action Cycles together with higher optimisation rate often resulted in better performance (e.g. Hiking) but there are some exceptions of the rule (e.g. Transport). With regards to the complete Action Cycle detection (see the Action Cycles section), it was slower by more than 5% than our "incomplete" approach

[1]Our implementation and experimental data can be found here: https://gitlab.com/ctu-fee-fras/public/speeding-up-redundant-action-detection-icaps-2022

					AE						GAE					
	#	t-Pl	Lm	t-Lm	Opt	AC	t	t-C	t-L	t-CL	Opt	AC	t	t-C	t-L	t-CL
Agricola	18	2975.9	74.5	0.008	0.0	-	0.018	0.060	**0.015**	**0.015**	0.0	-	0.019	0.060	**0.015**	0.016
Barman	57	2203.5	19.5	0.022	8.2	91.24	11.453	10.519	2.279	**1.361**	8.3	94.0	141.776	131.841	32.849	**17.173**
Caldera	29	1571.7	97.8	0.013	2.2	0.0	0.061	0.068	0.019	**0.017**	2.2	0.0	0.124	0.131	**0.025**	0.026
Caldera (split)	12	156.2	51.1	0.004	8.3	0.0	0.038	0.046	**0.018**	0.020	8.3	0.0	0.075	0.091	**0.033**	0.036
Cave Diving	27	1923.1	63.7	0.002	0.5	0.0	**0.005**	0.008	**0.005**	**0.005**	0.5	0.0	0.007	0.010	**0.006**	**0.006**
City Car	28	1787.5	48.0	0.004	32.4	40.9	0.334	0.285	0.211	**0.160**	32.4	31.4	1.979	1.807	1.245	**1.092**
Data Network	34	2135.4	57.9	0.009	12.6	54.8	1.198	1.129	0.435	**0.332**	12.7	56.1	18.429	16.248	7.991	**4.919**
Flashfill	27	1326.0	61.8	0.024	0.0	-	0.182	0.184	0.121	**0.117**	0.0	-	0.178	0.180	**0.113**	0.117
Floortile	47	417.7	59.1	0.006	8.2	61.6	0.148	0.173	0.021	**0.018**	8.2	56.7	0.619	0.679	0.059	**0.051**
GED	68	2139.6	43.6	0.017	1.3	87.5	0.184	0.366	**0.054**	0.082	1.3	87.5	0.319	0.660	**0.104**	0.222
Hiking	47	2160.8	50.6	0.015	10.8	73.1	3.519	1.524	2.622	**0.634**	11.9	73.2	707.303	347.963	469.055	**115.079**
Child Snack	25	755.8	87.4	0.003	4.3	25.0	0.102	0.110	0.011	**0.008**	4.3	25.0	0.352	0.372	**0.018**	**0.018**
Maintenance	44	475.4	93.4	0.031	5.8	0.0	10.946	11.259	**0.996**	1.020	5.8	0.0	123.718	127.385	**10.080**	10.341
Nurikabe	20	948.0	91.1	0.009	0.0	-	0.423	0.436	**0.081**	0.083	0.0	-	0.434	0.443	**0.084**	0.087
Openstacks	53	4910.6	82.2	0.052	0.0	-	100.082	100.911	**0.885**	0.888	0.0	-	100.382	101.197	0.892	**0.881**
Org. Synth.	9	977.3	100.0	0.001	0.0	-	**0.002**	**0.002**	**0.002**	**0.002**	0.0	-	0.002	0.002	**0.001**	**0.001**
Org. Syn. (spl.)	21	1834.5	89.3	0.013	0.0	-	0.150	0.157	0.069	**0.068**	0.0	-	0.147	0.155	**0.068**	**0.068**
Parking	45	4444.0	64.8	0.014	0.3	100.0	0.078	0.151	0.040	**0.039**	0.3	100.0	0.095	0.173	**0.041**	0.042
Settlers	25	1993.7	26.2	0.153	11.3	48.8	10.053	10.073	2.396	**2.212**	11.7	45.7	65.368	63.643	16.527	**14.807**
Snake	26	1572.0	36.1	0.015	0.0	-	**0.065**	0.149	0.069	0.122	0.0	-	**0.067**	0.154	0.068	0.120
Spider	26	2001.4	63.4	0.031	0.3	0.0	0.487	0.831	0.246	**0.243**	0.3	0.0	0.552	0.942	**0.272**	0.275
Termes	31	1439.9	5.1	0.013	5.3	47.1	1.531	1.800	**0.887**	0.982	6.1	50.6	37.091	42.249	**18.310**	21.491
Tetris	37	3603.6	13.3	0.026	5.8	74.7	5.994	5.860	5.146	**4.923**	5.8	77.6	21.954	21.213	19.465	**19.088**
Thoughtful	69	1824.2	80.3	0.025	4.1	66.4	0.852	0.914	0.312	**0.194**	4.1	70.0	6.444	4.935	3.461	**1.406**
Transport	53	3020.9	39.5	0.020	8.1	86.8	4.848	4.911	0.716	**0.409**	8.1	88.0	146.769	135.495	35.334	**15.401**
Visitall	60	3063.6	85.8	0.308	0.2	0.0	3300.63	3370.48	**67.394**	68.747	0.2	0.0	21072.2	21515.1	**454.287**	457.724
$\sum$	938	51662.3	59.2	0.840	5.0	52.9	3453.38	3522.40	85.049	**82.698**	5.1	52.8	22446.4	22513.1	1070.40	**680.487**

Table 1: Per domain results (across all generated plans) for AE and GAE and their enhancements. "#" represents the number of generated plans. "t-Pl" represents the total plan generation time (in seconds). "Lm" represents mean of the percentage of found Plan Action Landmarks in plans. "t-Lm" represents the total Plan Action Landmark identification time (in seconds). "Opt" represents mean of the plan improvement percentage. "AC" represents the mean of percentage of identified Action Cycles w.r.t all identified sets of redundant actions. Prefix "t-" represents the total optimisation time (in seconds). Suffixes "-L", "-C" and "-LC" represent Plan Action Landmark, Action Cycle and both enhancements together, respectively.

(presented in Algorithm 3) in AE despite identifying 4% more Action Cycles. Removing the condition on Line 27 of Algorithm 3 resulted in about 2% slowdown with respect to our "incomplete" approach (in AE). Note that the results of these variants of Action Cycle detection are not listed in Table 1.

The combination of both enhancements achieved the best results overall (see Table 1) and, specifically, in 17 domains for AE and 14 domains for GAE. We can therefore observe that the use of Plan Action Landmarks can mitigate larger overheads related to Action Cycle detection. In particular, Plan Action Landmarks can often identify earlier that a set of marked actions is not redundant saving some time that would have been spent on trying to prove whether that set of redundant actions is an Action Cycle.

We would like to emphasise that AE enhanced by both Action Cycles and Plan Action Landmarks runs on average in an order of tens of milliseconds per plan. Visitall is a notable exception as the number of actions in all plans is about 205000, i.e., about 3500 actions per plan, yielding the optimisation time being more than 1 second per plan. That said, we can optimise plans in (almost) realtime while improving their quality by about 5% on average.

Discussion

The use of Plan Action Landmarks has shown to be the most efficient enhancement for both AE and GAE. Plan Action Landmark discovery prior running AE and GAE is a "one-time investment" of CPU time (albeit very small as the results indicate) as the use of Plan Action Landmarks does not possess any time overheads. On top of that, Plan Action Landmarks were discovered in every domain and hence they can be considered as a general type of knowledge that can be leveraged for early detection of actions that are not redundant. We believe that Plan Action Landmarks can be leveraged in more sophisticated optimisation techniques such as BDPO2 (Siddiqui and Haslum 2015) because they can provide valuable information on how important facts are being achieved in plans and that, we believe, can be exploited for more "targeted" optimisation.

The results of the Action Cycle enhancement are more mixed. Action Cycle detection introduces overheads to the original (G)AE, mostly, by maintaining the (partial) variable assignment x. It hence comes with no surprise that if none or a few Action Cycles are identified during the optimisation process, the overheads will outweigh the benefits and cause higher CPU time consumption than would have been spent by the original optimisation method. On the other hand, Action Cycles might be leveraged for optimising partial plans in online planning as partial plans usually do not achieve the goal and hence Action Cycles are only redundant actions that can possibly be identified in these partial plans.

Combining both enhancements brought promising results because of a synergy that exists between them. As mentioned above Plan Action Landmarks often mitigate overheads of the Action Cycle detection by early discovery that the marked actions are not redundant (and hence cannot form an Action Cycle). AE with both enhancements can be used (almost) in realtime as the optimisation time is usually in the order of at most tens of milliseconds while achieving decent reductions in plan costs (5% on average).

Also, we can usually find out whether and how often Action Cycles might be identified in a given domain by analysing a small sample of plans. Therefore, we might be able to predict whether Action Cycles will work well on a given domain (and a given planner).

Related Work

Identifying and removing redundant actions, which is the focus of this paper, can be understood as a "pre-optimisation" technique as it can be followed by more sophisticated optimisation techniques (as, for instance, shown in the ARAS system (Nakhost and Müller 2010a)). A foundational work of Fink and Yang (1992) studied redundant actions in totally and partially ordered plans. In particular, they defined four categories of redundant actions and provided complexity results for each of the categories. One of the categories, Greedily Justified actions, is a foundation for the Action Elimination algorithm (Alg. 1) introduced by Nakhost and Müller (2010a). Greedy Action Elimination, introduced by Balyo, Chrpa, and Kilani (2014), is an extension of Action Elimination aiming at removing the most expensive set of redundant actions in each iteration. Chrpa, McCluskey, and Osborne (2012b) proposed a technique for identifying redundant pairs of inverse actions. Later the technique was extended to consider "nested" pairs inverse actions (Chrpa, McCluskey, and Osborne 2012a). Balyo, Chrpa, and Kilani (2014) addressed the problem of finding a minimal reduction (proven to be NP-complete (Nakhost and Müller 2010b)) by compiling it to MaxSAT and have empirically shown that (G)AE often find minimal reductions.

Other plan optimisation techniques usually involve iterative identification of (very) suboptimal subsequences of actions which are then optimised. Westerberg and Levine (2001) proposed a technique based on Genetic Programming, however, it is unclear how much CPU time the optimisation process consumes. AIRS (Estrem and Krebsbach 2012) identifies suboptimal subsequences of actions according to a heuristic estimation. If the heuristic estimation indicates that states might be closer than they are, then an optimal planning technique is used to "connect" these states. Balyo, Barták, and Surynek (2012a,b) exploit a similar philosophy for optimising parallel plans via SAT planning. Planning Neighborhood Graph Search, which is a part of the ARAS system (Nakhost and Müller 2010a), expands a limited amount of nodes along the state trajectory. Then, by applying Dijsktra's algorithm, it looks for a better quality plan. Siddiqui and Haslum (2013) deorder plan into partially ordered "blocks" which are then optimised. A follow up work of Siddiqui and Haslum (2015) presents a system called BDPO2 that incorporates sophisticated strategies for selecting "windows" (extracted from "blocks") that are optimised.

Conclusion

In this paper, we proposed two improvements of the plan optimisation algorithms – Action Elimination (Nakhost and Müller 2010a) and Greedy Action Elimination (Balyo, Chrpa, and Kilani 2014) – that aim at identifying and removing sets of redundant actions from plans in polynomial time. Firstly, we identify Plan Action Landmarks that are actions which are not part of any set of redundant actions and thus they cannot be removed from the plan. Then, if (G)AE tries to remove a Plan Action Landmark, then we immediately know that it and other actions considered for removal cannot be removed. Secondly, we detect Action Cycles that if removed the remaining state trajectory will not change. Hence, if we know that the actions considered for removal form an Action Cycle, we can safely remove them from the plan without having to explore the rest of the plan.

We evaluated the proposed enhancements on all benchmarks from the Agile Tracks of IPC 2014 and 2018. For generating plans we used five state-of-the-art planners, successful in the recent editions of the IPC. The results have shown that Plan Action Landmarks provide a considerable performance improvement as they can be identified in all considered domains. Action Cycle detection was not that successful overall as in half of the considered domains no Action Cycles were identified. On the other hand, in domains such as *Hiking*, where Action Cycles were often detected, the performance improvement was much more apparent. Both enhancements together achieved the best results in 17 and 14 domains (out of 26) in AE and GAE, respectively. Also, AE with both enhancements can operate in (almost) realtime, yet optimising plans by about 5% on average.

For our future work, we would like to adapt our techniques for more expressive forms of planning. For example, temporal plans contain actions that are partially ordered and (possibly) applied simultaneously. Methods such as (G)AE (including our enhancements) have to be adapted to deal with partial ordering of actions and concurrency of actions. To give another example, in non-deterministic planning, our techniques can be leveraged during generating strong (cyclic) plans. In particular, generating strong (cyclic) plans by "determinisation", i.e., generating a deterministic plan for each non-deterministic alternative, as, for example, PRP does (Muise, McIlraith, and Beck 2012), provides an interesting opportunity to leverage our plan optimisation techniques. We would like to also investigate whether and how some techniques (e.g. Action Cycles) can be leveraged during the plan generation.

Acknowledgements

This research was funded by the Czech Science Foundation (project no. 18-07252S) and by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 “Research Center for Informatics”.

References

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Comput. Intell.*, 11: 625–656.

Balyo, T.; Barták, R.; and Surynek, P. 2012a. On Improving Plan Quality via Local Enhancements. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*, 154–156. AAAI Press.

Balyo, T.; Barták, R.; and Surynek, P. 2012b. Shortening Plans by Local Re-planning. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, 1022–1028. IEEE Computer Society.

Balyo, T.; Chrpa, L.; and Kilani, A. 2014. On Different Strategies for Eliminating Redundant Actions from Plans. In Edelkamp, S.; and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, August 15-17 2014*, 10–18. AAAI Press.

Balyo, T.; and Gocht, S. 2018. The Freelunch Planning System Entering IPC 2018. In *Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*, 5–8.

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

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012a. Determining Redundant Actions in Sequential Plans. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, 484–491. IEEE Computer Society.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 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, Atibaia, São Paulo, Brazil, June 25-19, 2012*, 338–342.

Estrem, S. J.; and Krebsbach, K. D. 2012. AIRS: Anytime Iterative Refinement of a Solution. In Youngblood, G. M.; and McCarthy, P. M., eds., *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference, Marco Island, Florida, USA, May 23-25, 2012*. AAAI Press.

Fink, E.; and Yang, Q. 1992. Formalizing Plan Justications. In Glasgow, J. I.; and Hadley, R. F., eds., *Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, 9–14.

Francès, G.; Geffner, H.; Lipovetzky, N.; and Ramírez, M. 2018. Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants. In *Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*, 23–27.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artif. Intell.*, 143(2): 219–262.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, 162–169. AAAI Press.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *J. Artif. Intell. Res.*, 22: 215–278.

Katz, M. 2018. Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection. In *Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*, 47–50.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

Nakhost, H.; and Müller, M. 2010a. 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 Press.

Nakhost, H.; and Müller, M. 2010b. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement - Extended Version. *Technical Report TR 10-01, Dept. of Computing Science. University of Alberta*.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Siddiqui, F. H.; and Haslum, P. 2013. Plan Quality Optimisation via Block Decomposition. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2387–2393. IJCAI/AAAI Press.

Siddiqui, F. H.; and Haslum, P. 2015. Continuing Plan Quality Optimisation. *J. Artif. Intell. Res.*, 54: 369–435.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. In *The 2014 International Planning Competition - Deterministic Track, Description of Participating Planners*, 64–65.

Westerberg, C. H.; and Levine, J. 2001. Optimising Plans using Genetic Programming. In *Proceedings of ECP*, 423–428.