Eliminating Redundant Actions in Partially Ordered Plans — A Complexity Analysis

Conny Olz and **Pascal Bercher**
Institute of Artificial Intelligence, Ulm University, Germany
conny.olz@uni-ulm.de, pascal.bercher@uni-ulm.de

Abstract

In this paper we study the computational complexity of post-optimizing partially ordered plans, i.e., we investigate the problem that is concerned with detecting and deleting unnecessary actions. For totally ordered plans it can easily be tested in polynomial time whether a single action can be removed without violating executability. Identifying an executable *subplan*, i.e., asking whether k plan steps can be removed, is known to be ***NP**-complete*. We investigate the same questions for *partially ordered* input plans, as they are created by many search algorithms or used by real-world applications – in particular time-critical ones that exploit parallelism of non-conflicting actions. More formally, we investigate the computational complexity of removing an action from a partially ordered solution plan in which every linearization is a solution in the classical sense while allowing ordering insertions afterwards to repair arising executability issues. It turns out that this problem is ***NP**-complete* – even if just a single action is removed – and thereby show that this reasoning task is harder than for totally ordered plans. Moreover, we identify the structural properties responsible for this hardness by providing a fixed-parameter tractability (FPT) result.

Introduction

While most of today's planning systems produce *totally ordered* action sequences as solutions, many also produce *partially ordered* courses of action, such as partial-order causal link (POCL) planners (McAllester and Rosenblitt 1991; Penberthy and Weld 1992; Younes and Simmons 2003), temporal planners based on constrained posting (Vidal and Geffner 2006), and hierarchical planning systems, since many of the latter like FAPE (Dvořák et al. 2014; Bit-Monnot 2016) or PANDA (Schattenberg 2009; Bercher, Keen, and Biundo 2014) rely directly on POCL techniques (see, e.g., the overview by Bercher et al. (2016)). Partially ordered plans are basically a compact representation for a set of total-order plans. Mutually unordered actions can be executed simultaneously allowing to postpone the decision about the exact execution order until runtime, which makes these plans more flexible in time-sensitive applications (Vidal and Geffner 2006; Muise, Beck, and McIlraith 2016; Aghighi and Bäckström 2017).

We investigate the computational complexity of post-optimizing partially ordered plans. That is, we aim at removing redundant actions so that the remaining – shorter or cheaper – plan still solves the planning task. It goes without saying that cheaper solutions are (probably *always*) preferable. However, solving tasks *optimally* in the first place is not always feasible, since this may be much harder than solving them suboptimally (Helmert 2003). For instance, A^* with an admissible heuristic is known to explore an exponential search space even with "almost-perfect" heuristics on several standard planning benchmark domains (Helmert and Röger 2008). Thus, we pursue the approach of checking whether an action can be removed from a given solution.

Answering this question is not only relevant for optimizing a given plan, but also for related tasks: In the context of *mixed-initiative planning* – a well-known example being the MAPGEN system used for the Mars Exploration Rovers Spirit and Opportunity (Bresina and Morris 2007; Ai-Chang et al. 2004) – change requests have to be realized. Knowing the computational complexity of such requests (like removing an action) is one first step towards automatic support for human experts (Bäckström 1998; Behnke et al. 2016). Fink and Yang (1992) give the application of plan-reuse: Given we found a solution to some problem and we want to find a solution to a subproblem thereof, we can simply use post-optimization. The last possible application of our findings that we would like to mention is plan explanation (Seegebarth et al. 2012; Bercher et al. 2014). Here, justifications for actions in a plan are verbalized – and presented to a human user in the form of a so-called explanation. These justifications basically follow the chain of causal links in a plan to a goal, presenting all actions in between as justification for the action in question. Explanations like these are of doubtful use in case the action in question is actually redundant. Post-optimizing plans will thus improve trust in explanations if no redundant actions will remain since questions about such redundant actions cannot even be asked.

The outline of this paper is as follows. In the next section we provide an overview of related work. We continue

with the formal framework introducing the concept of partially ordered plans. We then introduce and investigate the post-optimization problem: Here, we show our main result that it is ***NP**-complete* to decide whether we can remove a (given) single plan step from such a solution plan, such that it can be turned into a solution again by adding ordering constraints. In the following section we introduce a fixed-parameter tractability result that sheds more light on the source of hardness of the problem. We also provide a respective fixed-parameter tractable (FPT) algorithm to solve the respective problem. We then discuss our findings and its implications. Finally, we conclude the paper.

Related Work

For a given totally ordered solution plan it can trivially be tested in polynomial time whether the removal of one or more given actions results again in a solution (since the result is again a totally ordered action sequence). Fink and Yang (1992) proved the ***NP**-completeness* of deciding whether there exists a group of removable actions (not necessarily consecutive) in a totally ordered plan, i.e., whether a given solution is free of any redundant actions. Independently, Nakhost and Müller (2010) showed a slightly more general result. They proved that it is ***NP**-complete* to decide whether there exist k actions that can be removed from a totally-ordered solution plan.

Some only loosely related work on theoretical investigations of partially ordered plans studied the complexity of finding an executable action linearization under various restrictions (Nebel and Bäckström 1994; Tan and Gruninger 2014), the complexity of finding a solution given a delete-relaxed model (Bercher et al. 2013), and the complexity of reordering actions to optimize ordering constraint-related optimality criteria (Bäckström 1998; Aghighi and Bäckström 2017).

There are also various techniques that aim at optimizing a given solution plan. Fink and Yang (1992) investigated partially ordered plans by presenting polynomial algorithms for removing some of its redundant actions. In contrast to the investigations done by us, their algorithms do not allow ordering insertion after redundant actions got deleted. This is allowed by Muise, Beck, and McIlraith (2016), who provide a partial weighted MaxSAT encoding to find *deorderings* (only remove ordering constraints) or *reorderings* (orderings can be changed) with or without action eliminations to find partially ordered plans with maximal flexibility, i.e., minimal number of ordering constraints. Their work does not investigate the respective computational complexity, however. Closely related, the approach by Say, Cire, and Beck (2016) also aims at improving a partially ordered plan's flexibility while allowing action elimination. For this, they introduce several optimization criteria and corresponding mixed-integer linear programming (MILP) models. Siddiqui and Haslum (2015) present another technique, based on improving subproblems, to optimize the cost of a given plan. In contrast to the other approaches mentioned such a plan, while being cheaper, must not necessarily be a subplan, however.

Formal Framework

We consider the problem of optimizing partially ordered plans by removing unnecessary actions from it. These plans are solutions to standard (STRIPS) planning tasks, which we define first. A planning task Π is a tuple (V, A, s_I, g), where V is a finite set of propositional state variables, $s_I \in 2^V$ is the initial state, and $g \subseteq V$ the goal description. A is a finite set of *actions* in the standard STRIPS notation, where each action $a \in A$ consists of a *precondition* $prec \subseteq V$, an *add list* $add \subseteq V$, and a *delete list* $del \subseteq V$. If $a = (prec, add, del)$, we also write $prec(a)$, $add(a)$, $del(a)$ to refer to the respective elements. As usual, an action a is called applicable in a state $s \in 2^V$ if and only if $prec(a) \subseteq s$. If a is applicable in s, then the state transition function $\gamma : A \times 2^V \to 2^V$ returns the state resulting from applying a to s, $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. A sequence of actions $\bar{a} = (a_0 a_1 \dots a_n)$ is called applicable to a state s_0 if there exists a sequence of states $s_1 \dots s_{n+1}$ such that for all $1 \leq i \leq n+1$ holds $\gamma(a_{i-1}, s_{i-1}) = s_i$. The state s_{n+1} is called the *result* from applying the sequence. A *goal state* is a state s with $g \subseteq s$.

Solutions to planning tasks are ordinarily defined in terms of totally ordered action sequences, as given next.

Definition 1. We call an action sequence $\bar{a}$ a *total-order (t.o.) plan* $\overline{P}$ (also *t.o. solution*) to a planning task (V, A, s_I, g) if and only if it is applicable to s_I and results in a goal state.

Normally, t.o. plans contain many orderings that are not required to ensure their executability. If certain actions do not depend on each other, they do not have to be ordered with respect to each other and could also be executed in parallel. This can be achieved by simply maintaining only a *partial order* among those actions. The standard semantics for such partially ordered action plans is that *all* of its linearizations are executable and generate a goal state.

Several planning algorithms do not produce t.o. plans, but partially ordered ones instead. These approaches ordinarily rely on so-called *causal links*, the main concept behind partial-order causal-link (POCL) algorithms (McAllester and Rosenblitt 1991; Penberthy and Weld 1992; Younes and Simmons 2003). Causal links are further exploited in temporal constraint-based planning (Vidal and Geffner 2006), but even in *state-based* satisficing search (Lipovetzky and Geffner 2011) as well as in optimal state-based search (Karpas and Domshlak 2012). Also many *hierarchical* planning approaches produce partially ordered plans, as many of them rely upon POCL planning techniques or structures as well (Bercher et al. 2016).

A *partial* plan is a tuple $P = (PS, \prec, CL)$, where PS is a finite set of *plan steps* $ps = (l, a)$ with l being a label unique in PS, $a \in A$ an action, and $\prec$ is a partial order on PS. The distinction between actions and plan steps is important because an action may occur multiple times within a partial plan, so unique labeling is required in order to differentiate different occurrences of the same action. As for actions, we use $prec(ps)$, $add(ps)$, and $del(ps)$ to refer to the precondition, add list, and delete list of a plan step.

Causal links are used to make the implicit causal relationships between actions in a plan explicit and to serve as a technical vehicle to document and verify the progress of turning a partial plan into an actual solution. More precisely, a causal link $cl = (ps, v, ps') \in PS \times V \times PS$ indicates that the precondition v of the *consumer* plan step ps' is *supported* by the *producer* plan step ps (i.e., it also implies $v \in add(ps) \cap prec(ps')$). The variable v is also said to be *protected* by the causal link. We call v "protected" by its causal link because the solution criteria ensure that v remains true for all state sequences between the link's producer and consumer. Every causal link raises a so-called *causal threat* in case there is another action in the current plan that could delete this protected condition – and all solutions need to be threat-free. Formally: Let a partial plan $P = (PS, \prec, CL)$ contain a causal link, $(ps, v, ps') \in CL$. A *causal threat* is the situation where a plan step ps'' with $v \in del(ps'')$ may be ordered between ps and ps', i.e., if $(\prec \cup \{(ps, ps''), (ps'', ps')\})^+$ (with X^+ denoting the transitive closure of X) is a strict partial order. The step ps'' is then called a *threatening* plan step. Threats can be resolved by ordering the threatening step before the link's producer, $ps'' \prec ps$ (called *promotion*) or by ordering it behind the consumer, $ps' \prec ps''$ (called *demotion*).

To ease definitions, we require that any causal link between two plan steps implies the respective ordering. So, without loss of generality, there is a corresponding ordering (ps, ps') in $\prec$ for every causal link (ps, v, ps') in CL. Due to the absence of states in partial plans we require, as usual in POCL planning, that each partial plan contains two artificial actions encoding the initial state and goal description. The former, called *init*, does not show a precondition and uses the initial state as add effect and, analogously, the latter, called *goal*, has no effects and uses the goal description as precondition. We demand that *init* is always the first action according to $\prec$ and *goal* always the last.

We can now extend the concept of t.o. plans to partially ordered ones.

Definition 2. A partial plan $P = (PS, \prec, CL)$ is called a *partial-order causal link (POCL) plan* (also *POCL solution*) to a planning task if and only if every precondition is supported by a causal link and there are no causal threats.

It is common knowledge (and trivial to prove) that every linearization of a POCL plan is a t.o. plan. It thus compactly represents an up to exponentially large set of totally ordered solutions.

Post-optimizing Actions

When investigating whether the removal of an action (i.e., plan step) from a partially ordered plan is feasible, we still need to discuss which further operations on the resulting partial plan are allowed afterwards. Simply removing a single action and *checking* whether the resulting partial plan is a solution is possible in polynomial time, but this will very often result in a negative answer to the question – although a solution exists without the removed action. Consider the following POCL plan as an example.

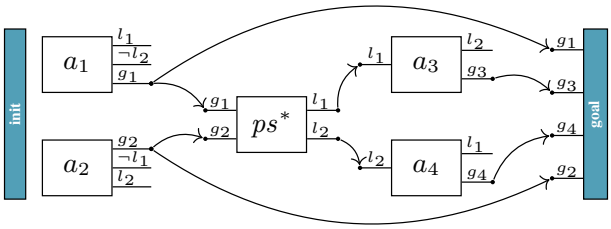

Figure 1: A POCL plan with seven plan steps (including *init* and *goal*), which are depicted as rectangles. The preconditions are represented on the left side of each action, the effects on the right side, where $\neg$ denotes a delete effect. The arrows indicate causal links. The plan does not possess more ordering constraints other than those implied by the links. The plan can be further improved with respect to the number of plan steps by removing ps^* and adding causal links and ordering constraints afterwards.

In the plan depicted here (Figure 1), only ps^* could possibly be removed, since all other actions are essential for the goal. Just removing it will not result in an improved solution for two reasons. First, it cannot be a solution *syntactically*, because there are preconditions that are not yet protected by a causal link (namely the l_1 and l_2 preconditions of a_3 and a_4, respectively). For protecting these conditions, there are several options available and not every one will lead to a solution. Second, also *semantically* it cannot be regarded a solution because not every linearization is an executable t.o. plan (e.g., a_1, a_2, a_3, a_4 is not). We can, however, obtain a solution by further refining the resulting partial plan by the "correct" ordering and link insertions (add the links (a_4, l_1, a_3) and (a_2, l_2, a_4) as well as the ordering $a_1 \prec a_2$).

We call this problem REMOVE & REPAIR and explore its computational complexity.

The Problem REMOVE & REPAIR

We start with the formal problem definition:

Definition 3. The decision problem REMOVE & REPAIR (R&R) is defined as follows: Let Π be a planning task, P a POCL plan that solves Π, and $ps^* \in P$ a plan step. Is there an ordering-refinement[1] $\tilde{P}$ of $P \setminus ps^*$[2] such that $\tilde{P}$ is a solution plan for Π?

Before we address the hardness of R&R, we mention the complexity of verifying whether a partial plan is a POCL solution, since we need it for the upcoming membership proofs. It is well-known that this problem is in ***P***, since checking the existence of the required causal links and the absence of causal threats is obviously a lower polynomial.

[1] Ordering refinement means that only ordering constraints and causal links may be added.

[2] $P \setminus ps$ denotes the partial plan $P' = (PS', \prec', CL')$ that results from P by removing ps, i.e., $PS' = PS \setminus \{ps\}$, $\prec' = \prec|_{PS'}$, and $CL' = CL|_{PS'}$, where $X|_Y$ denotes the set X restricted to elements of Y. We use the straight-forward extension of this notation to sets of plan steps PS', $P \setminus PS'$.

Lemma 1. *Deciding whether a partial plan* $P = (PS, \prec, CL)$ *is a POCL plan to a planning task is in* **P**.

To prove the ***NP**-completeness* of R&R we reduce another ***NP**-hard* problem to it. This problem is a new decision problem, which we defined to be close to our original problem. We call it CYCLE DISSOLVING PAIRS (CDP). We first formally define this problem, then prove its ***NP**-completeness*, and then reduce it to R&R.

Given a directed graph G and a partition of a subset of its vertex set such that each element has size two, the decision problem is: *Is it possible to make G acyclic by deleting at most one vertex of each partition element?* CDP mirrors the core hardness of R&R but is technically easier to handle.

Definition 4. The decision problem CYCLE DISSOLVING PAIRS (CDP) is defined as follows: Let $G = (V, E)$ be a directed graph and $\widehat{V} = \{V_1, V_2, \ldots, V_m\}$ a partition of a subset of V such that $|V_i| = 2$ for all $1 \leq i \leq m$. Is there a $U \subseteq V$ such that

- $U \subseteq \bigcup_{V_i \in \widehat{V}} V_i$,
- $|U \cap V_i| \leq 1$ for all $i = 1 \ldots m$ and
- $G \setminus U$ is acyclic?

We illustrate CDP with two examples given in Figure 2.

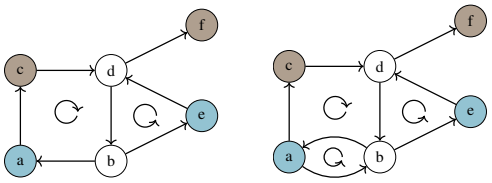

Figure 2: Examples of instances of CDP. In both cases the subset of vertices that one can remove are drawn in color, i.e. $\widehat{V} = \{\{c, f\}, \{a, e\}\}$.

For both examples given in Figure 2 the subset of vertices that we are allowed to remove is given by $\widehat{V} = \{\{c, f\}, \{a, e\}\}$. The graph on the left side is an example of a yes-instance. The vertices c and e can be removed so that the remaining graph is acyclic. Thus, $U = \{c, e\}$. The graph on the right side is an example of a no-instance. The cycle consisting of the vertices a and b can only be dissolved by removing a, because b is not a member of any of the subsets of $\widehat{V}$. But then e, which is the partner of a, must not be deleted. Thus, the answer to the decision problem is "no", as e is the sole vertex that could dissolve the cycle consisting of the vertices e, d and b, because d and b do not exist in $\widehat{V}$.

Theorem 1. CDP *is* **NP**-complete.

Proof. Membership: Guess a solution, verification can obviously be done in polynomial time.

Hardness: Proof by reduction from 3-SAT, which is ***NP**-complete* (Cook 1971; Garey and Johnson 1979).

Let an instance ϕ of 3-SAT be given, where ϕ consists of the clauses $C_1 \ldots C_n$, which depend on the Boolean variables $x_1, \ldots x_m$. The basic idea is that we associate every clause with a cycle of three vertices which correspond to the Boolean variables of that clause. The vertex that will be removed to dissolve the cycle indicates which variable will make the clause true. Therefore, we need further constructions. For every Boolean variable we construct two more vertices, which stand for the variable and its negation. Moreover, each of these pairs forms a partition element of $\widehat{V}$, which guarantees the allocation of every variable. Now, we want to connect those vertices to the above-mentioned 3-cycles encoding the clauses such that we can only delete vertices from the cycles in such a way that there is a unique choice whether a variable is true or false.

In greater detail, we construct an instance of CDP with $G = (V, E)$ and $\widehat{V}$ with $2 \cdot m + 2 \cdot 3 \cdot n$ vertices and $3 \cdot n + 2 \cdot 3 \cdot n$ directed edges. Figure 3 illustrates the subsequently described construction. For every variable x_j,

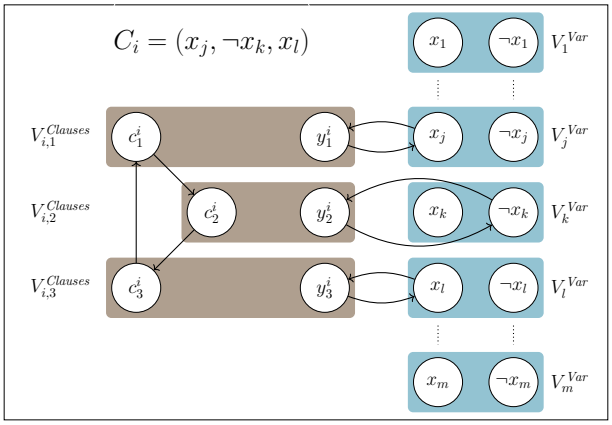

Figure 3: Part of a reduction from 3-SAT to CDP. The subgraph has been constructed given clause C_i top left.

$1 \leq j \leq m$, there are two vertices that we label x_j and $\neg x_j$, respectively. For every clause C_i, $1 \leq i \leq n$, we introduce six vertices, which we label c_1^i, c_2^i, c_3^i and y_1^i, y_2^i, y_3^i, respectively. The cs will form the cycles, whereas the ys are just auxiliary vertices, which form something like a bridge between the cycles and variables. So, there are cycles $\{(c_1^i, c_2^i), (c_2^i, c_3^i), (c_3^i, c_1^i)\} \subseteq E$ for all $i = 1 \ldots n$ associated with the clauses. To establish a connection to the literals we add more arcs in the following way. Let x_{i_k}, $i_k \in \{1 \ldots m\}$ be the k-th literal in clause i. Then we add (y_k^i, x_{i_k}) and (x_{i_k}, y_k^i) or $(y_k^i, \neg x_{i_k})$ and $(\neg x_{i_k}, y_k^i)$ if x_{i_k} is negated, for all $i = 1 \ldots n$ and $k = 1 \ldots 3$.

Finally, we can define a partition $\widehat{V} = \{V_1^{Var}, \ldots, V_m^{Var}, V_{1,1}^{Clauses}, \ldots, V_{n,3}^{Clauses}\}$ of a subset of V, where $V_j^{Var} = \{x_j, \neg x_j\}$ for $1 \leq j \leq m$ and $V_{i,k}^{Clauses} = \{c_k^i, y_k^i\}$ for $1 \leq i \leq n$ and $1 \leq k \leq 3$. V_j^{Var} will determine the assignment of the variables x_j for all $j = 1 \ldots m$ as mentioned before, whereas $V_{i,k}^{Clauses}$ ensures that we can not delete both a vertex associated with a variable in one cycle and another vertex associated with its negation in a different cycle. This yields a graph G. This construction is illustrated in Figure 4.

Now we claim that there is a subset U as described in the problem definition such that $G \setminus U$ is acyclic precisely when

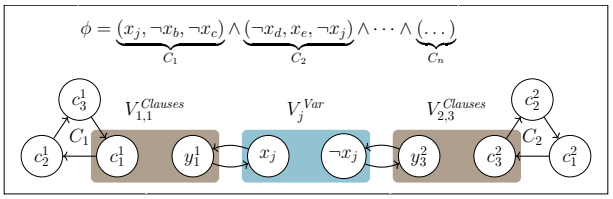

Figure 4: Illustrates the main mechanism of the reduction. The subgraph has been constructed from the clauses above it. If we delete the vertex x_j, the cycle containing x_j and y_1^1 is dissolved. Therefore c_1^1 can be removed, which corresponds to x_j in clause C_1. In order to dissolve the cycle containing $\neg x_j$ and y_3^2, only y_3^2 can be removed. Then, c_3^2 must remain, which corresponds to $\neg x_j$ in clause C_2.

the formula ϕ can be satisfied.

$\Rightarrow$ Assume first that there exists such a subset U. Claim: For all $j = 1 \dots m$ the vertex in $U \cap V_j^{Var}$ induces an assignment of true and false to the Boolean variable x_j so that ϕ evaluates to true, namely $x_j = true$ if $x_j \in U \cap V_j^{Var}$, otherwise $x_j = false$. Figure 4 exemplarily illustrates the following explanations. Without loss of generality, assume x_j has been deleted from the graph, then the cycles including the adjacent vertices y_k^i, $1 \leq i \leq n$, $1 \leq k \leq 3$ disappear as well. Therefore, the partner of y_k^i in $V_{i,k}^{Clauses}$, which is c_k^i, can possibly be put in U. Thus, we can assume that the variable x_j evaluates C_i to true, since the $V^{Clauses}$ are defined accordingly to the literals in the clauses. On the other hand, as the vertex $\neg x_j$ remains in the graph, the adjacent vertices $y_{k'}^{i'}$, $1 \leq i' \leq n$, $1 \leq k' \leq 3$, must be removed in order to dissolve the cycles containing them and $\neg x_j$. But then the $c_{k'}^{i'}$ corresponding to $\neg x_j$ in clause i' must not be removed, which is what we wanted. As $G \setminus U$ is acyclic, in particular there is no more cycle of the form (c_1^i, c_2^i), (c_2^i, c_3^i), (c_3^i, c_1^i) left, the assignment evaluates to true. As either x_j or $\neg x_j$ can be deleted because of the construction of V_j^{Var}, there are no conflicts concerning the assignment of a variable.

$\Leftarrow$ For the other direction, assume that there is an assignment to $x_1, \dots x_m$ so that ϕ evaluates to true. If $x_j = true$, we put the vertex x_j into U and for every adjacent vertex y_k^i, $1 \leq i \leq n$, $1 \leq k \leq 3$, we also put c_k^i in U. Otherwise, if $x_j = false$, we put the vertex $\neg x_j$ into U and also for every adjacent vertex $y_{k'}^{i'}$, $1 \leq i' \leq n$, $1 \leq k' \leq 3$, we put $c_{k'}^{i'}$ into U. We do this for all variables x_j, $j = 1 \dots m$. As the assignment evaluates to true, every clause evaluates to true and therefore there is no cycle including vertices labeled $c_{k''}^{i''}$, $1 \leq i'' \leq n$, $1 \leq k'' \leq 3$, left. To dissolve the remaining cycles that necessarily contain the vertices $y_{k''}^{i''}$ and $x_{j''}$ or $y_{k''}^{i''}$ and $\neg x_{j''}$, $1 \leq j'' \leq m$, we can put those $y_{k''}^{i''}$s into U. By construction, $G \setminus U$ is acyclic and U satisfies the properties stated in the problem definition. □

Before we formally prove the hardness of R&R via CDP we first show how the two problems relate to each other.

Consider Figure 5, where a POCL plan for a simple planning task is given. Plan step ps^* establishes the variables l_1, l_2, and l_3 but ps^* will be removed. We want to find an ordering-refinement of the resulting partial plan that is again a solution. First, take a look at the plan steps a_i, $i = 1 \dots 3$, which are ordered before ps^*. They form something like a cycle in their effects because in whatever order we put them, there is always a variable not true at the end. But there are also actions that are ordered after ps^*, which allow two options of ordering. If we order $b \prec c$ and add the causal link (b, l_3, c) all preconditions concerning l_3 are satisfied, analogously the same holds for l_1 if we order $c \prec b$ instead. But we have to make a choice, so assume we add (b, l_3, c) in this case. Then, we do not need to take l_3 into account regarding the ordering of the a_is as well. Thus, adding (a_2, l_2, g) and (a_1, l_1, b) and $a_3 \prec a_1 \prec a_2$ results in a solution plan. Therefore, the pair of plan steps b and c perform like a partition element in the CDP instance in the sense that the choice of ordering determines which state variable will be removed from consideration regarding the cycles of the effects of the a_is. Note that if there is no more cycle left in the effects of the a_is we can order them such that all needed preconditions are satisfied. We now generalize this idea.

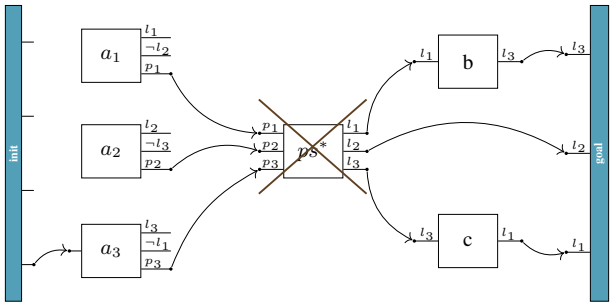

Figure 5: Partial instance of R&R that demonstrates difficult structural properties on a small scale.

Theorem 2. REMOVE & REPAIR *is* **NP**-complete.

Proof. Membership: Guess an ordering-refinement $\tilde{P}$ of $P \setminus ps^*$ such that $\tilde{P}$ is a valid POCL plan for Π. Validation can be done in polynomial time (Lemma 1).

Hardness: Karp-reduction[3] from CDP to R&R. Suppose we are given a directed graph $G = (V, E)$, where $V = \{v_1 \dots v_n\}$, and $\widehat{V} = \{V_1, V_2, \dots, V_m\}$ a partition of a subset of V such that $|V_j| = 2$ for all $j = 1 \dots m$. Let us construct a corresponding planning task $\Pi = (V, A, s_I, g)$ and a POCL plan P for it. Let $V = \{l_1 \dots l_n\}$. For every vertex $v_i \in V$ we introduce one action $a_i \in A$ that has no preconditions and $add(a_i) = \{l_i\}$. For every outgoing edge $(v_i, v_{i'})$ of v_i there is a negative effect $l_{i'}$, so $del(a_i) = \bigcup_{v_{i'} \in N_G^+(v_i)} \{l_{i'}\}$[4]. Moreover, for every $V_j = \{v_i, v_{i'}\}$, $j = 1 \dots m$, we introduce two more actions $b_j \in A$ and $c_j \in A$, where $prec(b_j) =$

[3] Karp-reductions are polynomial-time many-one reductions, named after Richard Karp.

[4] $N_G^+(v)$ denotes the set of all successors of a vertex v.

$\{l_i\}$, $prec(c_j) = \{l_{i'}\}$, $add(b_j) = \{l_{i'}\}$, $add(c_j) = \{l_i\}$ and $del(b_j) = del(c_j) = \emptyset$. There is one more action $ps^* \in A$ with $prec(ps^*) = del(ps^*) = \emptyset$ and $add(ps^*) = \{l_1 \dots l_n\}$. As initial state we use $s_I = \emptyset$ and as goal $g = \{l_1 \dots l_n\}$. For convenience, we label the plan steps according to their action names, as each action will appear just once in our plan. So, define $P = (PS, \prec, CL)$, where $PS = \bigcup_{i=1\dots n}\{a_i\} \cup \bigcup_{j=1\dots m}(\{b_j\} \cup \{c_j\}) \cup \{ps^*\}$ and $\prec' = \{(a_i, ps^*) \mid i = 1 \dots n\} \cup \{(ps^*, b_j), (ps^*, c_j) \mid j = 1 \dots m\}$ and $\prec = \prec'^+$. Furthermore, $CL = \{(ps^*, l_i, g) \mid i = 1 \dots n\} \cup \{(ps^*, l_{j'}, b_j), (ps^*, l_{j''}, c_j) \mid l_{j'} \in prec(b_j), l_{j''} \in prec(c_j), j = 1 \dots m\}$.

Then P is a POCL plan for Π, because all preconditions are protected by ps^* and the plan steps a_i, $1 \leq i \leq n$, which are the sole plan steps which can threaten the causal links originating in ps^*, are ordered before ps^*. Hence, there are no causal threats. This construction can clearly be done in polynomial-time.

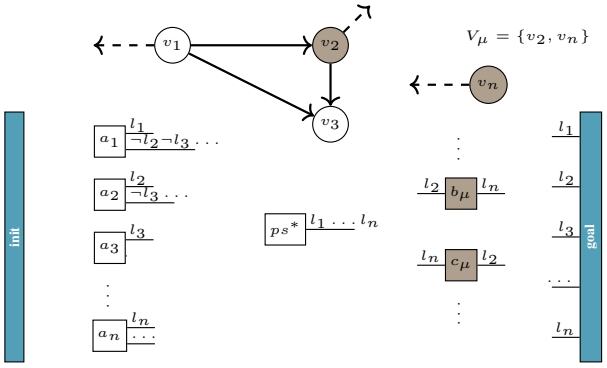

Figure 6: Reduction of a partial instance of CDP at the top to a partial instance of R&R. To maintain clarity in the figure, causal links and ordering contains are not drawn in but implied by the arrangement of the actions from left to right, i.e. $a_1 \dots a_n$ are not ordered w.r.t. each other, but only to ps^*. b_μ and c_μ analogously.

Now we need to show that our instance of CDP is a yes-instance if and only if there is an ordering-refinement $\tilde{P}$ of $P \setminus ps^*$ such that $\tilde{P}$ is a valid POCL plan for Π.

$\Rightarrow$ Assume there is a subset $U \subseteq V$ that satisfies the properties given in the problem definition and $\tilde{G} = (\tilde{V}, \tilde{E}) = G \setminus U$ is acyclic. Let $\tilde{P} = P \setminus ps^*$ and $R = \{r \mid v_r \in U\} \subseteq \{1 \dots n\}$ be the indices of vertices in U. For every $r \in R$ and $1 \leq j \leq m$ such that $add(b_j) = \{l_r\}$ we add the causal link (b_j, l_r, c_j) and implied ordering constraints $b_j \prec c_j$ to $\tilde{P}$. Analogously, for every $r \in R$ and $1 \leq j' \leq m$ such that $add(c_{j'}) = \{l_r\}$ we add the causal link $(c_{j'}, l_r, b_{j'})$ and the implied ordering constraints $c_{j'} \prec b_{j'}$. So, the selection of vertices that were removed from the graph determine the ordering of the plan steps labeled b and c. This implies that the deletion of an vertex v_r guarantees that preconditions concerning the corresponding variable l_r are supported in the whole plan by one of the b_js or c_js. Thus, the plan steps a_r for all $r \in R$ are then not needed and can be ordered in the front before all a_s, $s \in S = \{1 \dots n\} \setminus R$, so that their negative effects do not threaten any causal link. Hence, we extend $\tilde{\prec}$ by $\{(a_r, a_s) \mid r \in R, s \in S\}$. Furthermore, we add the causal links $\{(a_s, l_s, g) \mid s \in S\}$ and order the plan steps a_s, $s \in S$, according to the direction of arcs incident with the corresponding vertices in $\tilde{G}$. As a consequence, their positive effects were not deleted by plan steps that are executed after them such that there do not occur causal threats. More formally, for every edge $(v_i, v_{i'}) \in \tilde{E}$ we add $a_i \tilde{\prec} a_{i'}$.

Let us verify the resulting POCL plan. The preconditions (including the goal state) concerning the variables $\{l_r \mid r \in R\}$ are supported by the b_js and c_js as stated before, which are in particular not threatened. We claim that the remaining preconditions are protected by causal links originating in one of the a_ss, which are also not threatened. By construction, an action a_i has a negative effect $\neg l_k$ if and only if there is an edge $(v_i, v_k) \in E$. Therefore, for every causal link protecting a precondition l_s, $s \in S$, that is supported by plan step a_s, there does not exist another plan step a_t, $1 \leq t \leq n$ with $del(a_t) = l_s$, which may be ordered after a_s, because $\tilde{G}$ is acyclic and we ordered the a_ss according to $\tilde{G}$. Thus, $\tilde{P}$ is a POCL plan that solves Π.

$\Leftarrow$ For the other direction consider a POCL plan $\tilde{P} = (\tilde{PS}, \tilde{\prec}, CL)$ that is an ordering-refinement of $P \setminus ps^*$. Let us construct a subset $U \subseteq \widehat{V}$ such that $\tilde{G} = G \setminus U$ is acyclic. For every ordering of the form $(b_j, c_j) \in \tilde{\prec}$ remove the vertex v_i with $add(b_j) = l_i$ and analogously for every ordering of the form $(c_j, b_j) \in \tilde{\prec}$ remove the vertex $v_{i'}$, where $add(c_j) = l_{i'}$. This is permitted since we constructed the actions b_j and c_j according to the elements of $\widehat{V}$. We claim that the remaining graph $\tilde{G}$ is acyclic.

Define again $R = \{r \mid v_r \in U\}$, $S = \{1 \dots n\} \setminus R$ and $PS_S = \{a_s \mid s \in S\}$. So, the plan steps in PS_S correspond to the remaining vertices in $\tilde{G}$. The a_ss in $\tilde{P}$ must be ordered in a way such that they establish the variables $\{l_s \mid v_s \in \tilde{G}\}$ because they are not established by the b_js or c_js because of the orderings of the b_js and c_js. Consider $a_s \in PS_S$, that is ordered at the beginning, i.e. there does not exist $a_{s'} \in PS_S$ such that $a_{s'} \prec a_s$. a_s must support the precondition l_s of the b_js, c_js or g. Therefore, there must not be a plan step $a_t \in PS_S$ with $l_s \in del(a_t)$, since it would threaten the causal links originating in a_s because of the choice of a_s. This implies that v_s does not have any ingoing edge in $\tilde{G}$. Thus, there is no cycle containing v_s. We can repeat these arguments with respect to $PS_S \setminus \{a_s\}$ and $\tilde{G} \setminus v_s$, respectively. Then we find another vertex $v_{s''}$ that is not contained in any cycle. Inductively, it follows that $G \setminus U$ is acyclic. □

Note that the arguments in the hardness proof can easily be adapted to standard partially ordered plans, where executability is defined without causal links by demanding that all linearizations are executable. Membership is trivial. We can thus also consider the variant of R&R with standard partially ordered plans to be ***NP**-complete*.

So, we have seen that R&R is a computationally hard problem, unfortunately. Moreover, R&R modified by allowing not only adding ordering constraints but deleting them as

well is also ***NP**-hard* since it would not change the previous proof. There is no benefit in ordering one of the a_is after the b_js or c_js as the a_is do not have preconditions and the b_js and c_js have empty delete lists. So only adding ordering constraints is a special case and the possibility of deleting ordering constraints could make the problem harder. Membership is trivial due to Lemma 1.

Corollary 1. *Let Π be a planning task, P a POCL plan that solves Π, and $ps^* \in P$ a plan step. The problem of deciding whether ps^* can be removed such that the remaining plan can be repaired by changing causal links and ordering constraints arbitrarily is* **NP**-complete.

In the definition of R&R we are *given* a predetermined plan step, which we try to remove. But we can also ask ourselves whether there *exists* any plan step that we can remove such that there exists an ordering-refinement of the remaining plan that is a solution.

Definition 5. Let P be a POCL plan to some planning task Π. The problem of deciding whether there exists a plan step ps^* such that there exists an ordering-refinement of $P \setminus ps^*$ that solves Π is called $\exists PS$ - R&R.

Theorem 3. $\exists PS$ - R&R *is* **NP**-complete.

Proof. Membership: Guess a plan step ps^* and an appropriate ordering-refinement $\tilde{P}$ of $P \setminus ps^*$. We can verify in polynomial time whether $\tilde{P}$ is a POCL plan to Π according to Lemma 1.

Hardness can be shown by reducing R&R to it. So let $P = (PS, \prec, CL)$ be a POCL plan to some planning task Π and let ps^* be given. Moreover, let $\{ps_1 \dots ps_n\} = PS \setminus ps^*$ be the remaining plan steps. Modify this instance in the following way: For all $i = 1 \dots n$ add an additional state variable g_i that has not been in the domain before to $add(ps_i)$ and the goal state g. Moreover, add the appropriate causal links (ps_i, g_i, g) for all $i = 1 \dots n$. Then clearly, ps^* is the sole plan step that can possibly be removed from the resulting plan and therefore finding any plan step that can be removed solves the problem instance of R&R. □

By combining our main result (Thm. 2) with the last one (Thm. 3) we can obtain their generalization to k actions.

Corollary 2. *Let Π be a planning task and P a POCL plan that solves Π. The following two problems are* **NP**-complete*:*

- *Given k plan steps, $PS^* = \{ps_1^*, \dots, ps_k^*\} \subseteq P$, is there an ordering-refinement $\tilde{P}$ of $P \setminus PS^*$ such that $\tilde{P}$ is a solution plan for Π?*
- *Are there k plan steps $PS^* = \{ps_1^*, \dots, ps_k^*\} \subseteq P$, such that there is an ordering-refinement $\tilde{P}$ of $P \setminus PS^*$ with $\tilde{P}$ being a solution plan for Π?*

Note that the second part of the previous corollary already follows from the fact that deciding whether k actions can be removed from a *t.o. plan* is ***NP**-complete* (Nakhost and Müller 2010).

Parameterized Complexity

We have seen that R&R is ***NP**-complete*. So there does not exist a polynomial time algorithm that solves the problem for an arbitrary instance unless $\boldsymbol{P} = \boldsymbol{NP}$. Nevertheless, the proof indicates that the hardness results from certain structural properties in the plan. In practice, they may not appear extensively. Therefore, we look at the problem again under the light of fixing them as a parameter in the input. This leads to the theory of *parameterized complexity*, which has been initiated by Downey and Fellows (1999), but we follow the definition of Aghighi and Bäckström (2017).

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}_0$, where Σ is a finite alphabet and $\mathbb{N}_0$ is the set of non-negative integers[5]. An *instance* of the problem is a pair $(\mathbb{I}, k)$, where $\mathbb{I}$ is a string over Σ^* and $k \in \mathbb{N}_0$ is the *parameter*. A parameterized problem is *fixed-parameter tractable (fpt)* if there exists an algorithm that solves every instance $(\mathbb{I}, k)$ in time $f(k) \cdot |\mathbb{I}|^c$ where f is a computable function and c is a constant. FPT is the class of all fixed-parameter tractable decision problems.

Let us come back to R&R. If we delete a plan step and its respective causal links, there most likely remain plan steps with unsupported (also called open) preconditions. The parameter of the R&R instance that we fix is the number of plan steps satisfying all of the following three properties:

- They are ordered (not necessarily directly) behind the removed plan step,
- can be ordered before plan steps with unsupported preconditions (so they are not already ordered after them),
- and can support any of these open preconditions.

We will call the steps satisfying these properties $Atweens$.

More precisely, let P be a POCL plan, $ps^* \in P$ the plan step to be removed and $cl = (ps^*, l, ps) \in CL$, $l \in V$, $ps \in PS$, a causal link originating in the plan step to be removed. Then define $A_{cl} = \{a \in PS \mid (ps^*, a) \in \prec \wedge (ps, a) \notin \prec \wedge\, l \in add(a)\}$, $Atweens = \bigcup_{cl=(ps^*,l,ps)\in CL} A_{cl}$ and $\#_{Atweens} = |Atweens|$. The notation $\#_{Atweens}$ - R&R refers to the variant of R&R parameterized with the parameter $\#_{Atweens}$.

In order to show that $\#_{Atweens}$ - R&R is fixed-parameter tractable we present Algorithm 1, which runs in time $\mathcal{O}(\#_{Atweens}! \cdot |PS|^3 + |Prec| \cdot |PS|^2)$, where $|Prec| = \sum_{ps \in PS} |prec(ps)|$. The main idea is: After fixing a linearization of the $Atweens$ we can compute in polynomial time whether there exists an ordering-refinement that solves the planning task or not. In the worst case we need to try all linearizations, of which there are at most $\#_{Atweens}!$.

Theorem 4. $\#_{Atweens}$ - R&R *is in FPT.*

Proof. Let Π be a planning task, P a POCL plan to Π, $ps^* \in P$ a plan step, and $\#_{Atweens}$ be given. We ask for an ordering-refinement $\tilde{P}$ of $P \setminus ps^*$ such that $\tilde{P}$ is a solution plan for Π, i.e. we try to choose alternative supporters for causal links that were removed with ps^*. Therefore, we run

[5]Given an alphabet Σ, the set of all strings of length n over the alphabet Σ is indicated by Σ^n. The set $\bigcup_{i\in\mathbb{N}} \Sigma^i$ of all finite strings is indicated by the Kleene star operator as Σ^*, and is also called the Kleene closure of Σ.

this input on Algorithm 1. The procedure and the plan steps are split into three parts, which are sketched in Figure 7.

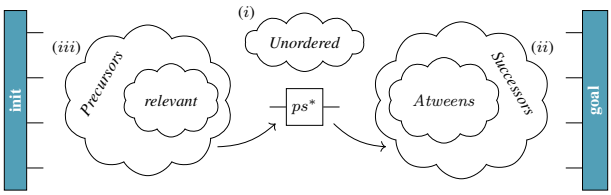

Figure 7: A high-level sketch of Algorithm 1.

Plan steps that are unordered with respect to ps^* (i) have assessable properties. Thus, we first try to find producers for open preconditions among these, because we can not make wrong decisions (as they did not threaten any of the causal links originating in ps^*). After that we fix an arbitrary total ordering of the $Atweens$ (ii) and check whether we can achieve a POCL plan by bringing the plan steps that were ordered before ps^* (iii) in an appropriate order. If this is not possible, we try the same with a different total order of the $Atweens$. Note that we are always done once we found producers for all open preconditions and there are no causal threats, so in the following we assume that after each step there are still open preconditions left.

To start with, we analyze plan steps that are unordered with respect to ps^*, so let $Unordered = \{ps \in PS \mid (ps, ps^*) \notin \prec \wedge (ps^*, ps) \notin \prec\}$. This property implies that they will not threaten any future causal link supporting one of the open preconditions since they did not threaten the links originating in ps^*. Therefore, without loosing any solution we could order all of them at the position where ps^* has been and add respective causal links. Then, possible threatening plan steps are already ordered before the $Unordered$ or after the consumers since there were no threats concerning causal links originating in ps^*. This procedure would in most cases insert more ordering constraints than necessary. To counter this, we just pick relevant producers among $Unordered$ as described in the he first forall-loop in line 3. So, consider a plan step c whose precondition l is not supported anymore. If there is a plan step $a \in Unordered$ with $l \in add(a)$, we can insert the causal link (a, l, c). Possible threatening plans steps to this link must have been ordered before ps^*, thus we order them also before a. We do this for every unsupported precondition, which takes at most $|Prec| \cdot |PS|^2$ computation steps, where $|Prec| = \sum_{ps \in PS} |prec(ps)|$.

Now, let $Atweens$ be as defined with respect to ps^*. Select a total order of these steps consistent with $\prec$ (line 10) and insert possible threat-free causal links where the plan steps out of $Atweens$ function as producer (line 11). Because of the total order we do not insert further ordering constraints to resolve causal threats and all other plan steps are already ordered before the $Atweens$ or are harmless.

After that we can try to find producers for the remaining open preconditions among the relevant plan steps that were ordered before ps^*, which are called $RelevantPrecursors$ ($RelPrecurs$) (line 14). By *relevant* we mean plan steps that have any of the open preconditions as a positive effect. Assume there does *not* exist a relevant precursor a such that neither a nor any of the plan steps ordered between a and ps^* deletes any of the remaining open preconditions. In this case there does not exist an ordering-refinement of this partial plan solving Π because no matter which total order of the $Precursors$ we pick, there is always a plan step deleting one of the needed literals ordered after all $RelPrecurs$ such that this open precondition can not be supported. Then, we need to try a different linearization of the $Atweens$. So, assume we can select a relevant precursor a such that neither a nor any of the plan steps ordered before ps^* but after a deletes any of the remaining open preconditions and insert the respective causal links. Threatening plan steps can then be ordered before a. Afterwards, we pick the next relevant precursor with the same properties and inductively repeat this procedure with the remaining relevant precursors until either there are no open preconditions left (and return the resulting POCL plan) or such a relevant precursor does not exist and there are still open preconditions. In the latter case there does not exist a further ordering-refinement that solves Π as argued before. Thus, we must go back to line 10 and try a different total order of the $Atweens$. If we fail after testing all possible total orders of the $Atweens$, $\tilde{P}$ does not exist.

We can assume that the single lines in the algorithm can be computed efficiently if the plan is encoded in an intelligent way. The while loop runs at most $|RelevantPrecursors|$ times for every linearization of the $Atweens$, which are at most $\#_{Atweens}!$ many. Per iteration at most $|RelevantPrecursors| \cdot |Precursors|$ steps of calculation are needed. Thus, the algorithm runs roughly in time $\mathcal{O}(\#_{Atweens}! \cdot |PS|^3 + |Prec| \cdot |PS|^2)$. □

If $\#_{Atweens} \ll |PS|$, Algorithm 1 is at least better than the brute force method to try all linearizations of the plan.

We have seen that the exponential runtime of the Algorithm 1 is due to the exponentially many linearizations of the $Atweens$. If the set of $Atweens$ is empty, Algorithm 1 runs in polynomial time. Therefore, we can impose the following restriction to the plan step to be removed such that R&R can be solved in polynomial time.

Definition 6. Let $P = (PS, \prec, CL)$ be a POCL plan. We call a plan step $ps \in PS$ *last establisher* if for all $l \in \{l \in add(ps) \mid \exists\, (ps, l, ps') \in CL, ps' \in PS\}$ there does not exist a plan step $\hat{ps}$ such that $ps \prec \hat{ps}$ and $l \in add(\hat{ps})$.

Corollary 3. R&R *can be decided in polynomial time if* ps^* *is a last establisher.*

Discussion

Our main result, REMOVE & REPAIR (R&R) being ***NP-complete***, is also related to the ***NP-completeness*** of deciding whether there *exists* an applicable action sequence given a partial plan without causal links, which was shown independently by Nebel and Bäckström (1994) (cf. Thm. 15) for event systems and later by Erol, Hendler, and Nau (1996) (cf. Thm. 8) in the context of hierarchical planning[6]. We

[6]This was shown for the special case where there are no abstract actions, which then coincides with the plan linearization problem.

Algorithm 1: Solves an instance of $\#_{Atweens}$ - R&R in $\mathcal{O}(\#_{Atweens}! \cdot |PS|^3 + |Prec| \cdot |PS|^2)$

Input: POCL plan $P = (PS, \prec, CL)$ to planning problem Π, $ps^* \in P$ and $\#_{Atweens}$
Output: Answer to R&R, i.e. a POCL plan P' if one exists or fail otherwise.

$P' = (PS_{new}, \prec_{new}, CL_{new}) \leftarrow P \setminus ps^*$;
$open \leftarrow \{cl \in CL \mid cl \text{ originates in } ps^*\}$;
forall $(ps^*, l, c) \in open$ **do**
  if $\exists\, a \in PS_{new}$ s.t. $l \in add(a) \wedge (a, ps^*) \notin \prec \wedge (ps^*, a) \notin \prec$ **then**
    $CL_{new} \leftarrow CL_{new} \cup \{(a, l, c)\}$;
    $\prec_{new} \leftarrow \prec_{new} \cup \{(t, a) \mid t \in PS_{new} \wedge l \in del(t) \wedge (c, t) \notin \prec_{new}\}^+$;
    $open \leftarrow open \setminus \{(ps^*, l, c)\}$;
$Atweens \leftarrow \bigcup_{cl \in open} A_{cl}$;
forall linearizations of plan steps in $Atweens$ consistent with $\prec'$ denoted $\overline{\prec}|_{Atweens}$ **do**
  $\prec_{temp} \leftarrow \prec_{new} \cup \overline{\prec}|_{Atweens}$;
  $CL_{temp} \leftarrow \{(b, l, c) \mid b \in Atweens \wedge (ps^*, l, c) \in open \wedge l \in add(b) \wedge (b, c) \in \prec_{temp}\}$;
  $open_{temp} \leftarrow \{(ps^*, l, c) \in open \mid (b, l, c) \notin CL_{temp}\}$;
  $Precursors \leftarrow \{a \in PS_{new} \mid (a, ps^*) \in \prec\}$;
  $RelPrecurs \leftarrow \{a \in PS_{new} \mid (a, ps^*) \in \prec \wedge\, add(a) \cap \{l \mid (ps^*, l, c) \in open_{temp}\} \neq \emptyset\}$;
  while $RelPrecurs \neq \emptyset$ and $open_{temp} \neq \emptyset$ **do**
    if $\exists\, a \in RelPrecurs$ s.t. $(del(a) \cap \{l \mid (ps, l, c) \in open_{temp}\} = \emptyset \wedge a \not\prec b \,\forall\, b \in Precursors$ with $del(b) \cap \{l \mid (ps, l, c) \in open_{temp}\} \neq \emptyset)$ **then**
      $CL_a \leftarrow \{(a, l, c) \mid l \in add(a) \wedge (ps^*, l, c) \in open_{temp}\}$;
      $CL_{temp} \leftarrow CL_{temp} \cup CL_a$;
      $\prec_{temp} \leftarrow \prec_{temp} \cup \{(t, a) \mid t \in Precursors$ threatens any of $CL_a\}$;
      $open_{temp} \leftarrow \{(ps^*, l, c) \in open_{temp} \mid (a, l, c) \notin CL_a\}$;
      $RelPrecurs \leftarrow RelPrecurs \setminus \{a\}$;
    else
      break;
  if $open_{temp} = \emptyset$ **then**
    return $P' = (PS_{new}, \prec_{temp}, CL_{new} \cup CL_{temp})$
return *fail*

regard this relationship interesting as is shows that finding an executable action sequence (without any prior knowledge about the input plan) is equally hard as "repairing" a plan in which all sequences have already been executable.

We would also like to highlight the relationship of our result to the ones by Fink and Yang (1992) and Nakhost and Müller (2010). Due to their work, it's known that it is ***NP-complete*** to decide whether there are k actions that can be removed from a t.o. plan so that the resulting plan still solves the problem. Note that it is trivially tractable to decide whether one or more *given* actions can be removed, because one can simply verify the executability of the remaining plan in polynomial time. Interestingly, for partially ordered plans the situation is more complicated. While we get exactly the same results if we only allow the removal of actions from a standard partially ordered plan (because verification is also a tractable problem for such partial plans without causal links (Nebel and Bäckström 1994, Thm. 12)[7]) or from a POCL plan, we showed that it already becomes ***NP-hard*** for a single *given* action if we allow "repairing" the resulting plan by ordering insertions as discussed above.

One of the implications of this finding is that it rules out a polynomial greedy-optimization algorithm that would have been possible if the R&R problem were decidable in polynomial time for $k = 1$ actions. If that problem were in ***P*** (as it is the case for t.o. plans), we could select any redundant action and repair the resulting partial plan. We could then continue until no more (single) action can be removed, thereby obtaining a greedy optimization algorithm. This algorithm, of course, would only find local minima since it could not identify all cases where *groups of actions* could still be removed. Due to our result, however, such a tractable greedy optimization algorithm cannot exist (unless $\boldsymbol{P} = \boldsymbol{NP}$), which raises the question whether further efficient approximation algorithms exist or whether it's equally efficient to aim for optimal optimizations in the first place. As mentioned in the related work section, several efficient approaches exist already despite the hardness of the problem (Siddiqui and Haslum 2015; Muise, Beck, and McIlraith 2016; Say, Cire, and Beck 2016).

Conclusion

We investigated the computational complexity of detecting unnecessary plan steps in partially ordered plans that can be deleted such that the resulting plan can be repaired to a solution by adding causal links and ordering constraints. Therefore, we could prove that this problem, called REMOVE & REPAIR, is ***NP-complete*** – even if there is just a single *given* action that we want to delete from the given solution. This is an interesting result since the similar problem for a *totally ordered* input plan is only in ***P***. Moreover, we presented a fixed-parameter tractable algorithm, which exploits the structural properties that are responsible for the hardness. Future work can be done by extending the studies Bäckström (1998) has done concerning the possibilities of post-optimizing the *makespan* of a partially ordered plan, i.e., the execution time of a partially ordered plan when taking parallelism into account.

Acknowledgements

We would like to thank the reviewers for their thorough reviews and useful comments that helped improving the paper. This work was done within the technology transfer project

[7] Nebel and Bäckström (1994) showed this result for event systems, but they essentially translate to standard partially ordered plans. One only has to check whether for each precondition there exists a predecessor action producing it, such that there is no other action that can delete it afterwards.

"Do it yourself, but not alone: *Companion*-Technology for DIY support" of the Transregional Collaborative Research Centre CRC/TRR 62 "*Companion*-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG). The industrial project partner is the Corporate Research Sector of the Robert Bosch GmbH.

References

Aghighi, M., and Bäckström, C. 2017. Plan reordering and parallel execution – A parameterized complexity view. In *Proc. of AAAI'17*, 3540–3546. AAAI Press.

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J. C.-j.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B. G.; Dias, W. C.; and Maldague, P. 2004. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Behnke, G.; Höller, D.; Bercher, P.; and Biundo, S. 2016. Change the plan – how hard can that be? In *Proc. of ICAPS'16*, 38–46. AAAI Press.

Bercher, P.; Geier, T.; Richter, F.; and Biundo, S. 2013. On delete relaxation in partial-order causal-link planning. In *Proc. of the 25th Int. Conf. on Tools with Artificial Intelligence (ICTAI'13)*, 674–681. IEEE Computer Society.

Bercher, P.; Biundo, S.; Geier, T.; Hörnle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *Proc. of ICAPS'14*, 386–394. AAAI Press.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a name? On implications of preconditions and effects of compound HTN planning tasks. In *Proc. of ECAI'16*, 225–233. IOS Press.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid planning heuristics based on task decomposition graphs. In *Proc. of SoCS'14*, 35–43. AAAI Press.

Bit-Monnot, A. 2016. *Temporal and Hierarchical Models for Planning and Acting in Robotics*. Dissertation, Université de Toulouse.

Bresina, J. L., and Morris, P. H. 2007. Mixed-initiative planning in space mission operations. *AI magazine* 28(2):75–88.

Bäckström, C. 1998. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research (JAIR)* 9:99–138.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proc. of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71)*, 151–158. ACM Press.

Downey, R. G., and Fellows, M. R. 1999. *Parameterized Complexity*. Springer.

Dvořák, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and acting with temporal and hierarchical decomposition models. In *Proc. of the 26th Int. Conf. on Tools with Artificial Intelligence (ICTAI'14)*, 115–121. IEEE.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence (AMAI)* 18(1):69–93.

Fink, E., and Yang, Q. 1992. Formalizing plan justifications. In *Proc. of the 9th Conf. of the Canadian Society for Computational Studies of Intelligence*, 9–14.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proc. of AAAI'08*, 944–949. AAAI Press.

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

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proc. of ICAPS'12*, 92–100. AAAI Press.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proc. of ICAPS'11*, 154–161. AAAI Press.

McAllester, D. A., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. of AAAI'91*, 634–639. AAAI Press.

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

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proc. of ICAPS'10*, 137–144. AAAI Press.

Nebel, B., and Bäckström, C. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence* 66(1):125–160.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. of KR'92*, 103–114. Morgan Kaufmann.

Say, B.; Cire, A. A.; and Beck, J. C. 2016. Mathematical programming models for optimizing partial-order plan flexibility. In *Proc. of ECAI'16*, 1044–1052. IOS Press.

Schattenberg, B. 2009. *Hybrid Planning & Scheduling*. Dissertation, University of Ulm, Germany.

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *Proc. of ICAPS'12*, 225–233. AAAI Press.

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

Tan, X., and Gruninger, M. 2014. The complexity of partial-order plan viability problems. In *Proc. of ICAPS'14*, 307–313. AAAI Press.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170:298–335.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research (JAIR)* 20:405–430.