Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)
Survey Track

A Survey on Plan Optimization

Pascal Bercher[1], **Patrik Haslum**[1] and **Christian Muise**[2]
[1]School of Computing, The Australian National University
[2]Queen's University
pascal.bercher@anu.edu.au, patrik.haslum@anu.edu.au, christian.muise@queensu.ca

Abstract

Automated Planning deals with finding a sequence of actions that solves a given (planning) problem. The cost of the solution is a direct consequence of these actions, for example its number or their accumulated costs. Thus, in most applications, cheaper plans are preferred. Yet, finding an *optimal* solution is more challenging than finding *some* solution. So, many planning algorithms find some solution and then post-process, i.e., optimize it – a technique called *plan optimization*. Over the years many different approaches were developed, not all for the same kind of plans, and not all optimize the same metric. In this comprehensive survey, we give an overview of the existing plan optimization goals, their computational complexity (if known), and existing techniques for such optimizations.

1 Introduction

Automated Planning aims at finding a course of action based on a declarative model of the the environment, the actor's available actions, and a goal to achieve. There are many different kinds of planning frameworks [Geffner and Bonet, 2013], some more complex than others: The simplest ones build on a model, where there is just a single agent, the environment is fully observable, and actions are deterministic. Extensions include, among others, dealing with time, resources, uncertainty, and all of them with or without a hierarchy defined among the actions. Solutions to such problems are usually one of the following two: *(1)* Totally or partially ordered sequences of actions that one has to execute in an order adhering the given ordering constraints, or *(2)* policies that map states to actions (this is usually deployed in settings with uncertainty, as a simple action sequence is not expressive enough to tell how to act).

Finding *optimal* solutions to planning problems is commonly known to be a much harder task than finding *any*, i.e., suboptimal solution. This is effectively demonstrated by Helmert and Röger [2008] who provided a seminal analysis showing that on certain planning benchmarks A^* is bound to explore an exponentially large search space even with an almost perfect heuristic, which they define as a heuristic h that in each state equals the perfect heuristic h^* minus a small constant. That is, in some domains searching for an optimal solution is *bound* to require a large effort, making it infeasibly hard. It is often more efficient to choose a suboptimal search procedure, which is not guaranteed to find optimal solutions, and thus usually does not. Having a (potentially) suboptimal solution at hand, one may then opt for post-optimizing it, i.e., by improving it without simply continuing the search.

Different plan metrics (such as number of actions, their costs, number of ordering constraints, or number of their induced linearizations) have been considered for optimization, their computational complexities investigated, and several kinds of techniques have been devised over the years. In this paper, we comprehensively survey plan optimization criteria, their complexities, and techniques. To the best of our knowledge, this is the first survey of such breadth, although an overview focused on plan cost optimization is, despite missing developments past its publication, still worth mentioning [Kilani and Chrpa, 2013].

We start with Section 2, which gives a brief overview of work that is only *somehow* related to our survey. More specifically, we narrow down the exact scope of the survey while also mentioning lines of research which technically fall outside of it, but are still worth being mentioned. Section 3 provides key concepts of the formalizations underlying most of the surveyed work. We then start with our actual survey in three main sections:

- Section 4 reviews known complexity results related to plan optimization.
- Section 5 is concerned with practical approaches that aim to optimize a given solution plan's *actions*, i.e., by substituting or removing them thus reducing costs.
- Section 6 reviews work that optimizes plans' orderings, i.e., by rearranging actions or removing ordering constraints. This is usually important in practice where flexibility is crucial or execution order plays a role that's not captured in the plan metric used to generate solutions.

Finally, in Section 7 we conclude this survey.

2 Scope of This Survey

In this paper, we survey literature that directly works on a given solution plan (or policy) and tries to improve it, specifically by removing or replacing parts of it – as this is what is usually referred to as *plan optimization*.

However, in this section, we want to mention some lines of research which are closely related to the endeavor of finding a better plan, but do not quite match our precise inclusion criterion from above. We do so as some of these research directions can still be used to find better plans.

Anytime and Incremental Search. There are several search algorithms that, upon encountering a solution, continue the search, systematically, for a solution of better quality, eventually converging to an optimal solution. The simplest exemplar is branch-and-bound [Land and Doig, 1960], but versions of the idea are found in, for example, beam-stack search [Zhou and Hansen, 2005], anytime $A^\star$ [Hansen and Zhou, 2007], and limited discrepancy (beam) search [Harvey and Ginsberg, 1995; Furcy and Koenig, 2005]. We exclude such algorithms from this survey because, although they may use information about the best solution found so far, they also rely on other information about the search process so far (e.g., branches of a search tree that have already been explored and found not to yield an improving solution). Population-based meta-heuristics (see, e.g., the survey by Blum and Roli [2003]) also share this characteristic, without the convergence guarantee.

Bounded-Cost and Bounded Suboptimal Search. The default objective in non-optimal planning (or search) is to simply produce *any* plan as quickly as possible, i.e., without regard for quality. A bounded-cost search algorithm takes as input a cost bound, and aims to find a solution within that bound as quickly as possible, i.e., without expending effort on achieving a better-quality solution than required by the bound [Stern *et al.*, 2011; Thayer *et al.*, 2012]. Bounded suboptimal search algorithms, the most famous of which is Weighted $A^\star$ [Pohl, 1970], take a relative bound parameter w and ensure the solution found is within a factor w of optimal. A series of increasingly better plans can be found by repeatedly applying a bounded-cost or suboptimal search algorithm, with the bound in each iteration set to yield an improvement on the last plan found [Richter *et al.*, 2010]. Although this can be viewed as a form of plan improvement, it is uninformed in that it makes no use of the input plan beyond its cost or suboptimality bound.

Diverse planning. Producing a set of *diverse* plans for a problem instance can be useful to achieve robustness, in adversarial settings, or when the user's objective function is not fully known. The diversity of the plan set can be combined with the objective of optimizing the quality of each plan in the set in different ways [Katz and Sohrabi, 2020]. Similar to plan quality optimization, while some approaches to diverse planning have used continuing search [Srivastava *et al.*, 2007; Nguyen *et al.*, 2012], a set of diverse plans can also be generated incrementally, by repeatedly solving the problem with the added constraint that the next plan must not have some property shared by all plans in the set found so far; such constraints can be compiled into the problem [Katz and Sohrabi, 2020], or expressed using PDDL3's trajectory constraints [Alfonso Gerevini, pers. comm.]

Plan Repair and Reuse. Plan repair deals with the situation where a plan fails in execution due to unexpected state changes, inaccuracies in the domain model, or perhaps due to some guess and verify/repair approach. The goal is then to find a new, valid plan, given the updated world knowledge if any. While replanning from scratch is one option, repairing the failed plan by making limited changes may save time, and there can also be a value in minimizing the deviation from the original plan, for instance if the changes need to be communicated to and understood by (multiple) humans. In plan adaptation, a solution plan is known for a problem instance that is similar to the present one, and the goal is to adapt it to the present problem to avoid having to solve it from scratch (see, e.g., the survey by Borrajo *et al.* [2015]). Plan repair and adaptation approaches have been developed for classical [Gerevini and Serina, 2000], numeric [Scala and Torasso, 2015] and hierarchical planning [Bercher *et al.*, 2014; Höller *et al.*, 2020; Barták *et al.*, 2021]. Several works compared plan-reuse with planning from scratch, both empirically, theoretically, and with different plan similarity measures [Nebel and Köhler, 1995; Fox *et al.*, 2006; Babli *et al.*, 2023]. The similarity with plan optimization is that a plan is given, and the aim to exploit that in the pursuit of a different plan. However, in plan repair or adaptation, the main objective is to restore or achieve plan validity, not improving plan quality or cost, and typically under stricter time or resource constraints.

Model Repair. Another line of research that is worth mentioning in our context is *model repair*. In one of its settings, one is given a plan that is *supposed* to be a solution, but is not (similar to plan repair/reuse). In order to make it a solution, the underlying *model* gets repaired (as opposed to the plan as done in plan repair/reuse). Works exist both for classical planning [McCluskey *et al.*, 2002; Lin and Bercher, 2021; Lin *et al.*, 2023; Gragera *et al.*, 2023] as well as in hierarchical planning [McCluskey *et al.*, 2002; Lin and Bercher, 2021; 2023]. In both settings, checking whether such changes are possible is NP-complete [Lin and Bercher, 2021; 2023]. Closely related is what's called *model reconciliation* [Sreedharan *et al.*, 2021]. Here, on top of being provided a plan and a planning model, a second model is being provided. One model is supposed to be the ground truth (by some robot) whereas the other is a human's mental model of it. The plan is optimal in the ground truth, but not according to the human's model thereof. To explain the discrepancy, a minimal set of change operations to the human's model is desired (the explanations) that make the plan optimal in that new model. Checking whether achieving this is possible with at most k model changes was proved to be Σ_2^P-complete [Sreedharan *et al.*, 2022], a class that is NP-hard and presumably slightly harder. In the work surveyed in the remainder we do however not change the model, but assume its correctness.

Plan Explanation. Lastly, we would like to point out a loose relationship between plan optimization and *plan explanations* [Chakraborti *et al.*, 2020]. Whereas some plan explanation approaches (specifically those that aim at explaining the necessity or purpose of an action in a plan[1]) do so by simply "explaining" a chain of actions from said action to a goal, sometimes by following a chain of causal links [Bercher

[1]For a comprehensive overview of possible kinds of explanations in general we refer to the work by Miller [2019].

et al., 2014] (we provide a definition of causal links in Section 3), these approaches are problematic for several reasons [Lindner and Olz, 2022]. For example, simply explaining the purpose of an action in a plan might be counter-intuitive if that action turns out to not be required in that plan, i.e., if an optimization exists that removes that action. Thus, one observation is that if a given plan can not be optimized to remove a specific action, then this forms an argument for the necessity of that action (in that plan). Thus, some plan explanation techniques do require that the plan to be explained is already optimized [Lindner and Olz, 2022] thus further showing the necessity for optimization.

3 Problem Formalization

We survey plan optimizations for formalisms in different frameworks. The simplest model of planning bases on classical STRIPS models where solutions are totally ordered sequences of actions in a fully observable deterministic environment. Most extensions to this model require significant changes to the underlying formalism. For example, for dealing with uncertainty, definitions of actions differ and even plans (i.e., solutions) are extended to policies, mapping states to actions with certain guarantees on success probabilities. Since providing all these different frameworks on a formal level is neither feasible given the space constraints, nor required for the sake of our summary, we won't provide most formalisms on a deep technical level. Instead, we provide only the most simple formalization – the STRIPS model – formally as well as some base concepts for its extension to hierarchical planning and explain differences when required.

3.1 Classical (STRIPS) Planning

A classical problem can be defined in a propositional fashion as a 4-tuple $\langle F, A, s_I, g\rangle$, where F is a finite set of facts (encoding all relevant state properties), $A \subseteq 2^F \times 2^F \times 2^F$ is a (finite) set of actions (encoding state transitions), $s_I \in 2^F$ is the initial state (from which planning or plan execution starts), and $g \subseteq F$ is the goal description (the set of facts required to hold eventually). Each action $(pre, add, del) \in A$ consists of a precondition $pre \subseteq F$, add list $add \subseteq F$, and delete list $del \subseteq F$. Given an action $a \in A$, we refer to these three elements as $pre(a)$, $add(a)$, and $del(a)$, respectively. The precondition specifies which properties must hold for an action to be applicable in a state, i.e., an action $a \in A$ is applicable in a state $s \in 2^F$ if and only if $pre(a) \subseteq s$. If a is applicable in s, then it leads to a successor state defined by its effects, $s' = (s \setminus del(a)) \cup add(a)$. Applicability of a sequence of actions is defined in the canonical sense, requiring that each action is applicable in the state right before it. Finally, an action sequence $\bar{a} = a_1, \ldots, a_n$ is called a solution (or plan) if it is applicable in the initial state s_I and leads to a state s' that makes all goals true, $s' \supseteq g$.

Whereas STRIPS is often chosen for simplicity in scientific papers, in practice, problems are provided in a lifted fashion, where facts and actions base on a predicate logic. Since some of the surveyed approaches base on such a formalization as well, we provide a running example based on such a lifted formalization: In the well-known Blocksworld domain, we are given a set of towers of blocks, where each block lies either on the table or on another block. An action pickup(?b) picks up a block ?b (we use question marks to denote variables) from the table into a gripper, whereas unstack(?b$_1$,?b$_2$) removes a clear (top-most) block ?b$_1$ from another block ?b$_2$. Actions stack(?b$_1$,?b$_2$) and putdown(?b$_1$) put down a block from the gripper onto another block or onto the table, respectively. Such lifted representations can be turned into propositional ones by *grounding*, i.e., by replacing variables by constants (here, each constant A, B, $\ldots$ represents one block).

Because plan optimization, as well as the respective complexity results, depend on the representation of plans, we first present different ways that plans may be represented:

- Sequential plans (action sequences)
- Partially ordered plans (PO plans)
- Block-decomposed PO plans (BDPO plans)
- Partial Order Causal Link plans (POCL plans)

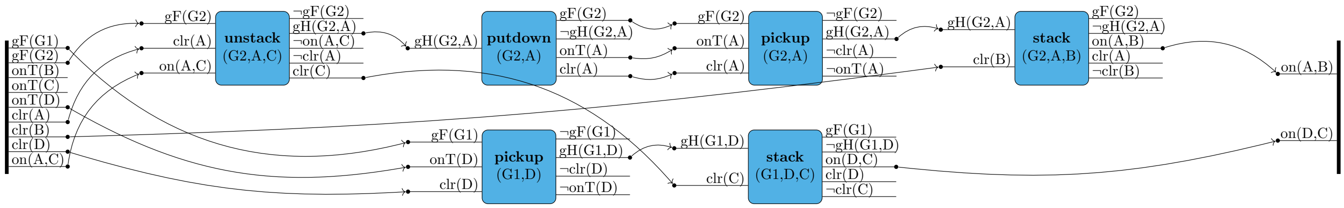
A *Sequential plan* is simply an action sequence, as defined for STRIPS. A *PO plan* maintains only a partial order among actions [Nebel and Bäckström, 1994; Bäckström, 1998]. For a PO plan to be valid requires that every sequence of the plan's actions compatible with the partial order (a so-called *linearization*) is a valid sequential plan. A *BDPO plan* extends the PO plan representation with a decomposition of the actions into subsets (termed "blocks"), which may be recursive. Like a PO plan, it represents a set of linearizations, but with the additional restriction that actions in a block cannot be interleaved with actions outside the block [Siddiqui and Haslum, 2012; 2013; 2015].[2] For example, in Fig. 1(b), the two 2-action blocks on the right can be placed in any order, but not interleaved. *POCL plans* are another extension of PO plans, introducing so-called *causal links* [Bercher, 2021], which make explicit which action's effects will satisfy each precondition and goal, thus making it easier to reason about the plan's validity. More precisely, each causal link *supports* and thus protects a single precondition p. Every action that deletes p must be ordered before the link's producer or after the link's consumer to resolve this so-called *causal threat*. Most algorithms and encodings do use POCL plans rather than PO plans. Fig. 1(a) shows an example POCL plan in the Blocksworld domain with two grippers.

In the context of partially ordered plans, one is often interested in (the optimization of) various properties concerning the partial order, specifically in *number of linearizations*, *the number of ordering constraints*, or *the makespan*, which is the critical path length, i.e., execution time when assuming that non-conflicting actions are executed in parallel (for the plan in Fig. 1(a) this would be 4, while for both plans in Fig. 2, where no parallelism is possible, it is 6).

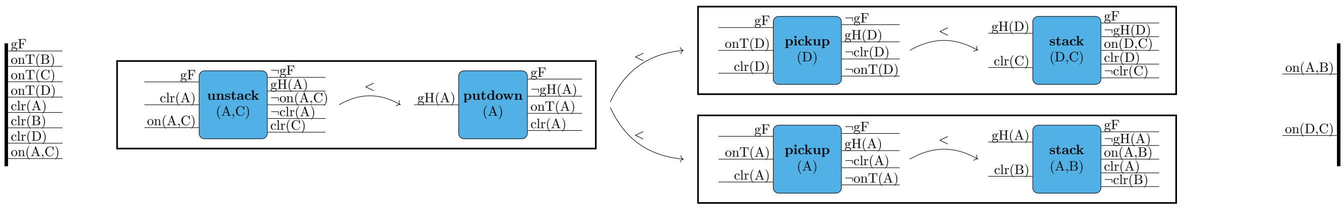
3.2 Hierarchical Planning

Hierarchical task network (HTN) planning is an extension of classical planning [Erol *et al.*, 1996; Ghallab *et al.*, 2004; Bercher *et al.*, 2019] that can easiest be described as an additional constraint that solution plans must satisfy: Here action

[2] Note that we only cite the works by Siddiqui and Haslum [2012; 2013] for the sake of completeness of this survey but would like to note that their article from 2015 subsumes them completely.

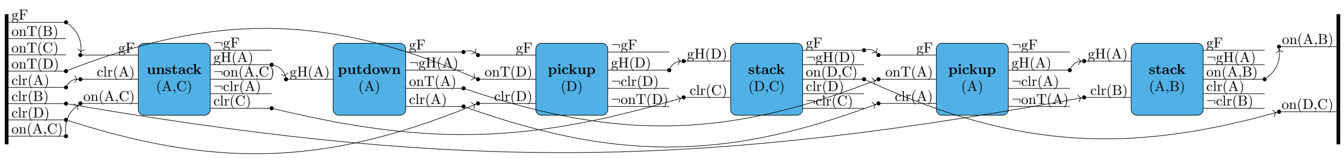


Subfigure a: A POCL plan for the 2-gripper Blocksworld problem. It is not totally ordered.

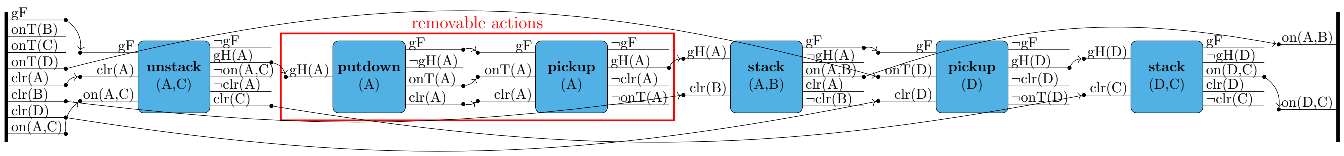


Subfigure b: A BDPO plan to the 1-gripper Blocksworld problem.

Figure 1: Example plans. Arrows between preconditions and effects denote causal links, other arrows are ordering constraints. Predicate abbreviations: clr means clear, gF means gripperFree, gH means gripperHolding, and onT means onTable.



Subfigure a: A totally ordered POCL plan that solves the underlying Blocksworld problem.



Subfigure b: A reordering of the plan in Fig. 2(a) with causal link structure changed accordingly.

Figure 2: Two totally ordered POCL plans, each a re-ordering of the other. Both solve the depicted problem.

sequences don't just have to be executable and make the goal true, but additionally need to lie within "the language" of a given grammar. For this latter restriction, HTN problems also contain compound tasks as well as so-called decomposition methods, which map a single compound task to a partially (or totally) ordered sequence of compound or primitive tasks (primitive tasks are exactly the actions). Thus, HTN planning organizes actions in a hierarchical manner, perfectly analogous to formal grammars. Whereas the language of a formal grammar equals all strings of terminal symbols that can be produced by the grammar, the set of solutions to an HTN planning problem is defined as the set of all executable and goal-achieving action sequences that can be obtained from a given initial compound task. As mentioned initially, the underlying task hierarchy can hence be regarded as an additional constraint, ruling out those action sequences which do not adhere to the HTN problem's grammar/task hierarchy. We include it here, since a range of results for plan optimization have also been achieved for this hierarchical setting. For further details on the formalism we refer to the survey by Bercher *et al.* [2019].

3.3 Plan Optimization Problems

Before we start our survey we provide an informal introduction to the most basic questions regarding plan optimization that have been studied in the literature.

- Given plan P, is there a smaller or cheaper plan P' (where P' is obtained via removal or the substitution of actions from P), such that P' still is a solution to the underlying problem?
- Given plan P, is there a *reordering* or *deordering* of P with certain properties, e.g., smaller makespan or smaller number of ordering constraints? Here, a *reordering* is a plan with the same actions but possibly different ordering constraints, and a *deordering* is a plan with the same actions and a subset of the ordering constraints.

Plans of the first kind are usually referred to as *redundant*, and most work within the post-optimization literature is on identifying and removing such (action) redundancies.

4 Complexity Investigations

We now look into complexity investigations of post-optimization problems, starting with improving a plan's actions, and then its orderings or linearizations.

4.1 Optimization of Plan Length/Cost

Fink and Yang [1992] defined four different levels of non-redundancy, which they term *justification*, of a (sequential) plan: A plan is *backward-justified* if every of its actions is the last achiever of a fact in the precondition of some later action or the goal; *well-justified* if no single action can be removed without invalidating the plan; *perfectly justified* if no proper subsequence of it is also a valid plan; and *greedily justified* if removing any action and its direct and indirect dependents invalidates the plan. Fig. 2(a), when ignoring the causal link structure, depicts a perfectly justified sequential plan. Fig. 2(b) is neither perfectly justified nor greedily, but satisfies all other criteria. For all criteria except perfect justification, a subplan achieving them can be computed in polynomial time. For perfect justifications, Fink and Yang [1992] proved the respective decision problem to be NP-complete. Later and independently Nakhost and Müller [2010b] also showed that, given a (sequential) plan P and a cost value k, determining whether a subplan of P with cost smaller than k exists is NP-hard.

Note how the problem becomes hard because we neither know how many actions might be removed nor which ones, so one has to systematically try out all these options. In a much more recent work, Olz and Bercher [2019] have shown that the corresponding problem becomes significantly harder when reasoning about partially ordered plans. More specifically, in the context of POCL plans, they showed that even checking whether a single given action can be removed is already NP-complete. The complexity arises from checking whether the partial order and causal links can be refined again (via ordering and causal link insertion) after action deletion so that all linearizations remain executable.

Only loosely related is the question of whether a given (sequential or partially ordered) plan is *optimal*. Of course, if a plan is an optimal solution, then the plan can also not be optimized (hence, optimality implies perfect justification). If, however the plan is not an optimal solution, it might still be perfectly justified. Determining plan(length) optimality is naturally co-NP-complete – both for (ground) classical and HTN planning [Lin *et al.*, 2024].

The work by Barták *et al.* [2021] technically falls both into the category of plan repair and plan optimization. Starting from a non-solution action sequence for an HTN planning problem, they seek to optimize (i.e., repair) it by identifying a minimal number of actions to remove from it so that the resulting plan turns into a solution. Their motivation is to stay as close to the input plan as possible (hence we can regard it as optimization of a given plan). They showed this problem to be, unsurprisingly, NP-complete. Also in the context of HTN planning, Behnke *et al.* [2016] investigated the complexity of (user-provided) change requests: given a plan for an HTN problem, the complexity of changing certain plan properties, such as enforcing that a certain action occurs in the modified solution, or to avoid a certain action altogether, was investigated. Depending on the exact change request, complexity ranges between NP-complete to undecidable. In all HTN optimization approaches, an additional source of computational hardness always arises from the verification that checks whether the resulting task network still lies in the task hierarchy – which is by itself already NP-hard unless everything is totally ordered [Behnke *et al.*, 2015].

4.2 Optimization of Orderings and Linearizations

For the optimization of ordering constraints, most work naturally involves PO or POCL plans. Nebel and Bäckström [1994], building on work by Dean and Boddy [1988] investigated complexities for temporal projection, planning, and plan verification. These works are not directly related to plan optimization, but given that they are seminal works related to partially ordered plans, we wanted to mention them nevertheless (notably, investigations include the question whether an executable linearization exists, which is inherent in many optimizations). Bäckström [1998] investigated a range of complexity questions regarding the reordering and deordering of PO plans, such as deciding whether a reordering or deordering with a given bound on the number of resulting ordering constraints exists. Although he showed several reasoning tasks to be NP-complete, we would like to emphasize the depth and width of the analyses, which makes it challenging to succinctly summarize all the detailed findings without omitting significant aspects. In addition, Bäckström also investigated what he called *parallel* plans, an extension of PO plans with the addition of non-concurrency constraints which forbid certain actions to execute in parallel even if otherwise feasible. More recently, Aghighi and Bäckström [2017] re-visited several of these findings, conducting a parameterized complexity analysis, considering various parameters, such as the original and desired size of the ordering and non-concurrency relations, among others.

Bercher and Olz [2020] also extended the work by Bäckström [1998]. More specifically, Bäckström [1998] left open one interesting question, namely about the computational hardness of checking for the existence of the deordering of PO plans with a specific makespan. This was proved NP-complete [Bercher and Olz, 2020]. Another contribution of the paper by Bercher and Olz [2020] was to make aware of subtle but important difference between PO plans and POCL plans, namely that there exist POCL plans for which there do not exist PO plans with the exact same ordering constraints and linearizations (cf. their Figure 1). This is also important in the context of plan optimization as it proves that the chosen data structure – PO plan versus POCL plan – influences which orderings and linearizations one could obtain.

Finally, note that the work by Behnke *et al.* [2016] on plan optimization/change requests for HTN plans, mentioned earlier, also covers various changes/requests related to the ordering constraints.

5 Approaches to Optimize Plan Length/Cost

The biggest body of work exists for improving a given solution. This involves questions like checking whether a certain (number of) action(s) is redundant and whether they could even be replaced by others. We organize this section by how the different kinds of approaches work.

5.1 Removing Redundant Actions

The tractable plan optimizations identified by Fink and Yang [1992] (cf. last section) have been used by several researchers: Nakhost and Müller [2010a; 2010b] applied greedy justification (under the name "action elimination") along with their PNGS method (discussed below). Balyo *et al.* [2014] extended the greedy justification algorithm to consider action costs, and select the greedily unjustified action set with the highest total cost to remove. They also implemented perfect justification via a MaxSAT encoding, and the action-cost sensitive formulation of it (i.e., finding the valid subplan of minimum cost) using Weighted MaxSAT. This allowed them to compare, empirically, the plan length and cost reduction achieved by the tractable algorithms compared to the perfect result. Their greedy action elimination achieved over 80% of the perfect cost reduction for all but one of the planner–domain pairings in their benchmark set. Interestingly, the performance of the tractable justification algorithms, relative to perfect-cost justification, seemed to depend to some extent on which planner had generated the input plan. Salerno *et al.* [2023] implemented cost-sensitive perfect justification via (optimal) classical planning instead. Med and Chrpa [2022] revisited action elimination, focusing on how to speed up the search for redundant action subsequences, for instance by identifying actions in the plan that are easily shown to be not redundant.

The notion of subplan above preserves the order of remaining actions from the original plan. However, deordering and reordering can also play a key role in identifying and removing redundancy from a plan. Consider our earlier Blocksworld problem with one gripper, initial state {on(A,C), onTable(B), onTable(C), onTable(D), clear(A), clear(B), clear(D), gripperFree}, and the goal of making on(A,B) and on(D,C) true. Two sequential plans for this problem are shown in Fig. 2. No subsequence of the first plan (Fig. 2(a)) is a valid plan; this plan is already perfectly justified. The second plan however (Fig. 2(b)), is a reordering of this plan (same actions, but in a different, but also valid, order); in this plan the subsequence putdown(A), pickup(A) can be removed. Chrpa *et al.* [2012a; 2012b] used this idea for searching for pairs of such inverse actions that can occur next to each other after plan deordering.

Similar to the work by Balyo *et al.* [2014], the work of Muise *et al.* [2012; 2016][3] uses a MaxSAT encoding to compute plans with a subset of actions that have a minimum total cost. This approach simultaneously reorders the plan and is described further in Section 6 below.

Sreedharan *et al.* [2023] introduce a generalized form of action justification that applies to conditional plans or policies for non-deterministic domains. While they do not explore the impact of optimizing these conditional plans via action removal, that is a natural direction of future work.

In HTN planning, the task of plan optimization has not yet seen as much attention as non-hierarchical planning. The only approach we are aware of is that by Barták *et al.* [2021], who exploit the similarity between formal grammars and HTN problems by adapting a parsing-based approach for plan verification to identify the minimal number of actions that can be deleted from a plan to turn it into a solution.

5.2 Replacing Subplans

The next step from removing redundant parts of a plan is to replace parts of it, such that the total plan length, cost, or makespan is reduced. The search for a replacement subplan is itself a planning problem: restricting it to a (small) segment of the original plan, with bounded cost, restricts the potential search depth, hence the expectation is that the subproblems will be easier to solve. Ratner and Pohl [1986] select fixed-size consecutive segments along a path in a graph and search for a shorter replacement path for each. Estrem and Krebsbach [2012] select pairs of states along the plan trajectory that maximize an estimate of redundancy: the ratio between a heuristic estimate of distance between the two states and the cost of the current plan segment. Balyo *et al.* [2012a; 2012b] apply sliding and randomly selected fixed-size windows over a parallel plan, using a SAT encoding to decide if a shorter (by parallel length) replacement subplan exists.

Also in this case, deordering or reordering the plan allows greater opportunities to identify improvable subplans, as the actions of the subplan do not have to be consecutive in the input plan. Siddiqui and Haslum [2015] showed that applying their iterated subplan improvement procedure to a BDPO representation of the input plan yielded around 40% greater improvement compared to the same procedure applied to the original, totally ordered, plan. This is because the block deordering can represent reorderings of the original plan: for example, the two reorderings in Fig. 2 are both linearizations of the BDPO plan in Fig. 1.

The planning-by-rewriting approach by Ambite and Knoblock [2001] also uses local modifications of partially ordered plans to improve their quality. Plan modifications are made by domain-specific rewrite rules, which have to be provided by the domain designer or learned from many examples of both good and bad plans.

Relaxing Specific Actions. Closely related to replacing subplans is the work by Waters *et al.* [2018; 2020]. Their approach works on a lifted action representation, and considers substituting actions' arguments. Their approach thus "technically" fits into this section since the resulting *ground* plans are obtained by replacing the original ground actions from the input plan (although from the same lifted action schemas). However, their goal is to improve plan flexibility, so we review it in more detail in the next section, which is concerned with the optimization of orderings and linearizations.

Searching the Plan Neighborhood. Plan Neighborhood Graph Search (PNGS) [Nakhost and Müller, 2010a] constructs a subgraph of the state space of the problem, built around the path induced by the current plan by expanding only states within a limited distance from those on that path. It then searches for the least-cost plan in this subgraph. ITSA* [Furcy, 2006] similarly explores an area, called a tun-

[3]We only cite the work by Muise *et al.* [2012] for the sake of completeness but note that their 2016 article subsumes it completely.

nel, of the state space, restricted to a fixed distance from the current plan. Compared to the subplan replacement methods discussed above, PNGS is not limited to finding a replacement for a selected part of the input plan – it can replace the whole plan – but instead, to limit search effort, restricted in how far away from the current plan it can venture in the search space, which subplan replacement methods are not. All the above methods can be seen as exploring a neighborhood around the current plan in the space of valid plans. Thus, iterating them, stepping from one better plan to the next, is a form of local search, similar to Large Neighborhood Search (LNS) [Shaw, 1998].

Westerberg and Levine [2001] applied a genetic algorithm to optimize plans. In difference to other approaches described above, this requires as input a collection of plans, to form the initial population, and can produce invalid as well as valid new plans. Their implementation is limited to 1-point crossover and 1-point mutation operators, suggesting it would not be able to find plan reorderings or eliminate redundant subplans of more than one action.

6 Approaches to Optimize Orderings and Linearizations

Even when a (solution) plan has been found that is acceptable regarding its solution costs or more precisely regarding the exact actions that are within the plan, depending on the application at hand it might still be important in which order those actions are executed or more specifically at which time step each action should be scheduled – examples are to convey instructions to human users in a sequence that are reasonable to them [Bercher *et al.*, 2014] or to obtain flexible plans with a higher chance of successful continuation, even in case of execution errors [Muise *et al.*, 2011]. In this section, we detail the approaches introduced to optimize the flexibility of partially ordered plans, in many cases causing a measurable increase in the number of linearizations a plan can represent.

To maximize the flexibility of a plan, a common approach is to minimize the number of ordering constraints in the respective PO or POCL plan while maintaining validity – every linearization must still be a sequential plan that achieves the goal from the initial state. Due to the combinatorics involved, a proxy for maximizing the number of linearizations is typically employed: minimizing the ordering constraints in the transitive closure of such a plan. Muise *et al.* [2016] demonstrated a strong correlation between this optimization criterion and the linearizations a POCL plan can represent.

One early work for minimizing the ordering constraints of a PO plan is due to Kambhampati and Kedar [1994]. They introduce a polynomial algorithm that converts a sequential plan into a partially ordered one by maintaining constraints between actions a_i and a_j from the total order for only one of three reasons: *(1)* a_i is the earliest achiever of a precondition of a_j that is unthreatened (i.e., no action between a_i and a_j in the sequential plan deletes the fact); *(2)* a_i deletes a fact that a_j adds for another action ordered after a_j; or *(3)* a_j deletes a fact that is a precondition of a_i. The algorithm is extremely efficient, and produces a deordering of the input plan. It was shown to theoretically not be minimal, let alone a minimum ordering [Bäckström, 1998], but empirically it finds a minimum deordering among all POCL plans for the vast majority of existing benchmarks [Muise *et al.*, 2016].

When it comes to finding an optimal deordering or reordering of plans, research has historically turned to combinatorial optimization. In particular, MaxSAT [Muise *et al.*, 2016; Waters *et al.*, 2020], MIP [Do and Kambhampati, 2003; Say *et al.*, 2016], and CSP [Waters *et al.*, 2018]. Most of these models share common high-level constraints to capture valid POCL plans: *(i)* Every precondition of an action needs a supporter. *(ii)* Any action that might delete (or threaten) a supported precondition must be ordered before the supporter or after the action that requires the fact. *(iii)* The ordering constraints included in the computed plan are transitively closed.

Further, while not every one of the above works presents a version for deordering, the encodings that are used for computing optimal reorderings are easily extended to accomplish this. For any pair of actions a_i and a_j in the input plan such that a_i is ordered before a_j, we add a constraint that forbids a_j being ordered before a_i. Note that this requires the transitive closure of ordering constraints to be included.

The MaxSAT encodings of Muise *et al.* [2016] jointly optimize for a reordering (optionally a deordering) and the total cost of actions included in the computed plan. This is achieved by assigning a cost of including an action to outweigh the cost of all possible ordering constraints (thus optimizing total action cost before considering ordering constraints). The MIP model of Say *et al.* [2016] encodes things similarly but achieves an improved performance through a different solving technology.

The MIP model of Do and Kambhampati [2003] shares many similarities, but also includes further modeling aspects to optimize temporal aspects of a plan: the input is a sequence of actions with specific time points and there are action durations specified. Further optimization metrics (e.g., total makespan) are considered as well for this work.

Finally, the CSP and MaxSAT models of Waters *et al.* [2018; 2020] optimize not only the ordering constraints of the input plans, but relax the action selection as well through *debinding* [Waters *et al.*, 2018] or *reinstantiation* [Waters *et al.*, 2020]. Debinding refers to relaxing the object parameters used for actions (e.g., turning $\mathsf{stack(D,C)}$ into $\mathsf{stack(D,?}b_2\mathsf{)}$ so that the plan effectively works for any object $?b_2$ of the right type), while reinstantiation allows for a change in those parameters to a new object. These extensions to the general task of relaxing the ordering constraint can be seen as a form of action optimization that allows for improved flexibility. Actions are not removed (and so total plan cost remains), but actions are replaced in order to allow for fewer ordering constraints.

7 Conclusion

Plan optimization is of practical importance because plan quality is often a crucial concern, measures of quality can be many and diverse, and it is often significantly easier to generate any valid plan without striving for optimality. In this paper, we provided a comprehensive overview of the rich variety of techniques that exist for plan optimization. There are some interesting takeaways from the collected works: First, while theory shows that many plan optimization problems are NP-hard, several studies have found that simple/greedy (and tractable) plan optimization approaches empirically perform very well, compared to the optimal/combinatorial solutions [Balyo *et al.*, 2014; Muise *et al.*, 2016]. Hence, unpacking the empirical difficulty of optimizing certain plans is an open question: Are there plans that are simultaneously highly sub-optimal and hard to optimize? And are there domains and problems for which current non-optimal planners will generate such plans? Second, there are synergies between different optimization goals. For instance, reducing orderings, or finding plan reorderings, can lead to greater opportunities for replacing subplans, thus reducing plan cost; replacing actions can remove threats and thus allow greater deordering; and, of course different plan optimization methods, which have different neighborhoods in the space of plans can be interleaved. The practical impacts of such combinations and their theoretical limitations are largely unexplored. Also, on the theoretical front, the complexity and potential of block decomposition in deordering plans are open questions.

Finally, complementary to plan optimization is lower bounding, i.e., computing bounds on how much further a plan can be optimized The combination of incremental lower bounding [Haslum, 2012; Seipp and Helmert, 2013; Davies *et al.*, 2015] and incremental plan optimization techniques may provide a powerful way to control the optimality gap in applications of planning. Verifying the optimality of a plan is equivalent to proving unsolvable a bounded planning problem with a bound ϵ less than the plan's cost (where the required value of ϵ depends on the range of values of the plan cost metric, e.g., it is 1 if costs are integral). Proving, efficiently, the unsolvability of planning problems is a question that has only relatively recently gained interest in the field [Muise and Lipovetzky, 2016].

Acknowledgements

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government. Christian Muise gratefully acknowledges funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[Aghighi and Bäckström, 2017] M. Aghighi and C. Bäckström. Plan reordering and parallel execution – a parameterized complexity view. In *AAAI 2017*, pages 3540–3546. AAAI Press, 2017.

[Ambite and Knoblock, 2001] J. L. Ambite and C. A. Knoblock. Planning by rewriting. *JAIR*, 15(1):207–261, 2001.

[Babli *et al.*, 2023] M. Babli, Ó. Sapena, and E. Onaindia. Plan commitment: Replanning versus plan repair. *EAAI*, 123:106275, 2023.

[Balyo *et al.*, 2012a] T. Balyo, R. Barták, and P. Surynek. On improving plan quality via local enhancements. In *SoCS 2012*, pages 154–156. AAAI Press, 2012.

[Balyo *et al.*, 2012b] T. Balyo, R. Barták, and P. Surynek. Shortening plans by local re-planning. In *ICTAI 2012*, pages 1022–1028. IEEE Press, 2012.

[Balyo *et al.*, 2014] T. Balyo, L. Chrpa, and A. Kilani. On different strategies for eliminating redundant actions from plans. In *SoCS 2014*, pages 10–18. AAAI Press, 2014.

[Barták *et al.*, 2021] R. Barták, S. Ondrčková, G. Behnke, and P. Bercher. Correcting hierarchical plans by action deletion. In *KR 2021*, pages 99–109. IJCAI, 2021.

[Behnke *et al.*, 2015] G. Behnke, D. Höller, and S. Biundo. On the complexity of HTN plan verification and its implications for plan recognition. In *ICAPS 2015*, pages 25–33. AAAI Press, 2015.

[Behnke *et al.*, 2016] G. Behnke, D. Höller, P. Bercher, and S. Biundo. Change the plan – how hard can that be? In *ICAPS 2016*, pages 38–46. AAAI Press, 2016.

[Bercher and Olz, 2020] P. Bercher and C. Olz. POP $\equiv$ POCL, right? complexity results for partial order (causal link) makespan minimization. In *AAAI 2020*, pages 9785–9793. AAAI Press, 2020.

[Bercher *et al.*, 2014] P. Bercher, S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg. Plan, repair, execute, explain – how planning helps to assemble your home theater. In *ICAPS 2014*, pages 386–394. AAAI Press, 2014.

[Bercher *et al.*, 2019] P. Bercher, R. Alford, and D. Höller. A survey on hierarchical planning – one abstract idea, many concrete realizations. In *IJCAI 2019*, pages 6267–6275. IJCAI, 2019.

[Bercher, 2021] P. Bercher. A closer look at causal links: Complexity results for delete-relaxation in partial order causal link (POCL) planning. In *ICAPS 2021*, pages 36–45. AAAI Press, 2021.

[Blum and Roli, 2003] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[Borrajo *et al.*, 2015] D. Borrajo, A. Roubíčková, and I. Serina. Progress in case-based planning. *ACM Computing Surveys*, 47(2):1–39, 2015.

[Bäckström, 1998] C. Bäckström. Computational aspects of reordering plans. *JAIR*, 9:99–137, 1998.

[Chakraborti *et al.*, 2020] T. Chakraborti, S. Sreedharan, and S. Kambhampati. The emerging landscape of explainable automated planning & decision making. In *IJCAI 2020*, pages 4803–4811. IJCAI, 2020.

[Chrpa *et al.*, 2012a] L. Chrpa, T. L. Mccluskey, and H. Osborne. Determining redundant actions in sequential plans. In *ICTAI 2012*, pages 484–491. IEEE Press, 2012.

[Chrpa *et al.*, 2012b] L. Chrpa, T. L. McCluskey, and H. Osborne. Optimizing plans through analysis of action dependencies and independencies. In *ICAPS 2012*, pages 338–342. AAAI Press, 2012.

[Davies *et al.*, 2015] T. O. Davies, A. R. Pearce, P. J. Stuckey, and N. Lipovetzky. Sequencing operator counts. In *ICAPS 2015*, pages 61–69. AAAI Press, 2015.

[Dean and Boddy, 1988] T. Dean and M. Boddy. Reasoning about partially ordered events. *AIJ*, 36:375–399, 1988.

[Do and Kambhampati, 2003] M. B. Do and S. Kambhampati. Improving the temporal flexibility of position constrained metric temporal plans. In *ICAPS 2003*, pages 42–51. AAAI Press, 2003.

[Erol *et al.*, 1996] K. Erol, J. A. Hendler, and D. S. Nau. Complexity results for HTN planning. *AMAI*, 18(1):69–93, 1996.

[Estrem and Krebsbach, 2012] S. J. Estrem and K. D. Krebsbach. AIRS: Anytime iterative refinement of a solution. In *FLAIRS 2012*, pages 26–31. AAAI Press, 2012.

[Fink and Yang, 1992] E. Fink and Q. Yang. Formalizing plan justifications. In *CSCSI 1992*, pages 9–14, 1992.

[Fox *et al.*, 2006] M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *ICAPS 2006*, pages 212–221. AAAI Press, 2006.

[Furcy and Koenig, 2005] D. Furcy and S. Koenig. Limited discrepancy beam search. In *IJCAI 2005*, pages 125–131. Morgan Kaufmann, 2005.

[Furcy, 2006] D. Furcy. ITSA*: Iterative tunneling search with A*. In *Proc. of the 2006 AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, pages 21–26. AAAI Press, 2006.

[Geffner and Bonet, 2013] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Gerevini and Serina, 2000] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *AIPS 2000*, pages 112–121. AAAI Press, 2000.

[Ghallab *et al.*, 2004] M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Gragera *et al.*, 2023] A. Gragera, R. Fuentetaja, Á. G. Olaya, and F. Fernández. A planning approach to repair domains with incomplete action effects. In *ICAPS 2023*, pages 153–161. AAAI, 2023.

[Hansen and Zhou, 2007] E. Hansen and R. Zhou. Anytime heuristic search. *JAIR*, 28:267–297, 2007.

[Harvey and Ginsberg, 1995] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *IJCAI 1995*, pages 607–615. Morgan Kaufmann, 1995.

[Haslum, 2012] P. Haslum. Incremental lower bounds for additive cost planning problems. In *ICAPS 2012*, pages 74–82. AAAI Press, 2012.

[Helmert and Röger, 2008] M. Helmert and G. Röger. How good is almost perfect? In *AAAI 2008*, pages 944–949. AAAI Press, 2008.

[Höller *et al.*, 2020] D. Höller, P. Bercher, G. Behnke, and S. Biundo. HTN plan repair via model transformation. In *KI 2020*, pages 88–101. Springer, 2020.

[Kambhampati and Kedar, 1994] S. Kambhampati and S. Kedar. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *AIJ*, 67(1):29–70, 1994.

[Katz and Sohrabi, 2020] M. Katz and S. Sohrabi. Reshaping diverse planning. In *AAAI 2020*, pages 9892–9899. AAAI Press, 2020.

[Kilani and Chrpa, 2013] A. Kilani and L. Chrpa. Post-planning plan optimization: Overview and challenges. In *ICAPS Workshop on Knowledge Engineering for Planning and ScheduLing (KEPS)*, 2013.

[Land and Doig, 1960] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[Lin and Bercher, 2021] S. Lin and P. Bercher. Change the world – how hard can that be? On the computational complexity of fixing planning models. In *IJCAI 2021*, pages 4152–4159. IJCAI, 2021.

[Lin and Bercher, 2023] S. Lin and P. Bercher. Was fixing this *Really* that hard? On the complexity of correcting HTN domains. In *AAAI 2023*, pages 12032–12040. AAAI Press, 2023.

[Lin *et al.*, 2023] S. Lin, A. Grastien, and P. Bercher. Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In *AAAI 2023*, pages 12022–12031. AAAI Press, 2023.

[Lin *et al.*, 2024] S. Lin, C. Olz, M. Helmert, and P. Bercher. On the computational complexity of plan verification, (bounded) plan-optimality verification, and bounded plan existence. In *AAAI 2024*, pages 20203–20211. AAAI Press, 2024.

[Lindner and Olz, 2022] F. Lindner and C. Olz. Step-by-step task plan explanations beyond causal links. In *RO-MAN 2022*, pages 45–51. IEEE, 2022.

[McCluskey *et al.*, 2002] T. L. McCluskey, N. E. Richardson, and R. M. Simpson. An interactive method for inducing operator descriptions. In *AIPS 2002*, pages 121–130. AAAI Press, 2002.

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

[Miller, 2019] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *AIJ*, 267:1–38, 2019.

[Muise and Lipovetzky, 2016] C. Muise and N. Lipovetzky. The first unsolvability international planning competition. http://unsolve-ipc.eng.unimelb.edu.au/, 2016.

[Muise *et al.*, 2011] C. Muise, S. A. McIlraith, and J. C. Beck. Monitoring the execution of partial-order plans via regression. In *IJCAI 2011*, pages 1975–1982. AAAI Press, 2011.

[Muise *et al.*, 2012] C. Muise, S. A. McIlraith, and J. C. Beck. Optimally relaxing partial-order plans with MaxSAT. In *ICAPS 2012*, pages 358–362. AAAI Press, 2012.

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

[Nakhost and Müller, 2010a] H. Nakhost and M. Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *ICAPS 2010*, pages 121–128. AAAI Press, 2010.

[Nakhost and Müller, 2010b] H. Nakhost and M. Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement – extended version. Technical Report 10-01, Department of Computing Science, University of Alberta, 2010.

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

[Nebel and Köhler, 1995] B. Nebel and J. Köhler. Plan reuse versus plan generation: a theoretical and empirical analysis. *AIJ*, 76(1-2):427–454, 1995.

[Nguyen *et al.*, 2012] T. A. Nguyen, M. B. Do, A. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *AIJ*, 190:1–31, 2012.

[Olz and Bercher, 2019] C. Olz and P. Bercher. Eliminating redundant actions in partially ordered plans – a complexity analysis. In *ICAPS 2019*, pages 310–319. AAAI Press, 2019.

[Pohl, 1970] I. Pohl. Heuristic search viewed as path finding in a graph. *AIJ*, 1(3):193–204, 1970.

[Ratner and Pohl, 1986] D. Ratner and I. Pohl. Joint and LPA*: Combination of approximation and search. In *AAAI 1986*, pages 173–177. Morgan Kaufmann, 1986.

[Richter *et al.*, 2010] S. Richter, J. Thayer, and W. Ruml. The joy of forgetting: Faster anytime search via restarting. In *ICAPS 2010*, pages 137–144. AAAI Press, 2010.

[Salerno *et al.*, 2023] M. Salerno, R. Fuentetaja, and J. Seipp. Eliminating redundant actions from plans using classical planning. In *KR 2023*, pages 774–778. IJCAI, 2023.

[Say *et al.*, 2016] B. Say, A. A. Cire, and J. C. Beck. Mathematical programming models for optimizing partial-order plan flexibility. In *ECAI 2016*, pages 1044–1052. IOS Press, 2016.

[Scala and Torasso, 2015] E. Scala and P. Torasso. Deordering and numeric macro actions for plan repair. In *IJCAI 2015*, pages 1673–1681. AAAI Press, 2015.

[Seipp and Helmert, 2013] J. Seipp and M. Helmert. Counterexample-guided cartesian abstraction refinement. In *ICAPS 2013*, pages 347–351. AAAI Press, 2013.

[Shaw, 1998] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP 1998*, pages 417–431. Springer, 1998.

[Siddiqui and Haslum, 2012] F. H. Siddiqui and P. Haslum. Block-structured plan deordering. In *AJCAI 2012*, pages 803–814. Springer, 2012.

[Siddiqui and Haslum, 2013] F. H. Siddiqui and P. Haslum. Plan quality optimisation via block decomposition. In *IJCAI 2013*, pages 2387–2393. AAAI Press, 2013.

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

[Sreedharan *et al.*, 2021] S. Sreedharan, T. Chakraborti, and S. Kambhampati. Foundations of explanations as model reconciliation. *AIJ*, 301:103558, 2021.

[Sreedharan *et al.*, 2022] S. Sreedharan, P. Bercher, and S. Kambhampati. On the computational complexity of model reconciliations. In *IJCAI 2022*, pages 4657–4664. IJCAI, 2022.

[Sreedharan *et al.*, 2023] S. Sreedharan, C. Muise, and S. Kambhampati. Generalizing action justification and causal links to policies. In *ICAPS 2023*, pages 417–426. AAAI Press, 2023.

[Srivastava *et al.*, 2007] B. Srivastava, T. A. Nguyen, A. Gerevini, S. Kambhampati, M. B. Do, and I. Serina. Domain independent approaches for finding diverse plans. In *IJCAI 2007*, pages 2016–2022. AAAI Press, 2007.

[Stern *et al.*, 2011] R. Stern, R. Puzis, and A. Felner. Potential-search: A bounded-cost search algorithm. In *ICAPS 2011*, pages 234–241. AAAI Press, 2011.

[Thayer *et al.*, 2012] J. Thayer, R. Stern, A. Felner, and W. Ruml. Faster bounded-cost search using inadmissible heuristics. In *ICAPS 2012*, pages 270–278. AAAI Press, 2012.

[Waters *et al.*, 2018] M. Waters, B. Nebel, L. Padgham, and S. Sardina. Plan relaxation via action debinding and deordering. In *ICAPS 2018*, pages 278–287. AAAI Press, 2018.

[Waters *et al.*, 2020] M. Waters, L. Padgham, and S. Sardina. Optimising partial-order plans via action reinstantiation. In *IJCAI 2020*, pages 4143–4151. IJCAI, 2020.

[Westerberg and Levine, 2001] C. H. Westerberg and J. Levine. Optimising plans using genetic programming. In *ECP 2001*, pages 272–274. AAAI Press, 2001.

[Zhou and Hansen, 2005] R. Zhou and E. A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *ICAPS 2005*, pages 90–98. AAAI Press, 2005.