A Survey on Plan Optimization

Pascal Bercher\textsuperscript{1}, Patrik Haslum\textsuperscript{1} and Christian Muise\textsuperscript{2}

\textsuperscript{1}School of Computing, The Australian National University
\textsuperscript{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 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* [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* [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; Babil 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 \( \Sigma_2^P \)-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 explanation [Chakraborti et al., 2020]. Whereas some plan explanation approaches (specifically those that aim at explaining the necessity or purpose of an action in a plan1) do so by simply “explaining” a chain of actions from said action to a goal, sometimes by following a chain of causal links [Bercher et al., 2011; Thayer et al., 2012]. Bounded suboptimal search has 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; Babil 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 \( \Sigma_2^P \)-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 explanation [Chakraborti et al., 2020]. Whereas some plan explanation approaches (specifically those that aim at explaining the necessity or purpose of an action in a plan1) do so by simply “explaining” a chain of actions from said action to a goal, sometimes by following a chain of causal links [Bercher et al., 2011; Thayer et al., 2012]. Bounded suboptimal search has 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; Babil 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.

1For 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 \( \langle F, A, s_1, g \rangle \), 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), \( s_1 \subseteq 2^F \) 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,

\[
\hat{s}' = (s \setminus del(a)) \cup add(a).
\]

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 \( \hat{a} = a_1, \ldots, a_n \) is called a solution (or plan) if it is applicable in the initial state \( s_1 \) 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 \( \text{pickup}(?b) \) picks up a block \( ?b \) (we use question marks to denote variables) from the table into a gripper, whereas \( \text{unstack}(?b_1, ?b_2) \) removes a clear (top-most) block \( ?b_1 \) from another block \( ?b_2 \). Actions \( \text{stack}(?b_1, ?b_2) \) and \( \text{putdown}(?b_1) \) 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 \( a, b, c, \ldots \) 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 composition 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].

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

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

...
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 suboptimal 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 Helwert, 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 $\epsilon$ less than the plan’s cost (where the required value of $\epsilon$ 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


