Received: 2 June 2022 | Accepted: 13 June 2022

DOI: 10.1002/aaai.12073

ARTICLE

Hierarchical planning and reasoning about partially ordered plans—From theory to practice

Pascal Bercher

School of Computing, College of Engineering, Computing and Cybernetics. The Australian National University, Canberra, Australia

Correspondence
Pascal Bercher, School of Computing, College of Engineering, Computing and Cybernetics, The Australian National University, Canberra, Australia.
Email: pascal.bercher@anu.edu.au

*The respective invited talk is available at https://slideslive.com/38952027/hierarchical-planning-and-reasoning-about-partially-ordered-plans-from-theory-to-practice

Abstract

This invited paper (part of the *New Faculty Highlights Invited Speaker Program* of AAAI'21*) surveys my work done until today. The reviewed work focuses on hierarchical task network (HTN) planning as well as on partial order causal link (POCL) planning. Lines of research include theoretical investigations (mostly computational complexity analyses), heuristic search, as well as the practical application of the technology for planning-based assistants, which support a human user in carrying out various tasks.

INTRODUCTION

AI planning is a general problem solving technique that can be deployed for autonomously solving a wide range of different problems (Ghallab, Nau, and Traverso 2004, 2016). This article surveys most of my work done in that field, which encompasses various kinds planning paradigms, as well as research areas that span from theory to practice.

Involved problem classes

Classical planning (Ghallab, Nau, and Traverso 2004) is concerned with the evolution of states. A state is simply a finite set of facts, that is, properties currently being true in the world that's being planned for. Assume you want to plan the setup of a complex home theater, where different hifi devices had to be connected by different cables, adapters, and the like. Each available object is represented by a constant, like HDMI-CABLE$_1$, HDMI-CABLE$_2$, or AMPLIFIER. Relying on a restricted first-order logic, predicates are used to express properties. For example, CONNECTED(HDMI-CABLE$_2$, AMPLIFIER) describes that the cable HDMI-CABLE$_2$ is connected with the amplifier. States are then sets of such predicates thereby describing all properties, which are currently true in the respective state. Classical planning problems are about finding the right sequence of actions (selected from a set of available actions) to transform the initial state into one in which all desired properties hold. In the example application, actions may be plugging in or plugging out specific cables into specific ports of the available devices. Not every action is applicable in every state, so for example neither can an HDMI cable be plugged into a port that already has another cable inside, nor into a (free) port of a wrong kind, such as cinch. To express this, actions have *preconditions*, which are sets of predicates that must be true (or false if they are negated) in the state to which the respective action is applied. Their *effects*, again sets of possibly negated predicates, specify how states are changed due to action application.

In the scenario above, finding a plan that solves the problem (in this case, the successful setup of the system, so that every device gets the required signals via connecting them to each other using the right cables) is relatively easy. The reason is that there is not a reason to ever apply any action twice. Once a cable is plugged in somewhere it does not have to be plugged out again (unless during plan execution something can go wrong, or new information can be obtained, but in this work, we assume full observation and a deterministic world), which means that we only need to find the right selection of actions. Even the order in which actions are applied is not important for the success of the plan in our example, although different orders might be differently intuitive when a user is being instructed to execute them (this is also a research question addressed later). What makes planning hard in general is that actions may have to be applied multiple times (to different world states) and that it is not clear which or how many actions have to be applied when to solve a problem. To solve problems efficiently, heuristic search is often applied (one of the areas to which contributions are surveyed later). Another way of coping with the complexity of planning problems is using a different kind of planning paradigm, *hierarchical planning*.

In hierarchical planning (Ghallab, Nau, and Traverso 2004; Bercher, Alford, and Höller 2019), we still use actions as described above, but in this context, they are called *primitive tasks*. They are called primitive to emphasize that they are the most “atomic” or “simple” form there is, particularly that they can be executed in the world. That is because hierarchical planning now also features a second kind of action/task, which the literature refers to as either abstract, complex, or compound. We use the latter terminology, that is, *compound tasks*. They are called that way because they are literally “compound” placeholders for other tasks, which may be either compound again or primitive. So in a sense each compound task can be regarded a macro as it will be substituted by some other tasks, predefined in the model. This defines a possibly cyclic/recursive hierarchy among all tasks. For example, Figure 1 shows how a (recursive) task hierarchy in the domain sketched above.

The figure shows a compound task at the top, $connect(?d_{so}, ?d_{si})$, representing the abstract activity to connect the source device $?d_{so}$ with a sink device $?d_{si}$. The questionmarks are used to indicate variables in contrast to concrete constants such as AMPLIFIER. There are two possibilities how these two devices could be connected. Either directly by plugging in a cable right in between them—this is depicted on the left, where two primitive tasks (i.e., actions) are used to plug in the two ends of the respective cable. An alternative, shown on the right, is to establish this connection via another

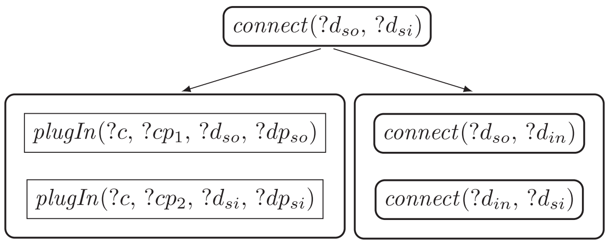

$connect(?d_{so}, ?d_{si})$
$plugIn(?c, ?cp_1, ?d_{so}, ?dp_{so})$
$plugIn(?c, ?cp_2, ?d_{si}, ?dp_{si})$
$connect(?d_{so}, ?d_{in})$
$connect(?d_{in}, ?d_{si})$

FIGURE 1 A compound task at the top with two decomposition methods. The method on the left contains two primitive tasks (their preconditions and effects have been omitted), and the one on the right contains two compound tasks. (Bercher, Alford, and Höller 2019, Figure 1). Copyright: International Joint Conferences on Artificial Intelligence, Proceedings 2019, IJCAI.ORG. All rights reserved. Reproduced by permission

device in between, called intermediate device $?d_{in}$ in that figure. The two involved tasks, $connect(?d_{so}, ?d_{in})$ and $connect(?d_{in}, ?d_{si})$ are again compound because there could be yet other devices in the middle. This model makes sense because we could rely on combinations of adapters and cables (also regarded devices), or because of other devices in between. For example, we could provide a television with a video signal from a blu-ray player, but by going through an audio/video amplifier in between.

We thus see that hierarchical problem modeling gives more control on plans as this task hierarchy must be adhered during plan generation—similar to formal grammars (Höller et al. 2014). It thus provides a means to domain modelers to model expert knowledge on how various tasks may be solved. Seemingly paradoxically, this control over possible plans also makes the problem computationally more complex and allows the specification of problems that could not be modeled without a task hierarchy. Therefore, heuristic search is also deployed in this paradigm to speed up the solving process or to even find a solution at all. Note that on top of advice and/or higher expressivity, two further noteworthy motivations of using a hierarchical planning paradigm are to simply have another way of modeling problems (as in some applications, it might be easier to encode a problem by means of “first do X, then Y, then Z” instructions rather than by specifying all state-based preconditions and effects), and to exploit the hierarchy for conveying solutions on higher levels of abstraction (as we will later see as well).

Investigated research questions and outline

My main contributions surveyed in this article are concerned with both planning paradigms, classical and hierarchical planning, though most results are achieved in the latter.

Most results are concerned with theoretical investigations. In the realm of classical, nonhierarchical problems, investigated questions are concerned with *reasoning about partially ordered plans*, for example whether actions can be removed without making the plan not executable anymore, or whether a given nonsolution plan can be turned into a solution. In the realm of hierarchical planning, questions mostly evolve around the so-called plan existence problem, that is, the question whether a given planning problem has a solution or not. These investigations aim at determining the computational complexity of these (and related) decision problems. The motivation behind these investigations is manifold. Knowing the computational complexity of these problems provides us with a deeper problem understanding; an idea for the selection or design of algorithms to tackle the respective problem as the complexity directly impacts algorithm time and space requirements; possible problem reductions, which enable the use of well-established problem solvers for different kinds of problems, such as SAT (satisfiability) or ILP (Integer Linear Program) solvers for NP-complete problems.

Still theoretical foundations but of a more applied nature are my contributions to solving these problems efficiently. In particular, in the area of hierarchical planning, I am concerned with algorithm and heuristic design that enables to solve planning problems quickly via heuristic search.

Finally, my last line of research is the development and deployment of planning-based technology that contributes towards the successful application of planning in practice. On top of plan generation via heuristic search, such technology involves plan recognition (*based on an observed partial plan, which goals does the agent pursue?*), plan repair (*how can a failed plan be fixed such that the problem can still be solved?*), plan linearization (*in which order should actions best be instructed to a human user?*), and plan explanation (*why is a certain action in the plan?*). Many of these technologies have been deployed in two implemented systems: We implemented an assistance system that supports a human user in the task of assembling a home theater by providing a detailed sequence of step-by-step instructions. In cooperation with Robert Bosch GmbH, we created a follow-up system that assists handymen in the execution of do-it-yourself (DIY) projects.

The article is structured as follows: The next section covers the investigated problem classes in a bit more detail, and lays out some of my concrete contributions. The rest of the article follows the outline from above: Section 3 covers my research on theoretical foundations, Section 4 covers heuristic search, and Section 5 gives some overview about the two planning-based assistants.

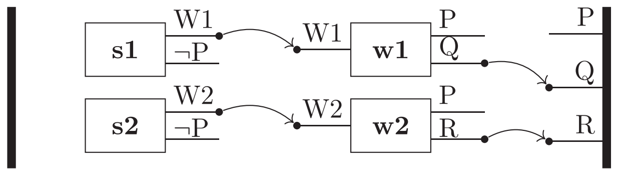


FIGURE 2 Example POCL plan that is not yet a solution due to a missing causal link (Bercher and Olz 2020, Figure 1)

PROBLEM FORMALIZATION

In the Introduction, we already provided the core concepts of the problem classes used throughout this article. Here, we provide slightly more technical definitions and point towards specific contributions related to the problem classes.

Classical planning and POCL plans

We already covered classical planning problems, which require to find a *plan*—a sequence of actions that turns the initial state into a state in which all desired goal properties hold. Many of my contributions are, however, concerned with partial order causal link (POCL) planning (Weld 1994), which evolves around partially ordered plans. Here, plans are not totally ordered action sequences, but only partially ordered as for example shown in Figure 2. POCL plans have several applications, but are mostly used today for plan optimization techniques (Siddiqui and Haslum 2015; Muise, Beck, and McIlraith 2016; Waters et al. 2018; Waters, Padgham, and Sardina 2020).

POCL plans do not only differ from action sequences in having only a partial order defined amongst actions, but they are also more complicated in the way executability is checked. Checking this for action sequences is trivial as we have the initial state to check executability of the first action, then we can easily compute the successor state, check executability of the next action, and so on. In case of a plan whose actions are only partially ordered, we do not have states available and thus need another means to check executability. This is done via *causal links*.

A POCL plan is considered a solution when every precondition is supported by a causal link and there do not exist any causal threats. A causal threat is the situation where an action that deletes a fact f (the threatening action) could be ordered in between two other actions that share a causal link protecting f. If such a situation occurs, it cannot be guaranteed that the respective causal link "does its job," which is ensuring that the precondition f will hold when it is required. Such a situation can be resolved by moving the threatening action before the link's producer, or moving it behind the link's consumer. POCL solution plans have the property that every linearization (i.e., every total order that is compatible with the ordering constraints) is an executable action sequence that satisfies all goals. Such plans can, therefore, encode up to an exponential number of linearizations in just a single plan.

When considering the example in Figure 2, we see that it is not yet a solution since one causal link is still missing. Adding such a causal link will then, however, raise a causal threat that needs to get resolved. For example, if the link protecting P of the goal is added using $w1$ as producer, then $s2$ (which is currently not ordered w.r.t. $w1$) is threatening that causal link and must thus be ordered before $w1$ to resolve that causal threat. Afterwards, the respective plan is a solution. This example shows that causal links are overly restrictive, since in this case, this causal threat is actually not an issue: Even though $s2$ threatens the link produced by $w1$, the action $w2$ would repair it again. The causal threat criterion still forces us to add an ordering constraint thus reducing the set of linearizations that can be obtained.

HTN planning

Hierarchical planning centers around the idea of problem decomposition. There are many different variants, some decompose tasks, others decompose goals (facts)—an overview is given in a recent survey of the field (Bercher, Alford, and Höller 2019). In this work, we focus on the decomposition of tasks, the most influential formalization of which is *hierarchical task network (HTN) planning* by Erol, Hendler, and Nau (1996).

The high-level introduction to hierarchical planning given in Introduction was to HTN planning—so we know already the most basic concepts. A bit more formally, an HTN planning problem consists of: A set of primitive tasks (those are the actions as we have seen them in classical planning and graphically illustrated in Figure 2); a set of compound tasks and for each a set of decomposition methods, each stating by which predefined plan its compound task may be replaced by—hence the name "compound" as each such task can be interpreted as the combination of all tasks in either of its methods; and an initial set of compound or primitive tasks.

The goal of an HTN planning problem is to refine all initially given compound tasks into a primitive executable plan. So the hardness arises from choosing for each compound task in a current plan such a method that will eventually contribute towards finding a primitive plan whose actions allow the plan to be executed. Formally, the respective primitive plan must possess an executable action linearization.

An important contribution to the field of HTN planning was a novel formalization of HTN planning problems (Geier and Bercher 2011), which became another accepted standard (Bercher, Alford, and Höller 2019). Apart from its simplicity, a major plus compared to the original by Erol, Hendler, and Nau (1996) is that it makes the solution criteria explicit that clearly and declaratively state under which circumstances a plan is regarded a solution to an HTN problem. Simplicity of the problem was achieved by removing some constraints that are available in the original formalism, and also by choosing a *propositional* formalism. In Introduction and as illustrated in Figure 1, we were explaining that states and thus action preconditions and effects are defined based on a first-order logic, where facts are formalized as predicates. Formalizations do, however, get significantly simplified if just propositional symbols are allowed, such as P, Q, and R, as used in our introduction to POCL plans.

Whereas formalization, proofs, and algorithms can be described (and understood) much more easily in a propositional formalism than one based on a first-order logic, *actual problems* (e.g., for a benchmark set) are still described in terms of the latter. For this we developed HDDL, the hierarchical domain description language (Höller et al. 2020). It was also used as the official description language for the International Planning Competition (IPC) on Hierarchical Planning that we carried out in 2020 (see ipc.hierarchical-task.net). Problems described in such a way can then be turned into a propositional one via grounding, where—conceptually—each variable is replaced by an appropriate constant. The actual technique does, however, perform various reachability analyses that eliminate groundings that can provably never contribute towards finding a solution (Behnke et al. 2020).

TIHTN planning

In HTN planning, the only means to change a plan is by decomposing one of its compound tasks by replacing it by one of its methods, specifically by the plan of the respective method. This requires careful domain modeling as the HTN model must be designed in a way that all desired solutions can be found by strictly adhering the task hierarchy.

For example, when going back to the example shown in Figure 1, it becomes apparent that without the method on the right, we could only connect two devices with exactly one cable in between. This, for example, precludes solutions where multiple cables have to be connected via adapters. So designing an HTN model that allows to generate all intended solutions is not exactly trivial.

In some cases, it might be easier to write a model that is only partially hierarchical (Kambhampati, Mali, and Srivastava 1998) in the sense that certain parts of desired solution plans do not have to be found by strictly adhering the task hierarchy, but instead by arbitrary (primitive) task insertion, just as in classical planning where there is no restriction on "allowed" action sequences other than allowing only executable ones (a requirement still present in HTN planning).

Another major contribution of mine is thus the formalization of a hierarchical problem class where such action insertions are permitted. The resulting framework is called *HTN planning with task insertion—or (TIHTN) planning* (Geier and Bercher 2011; Alford, Bercher, and Aha 2015b).

THEORETICAL INVESTIGATIONS

Much of my work is concerned with the theoretical analysis of planning problems. Most notably, this involves the investigation of the computational complexity of important decision problems, such as the *plan existence problem* ("Does the current problem have a solution?") or about *change requests* ("Can we perform change X and still maintain property Y?"), which have been done for all problem classes.

Complexities for POCL plan existence

The question we are interested in is to know whether a given POCL plan can be turned into a solution by adding additional actions, causal links, and ordering constraints.

Standard classical planning problems are well-known to be PSPACE-complete (Bylander 1994). In such problems, we check plan existence from an initial state rather than from some POCL plan. But having to deal with some additional initial actions, causal links, and orderings does not make this harder, so the POCL plan existence problem is also PSPACE-complete (Bercher 2021). The computational complexity of special cases was also investigated, for example, when all actions have only positive effects but no negative ones. Such special cases are of interest because they can be created, for example, by just ignoring all negative effects, although they are in the model (this is called delete-relaxation). This is desirable if the respective relaxation/special case is of lower computational complexity than the original problem as heuristics can them as exploit them due having a lower runtime. This is the case as our investigations show complexities of poly-time or NP, depending on the chosen relaxation (Bercher 2021).

Complexities for HTN plan existence

Erol, Hendler, and Nau (1996) proved that HTN planning is expressive enough to encode undecidable problems. They did so by showing that the (undecidable) language intersection problem of two context-free grammars can be encoded by an HTN planning problem. We reproduced their result for our formalism thus showing that their result still holds despite our simplifications (Geier and Bercher 2011).

The comprehensive complexity study by Erol, Hendler, and Nau (1996) does not just show the hardness of the general case, but also several important special cases, namely, *acyclic* problems (shown to be decidable)—these are problems where the task hierarchy does not allow recursion, so no compound task can introduce itself again. *Totally ordered* problems (shown to be in EXPTIME)—these are problems where the initially given tasks as well as those in all methods are totally ordered task sequences. *Regular* problems (shown to be PSPACE-complete)—these are, similar to right-regular formal grammars, problems where the plans in decomposition methods may only have at most one compound task, which then has to be the last one. The results above are for a propositional setting. In case of first-order formalization with variables complexities are one exponential factor harder (Erol, Hendler, and Nau 1996).

Tight bounds for all these special cases (and their combinations) were shown by Alford, Bercher, and Aha (2015a). Tight bounds were also shown for *tail-recursive* problems, a novel generalization of regular HTN problems, identified by Alford et al. (2012). Tail-recursive problems are a generalization of regular problems and the computationally most expensive (and thus most expressive) special case currently known to be decidable. Due to its decidability, problems adhering these syntactical restrictions can be compiled into classical planning problems (Alford et al. 2016).

Directly motivated by the design of heuristics, we also investigated the complexity of delete-relaxed HTN problems with empty ordering constraints. We showed this problem to be NP-complete (Höller, Bercher, and Behnke 2020), which is in line with results by Alford et al. (2014) who showed it for delete-relaxed HTN problems (i.e., without the relaxation of ignoring all ordering constraints).

Extensions

We also conducted complexity investigations for the plan existence problem of two extensions of the HTN formalism introduced.

One is referred to as *hybrid planning* (Kambhampati, Mali, and Srivastava 1998; Bercher et al. 2016; Bercher, Lin, and Alford 2022), which combines HTN planning with POCL planning. In a nutshell, compound tasks specify preconditions and effects, which are used to pose constraints on decompositions methods, which now may also contain causal links. These constraints are used to make sure that methods "implement" the modeler's intent of the compound task. We showed that these constraints do not change the formalism's expressiveness in terms of plan existence complexity (Bercher et al. 2016; Bercher, Lin, and Alford 2022).

The second is an extension of deterministic actions to nondeterministic ones (Chen and Bercher 2021, 2022). In this setting, we are usually interested in generating a plan that can be executed no matter the nondeterministic outcome. We proposed various ways on when actions are selected for execution from primitive solution plans: offline before plan execution or online after we witnessed the effects of the action executed last. While the latter is more flexible, it comes with higher computational costs.

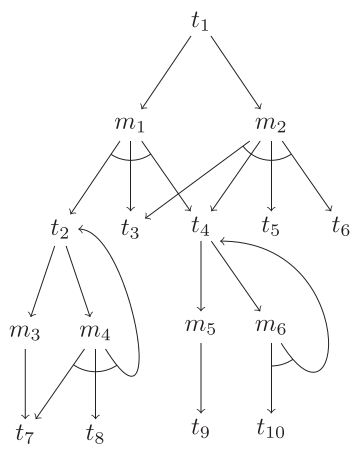


FIGURE 3 A task decomposition graph (TDG) of an HTN planning problem. The primitive task t_3 is a landmark, as well as the compound task t_4. Since the only way to refine t_4 into a primitive task network is using m_5, this method and its primitive task t_9 are also landmarks.

Complexities for landmarks and compound tasks' implications on states

Landmarks and TDGs.

In recent work, we investigated the complexity of deciding whether a given fact, task, or method is a *landmark* (Höller and Bercher 2021). Landmarks are an important concept in planning as landmark information may be exploited both by algorithms and heuristics. A landmark fact is a fact that has to be true at some point during plan execution for any solution. Likewise, a primitive task landmark is an action that is contained in any solution. A compound task landmark is a compound task that is eventually decomposed for finding a solution. Likewise, a method is a landmark if it has to be applied in order to find a solution. Checking whether a fact, task, or method is a landmark was shown to be in the co-class of the corresponding plan existence problem. This is because one essentially has to check whether a slightly modified problem does *not* have a solution—which is the co-version of a problem. To illustrate the concept of landmarks, consider the so-called task decomposition graph (TDG) that is illustrated in Figure 3.

The TDG is an AND/OR graph and a canonical data structure in HTN planning that I formalized for the exploitation of landmarks (Elkawkagy et al. 2012). But since a TDG is a canonical structure representing a problem's task hierarchy, it may also serve other purposes such as the computation of nonlandmark-based heuristics (Bercher et al. 2017) or for grounding an HTN model (Behnke et al. 2020). The TDG is the extension of the task decomposition tree (TDT) (Elkawkagy, Schattenberg, and Biundo 2010) that can also represent task recursion without becoming infinite. That is, in the absence of recursion, the TDGs and TDTs coincide, though Elkawkagy, Schattenberg, and Biundo (2010) did not provide a formal definition for the TDT yet. TDGs contain two kinds of nodes: task nodes for the primitive and compound tasks as well as method nodes representing the decomposition methods. Figure 3 uses connectors for method nodes to indicate that all tasks within a method have to be dealt with (i.e., applied to a state or decomposed further). TDGs are finite structures since recursion just results in the addition of another edge to an existing node (thus making the graph cyclic), it will not create another instance of said task as it would during planning. TDGs can be built starting from the initial tasks (here just t_1) until all tasks that are reachable via methods have been included. While some landmarks can be inferred and computed efficiently (in polynomial time) as illustrated in the example TDG, *in general*, this deduction is as hard as solving the problem itself. This should not be hard to see: for example, task t_6 becomes a landmark only if method m_1 will not be successful for turning t_1 into a primitive executable plan. This, however, resembles solving the planning problem where m_2 was removed and checking whether it has a solution.

Preconditions and effects of compound tasks.

Somehow related to the concept of landmarks is some of our more recent work on preconditions and effects of compound tasks (Olz, Biundo, and Bercher 2021). Recall that the purpose of compound tasks is just to get replaced by the plan of the chosen decomposition method. So in contrast to their primitive counterparts, they do not have any preconditions or effects. Yet sometimes we can infer some state information anyway, for example, when each method of a task always adds a fact f, but never deletes in, then we know that f can be regarded an "inferred effect" of said task. Generally, such inferred effects are state properties that will hold true after the last task resulting from the decomposition was executed, no matter what method was chosen. (preconditions were defined similarly.) The results turned out to be the same as for landmarks: Determining whether a fact is precondition or an effect is as hard as solving the respective planning problem itself (Olz, Biundo, and Bercher 2021). However, we also identified a problem relaxation that allows computing some of these preconditions and effects in polynomial time.

Complexities for TIHTN plan existence

When we developed the TIHTN formalism, we were primarily interested in the question whether the framework maintains its undecidability when we are allowed to insert tasks. In short: no, TIHTN planning is decidable (Geier and Bercher 2011). To be more precise, we showed that due to task insertion, we will never have to use cyclic decompositions because we can instead just rely on task insertion.

To illustrate, consider the planning model visualized in Figure 3 once more. Assume m_2 is a landmark, so it will be used. Further assume that for making tasks t_5 and t_6 executable, we will have to introduce task t_{10}. This means that—when using HTN planning—we will first decompose t_4 using m_6 into t_{10} and t_4 again, and then, we apply m_5 to turn t_4 into the primitive task t_9. We thus *had* to rely on the cyclic model to generate a solution. In TITHN planning that is not required, we could have just used m_5 directly and added the required t_{10} using task insertion instead.

Based on this observation, we conducted a comprehensive complexity analysis of all the special cases for HTN planning outlined before (Alford, Bercher, and Aha 2015b).

Complexities of change requests

POCL planning

Each POCL solution is a compact representation of an up to exponentially large number of totally ordered plans—each of which is a solution to the underlying classical planning problem. Such solutions do not need to be locally optimal, however, which means that one or more actions could potentially be removed without making it invalid. We thus asked the question how hard it is to determine whether we can delete a *single* given action from a solution plan and making it a solution again by adding missing causal links and ordering constraints (Olz and Bercher 2019). It turns out that this task is already NP-complete.

In a follow-up work, we investigated the problem of optimizing a plan's makespan (Bercher and Olz 2020), that is, its execution time when exploiting parallelism. For example, the POCL plan depicted in Figure 2 (even when the missing link and ordering constraint would be added) has a makespan of 2, since $s1$ and $s2$ can be executed in parallel as well as $w1$ and $w2$. The concrete modifications we looked at is *deordering*, which means removing ordering constraints. Our main result is that deciding whether there exists a deordering of a given plan resulting in a makespan of k is NP-complete. If changing causal links is not allowed, then deordering becomes decidable in polynomial time.

HTN planning

Also in the context of HTN planning, we were looking into plan optimizations. Motivated by user requests, we investigated the computational complexity of checking whether a certain change request to a given HTN solution plan can be performed (Behnke et al. 2016). The main result is that basically all changes (like adding or deleting an action or an ordering etc.) are NP-complete. This result is actually not surprising since Behnke, Höller, and Biundo (2015) showed that just the verification whether a given task network is a solution is already NP-complete (and we need to check that the resulting plans after the change is again a solution). If however k changes are allowed (where k is an argument to the decision problem encoded binary), then the problems turn NEXPTIME-complete.

A closely related question was investigated recently (Barták et al. 2021), where we are again given a plan as an input. This time, however, it is not a solution already—it is just *supposed* to be a solution, but plan verification tells that it is not, for example, because the plan cannot be produced by the task hierarchy alone. So we were interested in deleting the minimal number of actions from the plan so that the resulting (maximally long) plan is a solution. This was shown to be NP-complete.

We also investigated the opposite case (Lin and Bercher 2021). Again we are provided with a plan that is not a solution, but should. But instead of fixing the problem by changing the plan, we change the *model*. That is, allowed operations are adding and removing actions to decomposition methods. Despite restricting to a total-order setting, the problems turns out to be NP-complete. We also investigated the similar question for changing actions' preconditions and effects. The main result here is that checking whether k such changes exist is polytime-decidable if we are allowed to change add effects, and otherwise NP-complete.

HEURISTIC SEARCH IN HTN PLANNING

As explained in more detail in our survey on HTN planning (Bercher, Alford, and Höller 2019), there are various standard algorithms for solving HTN planning problems. Many of them are implemented within the PANDA framework (Höller et al. 2021).

Our early work on heuristics is implemented in a *plan space-based hybrid planner* (Bercher, Keen, and Biundo 2014), which is essentially a standard POCL planner extended so that it can deal with task decomposition in HTN planning. The planner is not maintained anymore, so all more novel heuristics from 2018 and later are implemented in a *state-based progression planner* (Höller et al. 2018, 2020) in the spirit of the famous SHOP2 system (Nau et al. 2003). For the latter, we also proposed novel variants of standard progression search that eliminate redundant choice points during search thus producing fewer redundant search nodes.

TDG- and landmark-based heuristics

Early work on domain-independent heuristics for HTN planning centered around the computation of landmarks (Elkawkagy et al. 2012; Bercher, Keen, and Biundo 2014). This work focused on constructing a TDG (as the one provided in Figure 3) and computed landmarks via taking the intersection of all decomposition methods that belong to the same compound task. This method was however only able to identify *task landmarks* (i.e., method and fact landmarks were not computed yet). Our most recent work subsumes all these approaches, that is, even restricted to task landmarks it finds a (not necessarily strict) superset of landmarks although it only deploys a poly-time landmark procedure (Höller and Bercher 2021). This procedure transfers a landmark extraction technique from classical planning to the HTN setting—also by computing an AND/OR graph. We refer to our paper for details. So far we only deployed a simple heuristic based on these landmarks defined as the number of landmarks, which are not yet fulfilled by the current search node.

Also based on TDGs is a heuristic that (admissibly) estimates for each task the size of the primitive plan that can be achieved from it (Bercher et al. 2017). This is a simple min/sum equation that sums over all tasks in a decomposition method and chooses the cheapest method per compound task. Even in the presence of loops, these estimates can be computed in polynomial time. In our example from Figure 3, the heuristic for t_1 would be 3, based on the (cheapest) method m_1 producing a relaxed plan consisting of t_7, t_3, and t_9. (Choosing m_2 would have resulted in an estimate of 4 using t_3, t_9, t_5, and t_6.) Task interaction is captured only in a limited way. Reconsider our example, where the only way to make t_5 and t_6 executable is by adding t_{10}. Still, the heuristic is not able to detect this. Estimates are, however, improved upon search progress by rebuilding the TDG during search thus ruling out decomposition methods (and thus parts of the tree) that are not available anymore.

Compilation-based heuristics

Encoding into classical problems

We can observe that there are only a few heuristics available for HTN planning, yet *many* for state-based classical planning. To exploit this situation, we proposed a compilation that encodes each HTN search node into a (relaxed) classical planning problem, so that we can use an existing classical heuristic as estimate instead (Höller et al. 2018, 2020). From a more abstract viewpoint, we encode the TDG of the current search node as classical planning problem. That is, we encode that a primitive executable refinement of the current search node must be found, while task repetition does not matter, that is, a compound task is regarded decomposed if each subtask is executed at least once. Note that here we have *two* relaxations: one by the encoding into the classical problem and a further one done by the deployed classical heuristic. This is in contrast to approaches (I was involved in) that compile the entire HTN problem in a solution-preserving way into a (sequence of) classical planning problem(s) (Alford et al. 2016; Behnke et al. 2022).

Encoding into (integer) linear programs

Section 3.2 mentioned that we proved NP-completeness of delete- and ordering-free HTN problems (Höller, Bercher, and Behnke 2020). In the same work, we exploit this by encoding this problem into an ILP, one of the best-known frameworks for efficiently solving NP-complete problems. While this heuristic is also empirically the most-informed one (resulting into the smallest search space), it is not the best-performing one due to higher runtime costs.

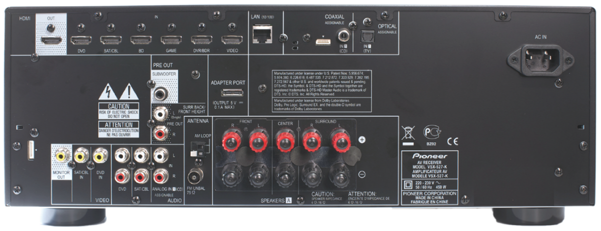


FIGURE 4 Back panels of the modeled amplifier (Bercher et al. 2014, Figure 1)

PLANNING-BASED ASSISTANCE SYSTEMS

Motivated by our research on *Companion technology* (Biundo et al. 2016)—a technology enabling cognitive technical systems to behave as companions that provide their functionality in a smart and adaptive way to their human users—we conducted work on integrating various planning capabilities with dialog and interaction management, as well as a central knowledge base for the provision of planning-based assistance (Biundo et al. 2011; Bercher et al. 2017).

The main ingredient for flexible assistance is to rely on a planning model in the first place—that way plans can be generated based on the current situation and thus made highly adaptive. These plans are then provided to the user in a step-by-step fashion so that he or she just has to follow the presented instructions. Each instruction is in turn generated automatically from the respective action. Due to relying on planning models, *plan repair* may be applied in case nonanticipated execution errors occur, and *plan explanation* may be deployed to answer questions about the plan that may come up at runtime. Two systems have been implemented following this approach.

Assembly assistant

As a proof of concept of how the interaction of various planning technologies can provide advanced assistance, we implemented a system that supports in the task of setting up a complex home theater (Bercher et al. 2014, 2015, 2018). In a fully general hifi assembly assistant, users would be able to select the available hardware devices and cables (only the ports are required, so image recognition could do the job), but in our running system, we only modeled one specific scenario. Here, the home theater consists of the four hifi devices illustrated in Figures 4 and 5.

In order to successfully use this system, the television must receive the video signals of the satellite receiver and the blu-ray player, and the amplifier must receive their audio signals (since the speakers are connected to this device). As can be seen due to the very high number of different ports, this might be a quite challenging task, in particular for nonexpert users.

This task was modeled as a planning problem and solved by our hybrid planner that combines HTN planning with POCL planning (Bercher, Keen, and Biundo 2014; Bercher et al. 2017).

Slightly simplified, required actions have the form illustrated in Figure 1, that is, $plugIn(?c, ?cp_1, ?d_{so}, ?dp_{so})$. The variables indicate the objects that the action refers to, that is, the cable $?c$, its port $?cp_1$, the device d_{so}, and its port $?dp_{so}$. This information is enough to render, automatically at runtime, a graphical instruction complemented by a textual representation in text form. For example, a text can be provided based on a schema like this: *“Please plug the [X] port of the [Y] cable into the [X] port of [Z].”* We used different variants of this so that the user does not get frustrated of always hearing the same. Since we kept images of all involved hardware, the dialog management was able to illustrate these instructions appropriately as shown in Figure 6.

At any point in time, the user is able to report the currently used cable as being broken. If that happens, the respective cable was marked as broken/unusable and a new plan was found and presented user based on plan repair (Bercher et al. 2014). The new plan incorporated the already executed plan so that the user did not have to start from scratch. Users are also able to ask about the purpose of any executed step. Then plan explanation (Seegebarth et al. 2012; Bercher et al. 2014) is initiated, which analyzes the causal links in the plan as well as the task hierarchy and exploits them to generate an explanation at runtime. We also conducted an empirical evaluation about our system in general and the impact of our plan explanations in particular (Bercher et al. 2014, 2018). One of the longer explanations presented in that evaluation (representing the causal link structure of the presented plan) was “*This step serves the goal to transmit the video signal of the blu-ray player to the TV. To this end, the video signal of the blu-ray player is transmitted over the HDMI-to-DVI adapter and the HDMI-to-DVI cable to the amplifier. From there, it is transmitted over the video-cinch cable to the TV.*” (Bercher et al. 2014).

DIY assistant

In cooperation with the Corporate Research Sector of the Robert Bosch GmbH, we developed a successor of the previous system applied to the scenario of DIY handyman support (Behnke et al. 2019; Bercher et al. 2021). Such DIY tasks include renovating furniture or constructing something from scratch, like a bird nesting house or any other

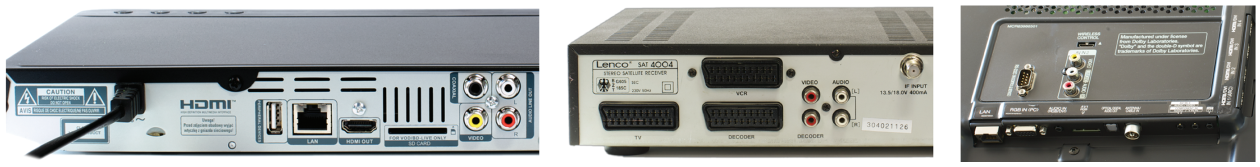
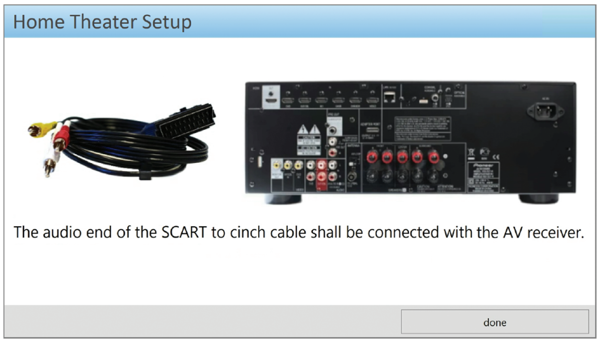


FIGURE 5 Back panels of the modeled devices. From left to right: blu-ray player, satellite receiver, and television

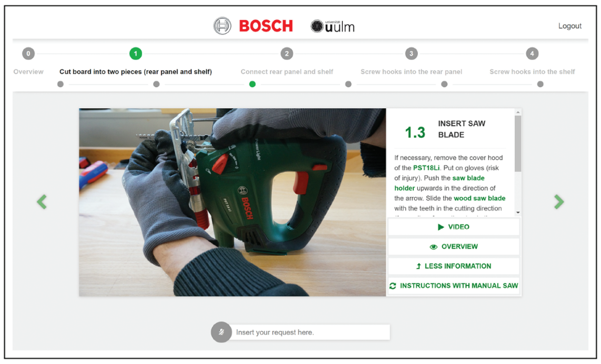


Home Theater Setup

The audio end of the SCART to cinch cable shall be connected with the AV receiver.

done

FIGURE 6 An instruction of our assembly assistant (Bercher et al. 2015, Figure 1, modified for enlargement)



BOSCH uulm Logout

0 Overview · 1 Cut board into two pieces (rear panel and shelf) · 2 Connect rear panel and shelf · 3 Screw hooks into the rear panel · 4 Screw hooks into the shelf

1.3 INSERT SAW BLADE

If necessary, remove the cover hood of the PST18Li. Put on gloves (risk of injury). Push the saw blade holder upwards in the direction of the arrow. Slide the wood saw blade with the teeth in the cutting direction

FIGURE 7 An instruction in our DIY assistant (Bercher et al. 2021, Figure 4). Reprinted by permission from Springer: Nature. KI—Künstliche Intelligenz. Do It Yourself, but Not Alone: *Companion*-Technology for Home Improvement—Bringing a Planning-Based Interactive DIY Assistant to Life. Bercher et al. (2021)

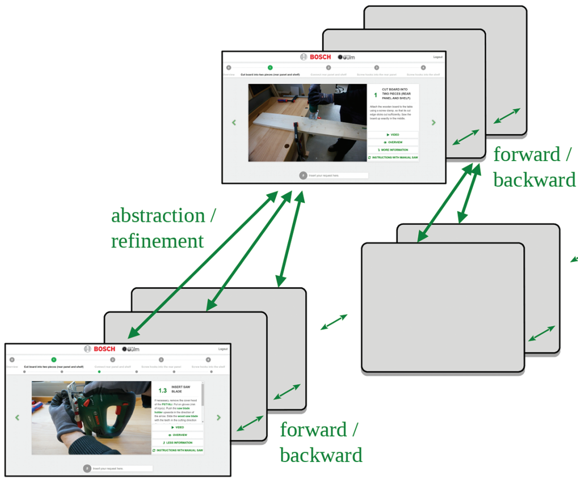


FIGURE 8 Illustration of navigation among different levels of abstraction of the assistant (Bercher et al. 2021, Figure 1).Reprinted by permission from Springer: Nature. KI—Künstliche Intelligenz. Do It Yourself, but Not Alone: *Companion*-Technology for Home Improvement—Bringing a Planning-Based Interactive DIY Assistant to Life. Bercher et al. (2021)

hobby project. Working on such projects often involves electric devices like drills or saws, where automated assistance can show even more of its potential since it can explain how these devices are to be used, remind of safety procedures, or even communicate with devices directly to inform the user about their internal states or wrong usage.

Assistance is again provided by a sequence of detailed instructions, generated at runtime based on the current situation, that is, available tools and equipment like nails or screws. An example instruction is shown in Figure 7.

In earlier work, we investigated how various components of complex assistance systems (including the underlying HTN planning model) can be represented in a central ontology to allow for a coherent knowledge model (Behnke et al. 2015). We again store certain parts of the model in an ontology to enable automatic question answering about certain background knowledge, e.g., why a certain kind of saw blade is used and how to recognize it, or why we ask to pre-drill (Behnke et al. 2019).

We exploit the model's task hierarchy by allowing the user to choose between different levels of abstraction and navigate back and forth on all levels (cf. Figure 8). Compound tasks were tasks like "Cut board into two pieces". This level of abstraction is appropriate for more experienced users, but others might need more detailed instructions. Reducing the abstraction level shows all involved primitive substeps such as steps to set up the tools.

The example project we modeled is the creation of a keyrack. We conducted a series of experiments with test subjects over the course of the project, evaluating the assistant's various development stages (Behnke et al. 2019, 2020; Kraus et al. 2020; Bercher et al. 2021). The last stages included communication between the assistant and the electric tools including pro-active dialogs (Kraus et al. 2020; Behnke et al. 2020).

CONCLUSION

This paper surveyed my work in the fields of POCL planning as well HTN planning. It spans from theory to practice, where most work done is on foundations of these frameworks, in particular via complexity investigations of various decision problems such as plan existence and changes to plans or models. Work surveyed also includes heuristic search as well as the application of the technology to two planning-based assistance systems.

CONFLICT OF INTEREST

The author declares that there is no conflict.

ORCID

Pascal Bercher https://orcid.org/0000-0002-0795-4320

REFERENCES

Alford, R., G. Behnke, D. Höller, P. Bercher, S. Biundo, and D. Aha. 2016. “Bound to plan: exploiting classical heuristics via automatic translations of tail-recursive HTN problems.” In *Proceedings of the ICAPS 2016*, 20–8. AAAI Press.

Alford, R., P. Bercher, and D. Aha. 2015a. “Tight bounds for HTN planning.” In *Proceedings of the ICAPS 2015*, 7–15. AAAI Press.

Alford, R., P. Bercher, and D. Aha. 2015b. “Tight bounds for HTN planning with task insertion.” In *Proceedings of the IJCAI 2015*, 1502–8. AAAI Press.

Alford, R., V. Shivashankar, U. Kuter, and D. Nau. 2012. “HTN problem spaces: structure, algorithms, termination.” In *Proceedings of the SoCS 2012*, 2–9. AAAI Press.

Alford, R., V. Shivashankar, U. Kuter, and D. Nau. 2014. “On the feasibility of planning graph style heuristics for HTN planning.” In *Proceedings of the ICAPS 2014*, 2–10. AAAI Press.

Barták, R., S. Ondrčková, G. Behnke, and P. Bercher. 2021. “Correcting hierarchical plans by action deletion.” In *Proceedings of the KR 2021*. IJCAI.

Behnke, G., P. Bercher, M. Kraus, M. Schiller, K. Mickeleit, T. Häge, M. Dorna, et al. 2020. “New developments for robert—assisting novice users even better in DIY projects.” In *Proceedings of the ICAPS 2020*, pp. 343–7. AAAI Press.

Behnke, G., D. Höller, P. Bercher, and S. Biundo. 2016. “Change the plan—how hard can that be?” In *Proceedings of the ICAPS 2016*, 38–46. AAAI Press.

Behnke, G., D. Höller, and S. Biundo. 2015. “On the complexity of HTN plan verification and its implications for plan recognition.” In *Proceedings of the ICAPS 2015*, 25–33. AAAI Press.

Behnke, G., D. Höller, A. Schmid, P. Bercher, and S. Biundo. 2020. “On succinct groundings of HTN planning problems.” In *Proceedings of the AAAI 2020*, 9775–84. AAAI Press.

Behnke, G., F. Pollitt, D. Höller, P. Bercher, and R. Alford. 2022. “Making translations to classical planning competitive with other HTN planners.” In *Proceedings of the AAAI 2022*. AAAI Press.

Behnke, G., D. Ponomaryov, M. Schiller, P. Bercher, F. Nothdurft, B. Glimm, and S. Biundo. 2015. “Coherence across components in cognitive systems – one ontology to rule them all.” In *Proceedings of the IJCAI 2015*, pp. 1442–9. AAAI Press.

Behnke, G., M. Schiller, M. Kraus, P. Bercher, M. Schmautz, M. Dorna, M. Dambier, W. Minker, B. Glimm, and S. Biundo. 2019. “Alice in DIY wonderland or: instructing novice users on how to use tools in DIY projects.” *AI Communications* 32(1): 31–57.

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

Bercher, P., R. Alford, and D. Höller. 2019. “A survey on hierarchical planning—one abstract idea, many concrete realizations.” In *Proceedings of the IJCAI 2019*, 6267–75. IJCAI.

Bercher, P., G. Behnke, D. Höller, and S. Biundo. 2017. “An admissible HTN planning heuristic.” In *Proceedings of the IJCAI 2017*, 480–8. IJCAI.

Bercher, P., G. Behnke, M. Kraus, M. Schiller, D. Manstetten, M. Dambier, M. Dorna, W. Minker, B. Glimm, and S. Biundo. 2021. “Do it yourself, but not alone: *companion*-technology for home improvement—bringing a planning-based interactive DIY assistant to life.” *KI* 35: 367–75.

Bercher, P., S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg. 2014. “Plan, repair, execute, explain—how planning helps to assemble your home theater.” In *Proceedings of the ICAPS 2014*, 386–394. AAAI Press.

Bercher, P., D. Höller, G. Behnke, and S. Biundo. 2016. “More than a name? on implications of preconditions and effects of compound HTN planning tasks.” In *Proceedings of the ECAI 2016*, 225–233. IOS Press.

Bercher, P., D. Höller, G. Behnke, and S. Biundo. 2017. *Companion Technology—A Paradigm Shift in Human-Technology Interaction. Chapter 5: User-Centered Planning*. Cognitive Technologies, 79–100. Springer. https://doi.org/10.1007/978-3-319-43665-4_5

Bercher, P., S. Keen, and S. Biundo. 2014. “Hybrid planning heuristics based on task decomposition graphs.” In *Proceedings of the SoCS 2014*, 35–43. AAAI Press.

Bercher, P., S. Lin, and R. Alford. 2022. “Tight bounds for hybrid planning.” In *Proceedings of the IJCAI 2022*. IJCAI.

Bercher, P., and C. Olz. 2020. “Pop ≡ POCL, right? Complexity results for POCL makespan minimization.” In *Proceedings of the AAAI 2020*, 9785–93. AAAI Press.

Bercher, P., F. Richter, F. Honold, F. Nielsen, F. Schüssel, T. Geier, T. Hörnle, et al. 2018. “A Companion-system Architecture for Realizing Individualized and Situation-adaptive User Assistance.” Technical report, Ulm University.

Bercher, P., F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nothdurft, et al. 2015. “A planning-based assistance system for setting up a home theater.” In *Proceedings of the AAAI 2015*, 4264–5. AAAI Press.

Biundo, S., P. Bercher, T. Geier, F. Müller, and B. Schattenberg. 2011, April. “Advanced user assistance based on AI planning.” *Cognitive Systems Research* 12(3-4): 219–36. Special Issue on Complex Cognition.

Biundo, S., D. Höller, B. Schattenberg, and P. Bercher. 2016. “Companion-technology: an overview.” *Künstliche Intelligenz* 30(1): 11–20. Special Issue on Companion Technologies.

Bylander, T. 1994. “The computational complexity of propositional STRIPS planning.” *Artificial Intelligence* 94(1-2): 165–204.

Chen, D., and P. Bercher. 2021. “Fully observable nondeterministic HTN planning—formalisation and complexity results.” In *Proceedings of the ICAPS 2021*, 74–84. AAAI Press.

Chen, D. Z., and P. Bercher. 2022. “Flexible fond htn planning: a complexity analysis.” In *Proceedings of the ICAPS 2022*. AAAI Press.

Elkawkagy, M., P. Bercher, B. Schattenberg, and S. Biundo. 2012. “Improving hierarchical planning performance by the use of landmarks.” In *Proceedings of the AAAI 2012*, 1763–9. AAAI Press.

Elkawkagy, M., B. Schattenberg, and S. Biundo. 2010. “Landmarks in hierarchical planning.” In *Proceedings of the ECAI 2010*, 229–34. IOS Press.

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

Geier, T., and P. Bercher. 2011. “On the decidability of HTN planning with task insertion.” In *Proceedings of the IJCAI 2011*, 1955–61. AAAI Press.

Ghallab, M., D. Nau, and P. Traverso. 2016. *Automated Planning and Acting*. New York: Cambridge University Press.

Ghallab, M., D. S. Nau, and P. Traverso. 2004. *Automated Planning: Theory and Practice*. Amsterdam: Morgan Kaufmann.

Höller, D., G. Behnke, P. Bercher, and S. Biundo. 2014. “Language classification of hierarchical planning problems.” In *Proceedings of the ECAI 2014*, 447–52. IOS Press.

Höller, D., G. Behnke, P. Bercher, and S. Biundo. 2021. “The PANDA framework for hierarchical planning.” *KI* 35: 391–6.

Höller, D., G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford. 2020. “HDDL: An extension to pddl for expressing hierarchical planning problems.” In *Proceedings of the AAAI 2020*, 9883–91. AAAI Press.

Höller, D., and P. Bercher. 2021. “Landmark generation in HTN planning.” In *Proceedings of the AAAI 2021*. AAAI Press.

Höller, D., P. Bercher, and G. Behnke. 2020. “Delete- and ordering-relaxation heuristics for HTN planning.” In *Proceedings of the IJCAI 2020*, 4076–83. IJCAI.

Höller, D., P. Bercher, G. Behnke, and S. Biundo. 2018. “A generic method to guide HTN progression search with classical heuristics.” In *Proceedings of the ICAPS 2018*, 114–22.

Höller, D., P. Bercher, G. Behnke, and S. Biundo. 2020. “HTN planning as heuristic progression search.” *JAIR* 67: 835–80.

Kambhampati, S., A. Mali, and B. Srivastava. 1998. “Hybrid planning for partially hierarchical domains.” In *Proceedings of the AAAI 1998*, 882–8. AAAI Press.

Kraus, M., M. Schiller, G. Behnke, P. Bercher, M. Dorna, M. Dambier, B. Glimm, S. Biundo, and W. Minker. 2020. “Was that successful? On integrating proactive meta-dialogue in a DIY-assistant system using multimodal cues.” In *Proceedings of the ICMI 2020*, 585–594. ACM.

Lin, S., and P. Bercher. 2021. “Change the world—how hard can that be? On the computational complexity of fixing planning models.” In *Proceedings of the IJCAI 2021*, 4152–9. IJCAI.

Muise, C., J. C. Beck, and S. A. McIlraith. 2016. “Optimal partial-order plan relaxation via maxsat.” *JAIR*, 57: 113–49.

Nau, D., T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. 2003. “SHOP2: an HTN planning system.” *JAIR* 20: 379–404.

Olz, C., and P. Bercher. 2019. “Eliminating redundant actions in partially ordered plans—a complexity analysis.” In *Proceedings of the ICAPS 2019*, 310–9. AAAI Press.

Olz, C., S. Biundo, and P. Bercher. 2021. “Revealing hidden preconditions and effects of compound HTN planning tasks—a complexity analysis.” In *Proceedings of the AAAI 2021*, pp. 11903–12. AAAI Press.

Seegebarth, B., F. Müller, B. Schattenberg, and S. Biundo. 2012. “Making hybrid plans more clear to human users—a formal approach for generating sound explanations.” In *Proceedings of the ICAPS 2012*, 225–33. AAAI Press.

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

Waters, M., B. Nebel, L. Padgham, and S. Sardina. 2018. “Plan relaxation via action debinding and deordering.” In *Proceedings of the ICAPS 2018*, 278–87. AAAI Press.

Waters, M., L. Padgham, and S. Sardina. 2020. “Optimising partial-order plans via action reinstantiation.” In *Proceedings of the IJCAI 2020*, 4143–51. IJCAI.

Weld, D. S. 1994. “An introduction to least commitment planning.” *AI Magazine* 15(4): 27–61.

AUTHOR BIOGRAPHY

Pascal Bercher is a Senior Lecturer at the Australian National University. Before joining the ANU late 2019, he was at the Institute of Artificial Intelligence of Ulm University, where he pursued his doctoral degree from 2009 to 2017, followed by 2 years of post-doc. From 2016 to 2019, he was the project coordinator of a technology transfer project, which was a collaboration of Ulm University’s Institute of Artificial Intelligence and its Institute of Communications Engineering with the Corporate Research Sector of Robert Bosch GmbH. He focuses on theoretical investigations as well as algorithm and heuristic development for hierarchical task network (HTN) planning and partial order causal link (POCL) planning—as surveyed in this paper.

How to cite this article: Bercher, P. 2022. “Hierarchical planning and reasoning about partially ordered plans—From theory to practice.” *AI Magazine* 43: 353–364. https://doi.org/10.1002/aaai.12073