Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Hierarchical Planning through Propositional Logic – Highly Efficient, Versatile, and Flexible

von Gregor Behnke
aus Ribnitz-Damgarten

am Institut für Künstliche Intelligenz
Institutsdirektorin: Prof. Dr. Susanne Biundo-Stephan

im August 2019

Amtierender Dekan: Prof. Dr.-Ing. Maurits Ortmanns
Gutachter: Prof. Dr. Susanne Biundo-Stephan
Gutachter: Prof. Dr. Dana S. Nau
Gutachter: Prof. Dr. Uwe Schöning
Tag der Promotion: 02. Dezember 2019

ᚠᛏᚨᚱᛏᚨᚱ ᚾᚨᚷ ᛏᚨᚱᚨᚾᚾᚨ ᚨᚱᚾᚨᚾᚾᚨ

Acknowledgements

Lord, what fools these mortals be!
– Puck, A Midsummer Night's Dream, Act III, Scene II

A foolish endeavour it would have been to undertake this work without the help of others. I therefore express my thanks and deepest gratitude to all those who have supported me in my research and beyond.

Prof. Dr. Susanne Biundo-Stephan
for allowing me to partake in the voyage to the vision of companion systems and all the opportunities surrounding it and for providing helpful critique and feedback on my work

Prof. Dr. Dana Nau and Prof. Dr. Uwe Schöning
for the huge effort they put into disentangling the indecipherable rambling in this thesis

Prof. Dr. Birte Glimm
for her direct and diligent critique and advice and for her willingness to work up to the middle of the night

Dr. Pascal Bercher
for always dutifully sacrificing his own time at my behest even though this thesis and my work acknowledges him all too rarely

Dr. Marvin Schiller
for his kindness at all times in discussions

the unknown reviewers
for liking our "enjoyable proofs"

Daniel Höller
for his infatuation with new ideas, their discussion, and implementation, for being a good friend, and for the time spent waiting for the bus

Jana Behnke
for all the effort she put into providing opportunities to a little boy, even though he woke her up at 5 am on every Sunday morning

Abstract

AI Planning is a core technology in enabling advanced assistance for human users. When faced with complex problems such as handicraft tasks for household repairs, Do-It-Yourself projects, or difficult assembly tasks, a planning-based assistance system can provide individualised and context-dependent instructions. It thus effectively supports the user in achieving his or her goals. High-quality assistance should in addition conform to the user's current wishes and preferences to ensure maximum utility. This can be achieved by involving the user into the planning process, i.e. by making planning mixed-initiative. Hierarchical Task Network (HTN) planning is a method particularly well suited for providing user support, as it resembles the means and structures humans use for problem solving. We identified two major challenges that every mixed-initiative HTN planning system must face and show how to address them: generating plans quickly and flexibly altering them according to the user's demands.

The main focus of this thesis lies on addressing the first challenge, i.e. on developing a quickly responding and efficient HTN planner. Designing efficient HTN planners is particularly difficult. Their algorithms have to take both dimensions of the problem description – state and hierarchy – into account and have to consider interactions between them. We use a translation into propositional logic, enabling for the first time a *uniform* view on the whole planning problem. First, we describe a new encoding for totally-ordered HTN planning, before extending it in two consecutive steps to capture general, partially-ordered domains. Second, we introduce Path Decomposition Trees (PDTs) and Solution Order Graphs (SOGs) which enable a compact encoding and alleviate unnecessary reasoning from a SAT solver. They also pave the way for future insights into structural properties of HTN planning problems, which allow for more efficient planning as well as for more advanced user support. Third, we show that our encodings are a significant empirical improvement over the current state of the art in HTN planning. Lastly, we present a fundamental technique for optimal HTN planning – which is especially important in assistance scenarios – namely a method to compute succinct depth bounds for plan-length optimisation.

In a mixed-initiative planning environment, users will frequently request the planner to change a currently considered plan. We start solving this second challenge by considering the most basic task involved: to verify that the changed plan is a solution to the planning problem at hand. First, we show that this task is $\mathbb{NP}$-complete. Second, we develop the first plan verifier for HTN planning, which is based on a transformation into propositional logic. Third, we analyse and categorise the requests possibly made by a user and show that the objectives posed by her or him can be suitably represented as formulae in Linear Temporal Logic (LTL). Fourth, we analyse the computational complexity of changing a plan, showing that this task can be between $\mathbb{NP}$-complete and undecidable. Lastly, we use our SAT-based planner to change a plan with respect to a request formulated as an LTL formula. We show that the full spectrum of LTL formulae can be supported efficiently in a propositional encoding. For that, we introduce a new theoretical foundation for reasoning about parallelism in LTL traces.

The practical applicability of our techniques has been demonstrated within a joint transfer project with Robert Bosch GmbH. In it, we developed an assistance system that guides novice users through a handicraft Do-It-Yourself project. The underlying hierarchical planning model is highly complex. Currently the only planner that is able to find plans for this model within an acceptable time frame is the SAT-based HTN planner developed as part of this thesis.

Contents

Note on References: References to bibliographic items are prefixed with a letter indicating their type. References prefixed with C are peer-reviewed core publications of this dissertation and reprinted in full in Appendix B. References prefixed with F are further publications of which I am a co-author. All F publications except for [F43] are peer-reviewed. References prefixed with R denote related work of with I am not a(n) (co-)author.

1. Introduction

Humans, especially inexperienced ones, often require assistance when faced with complex problems. For example, they may require assistance via providing appropriate instructions when performing handicraft tasks such as household repairs or construction projects [C3, F18], when developing a personal fitness schedule [F27], or when setting up a home theatre ensemble [F31, F40, R74]. In these scenarios, planning-based assistance can suitably provide individualised instructions and thus support the user to achieve his or her goals. Merely providing a sequence of instructions is often not sufficient [F26]. We argue that users often require a mixed-initiative interaction allowing them to adapt the presented instructions to their personal preferences and wishes. Existing planners are often not fast enough to ensure an appropriately short response time for such assistance.

This thesis is concerned with the theoretical foundations and the practical realisation of providing adaptive planning-based assistance. The main focus lies on developing the planning capabilities necessary for quickly processing the user's wishes and requests with respect to a given plan. In this thesis, we consider hierarchical planning in form of Hierarchical Task Network (HTN) planning [R88, R113, R143]. It is well suited to deliberate with humans on planning processes, as on the one hand it resembles the way humans plan themselves and on the other hand is quite appropriate for realising for advanced capabilities, e.g. the explanation and repair of plans [F30, R74, R87].

We start by discussing existing planning-based assistance systems in Chapter 3. We then argue that any request of a user to change a given plan can be suitably interpreted as formulae in Linear Temporal Logic (LTL). Next, we provide the first theoretical foundation for mixed-initiative planning by classifying and studying possible requests to change a given plan. For most of these requests, the task of plan verification – asking, given a sequence of actions, whether it is a solution to the current planning problem – plays a central role. We show that this task is $\mathbb{NP}$-complete for HTN planning [C9] and for another variant of hierarchical planning [F35]. Next, we classify possible change requests for plans along two axes: what to change and which follow-up modifications are allowed. For all these requests, we investigate their computational complexity [C8]. Lastly, we generalise these results in this thesis to cover arbitrary requests posed.

These theoretical results – especially the $\mathbb{NP}$-completeness of HTN plan verification – led to reviving an idea for solving HTN planning problems: *HTN planning via logic*. It is described in Chapter 4. This planning approach has – in addition to its efficiency – advantages when applied in mixed-initiative planning, as it enables a seamless integration of user requests into the planning process. Prior to the work of this thesis two

encodings for HTNs in logic existed. One translates HTN planning problems into propositional logic [R138], but assumes that first, the HTN structure is acyclic, that second, each task can occur at most once during decomposition – which is an even stricter than acyclicity, and that third tasks can be arbitrarily inserted – not just via decomposition. These restrictions are severe and limit the practical usability of the translation. The second translation encodes totally ordered HTN planning problems into *answer set programming*, modelling the progression search used by SHOP [R114]. We developed a new translation of HTN planning problems into SAT, which is applicable without restrictions and adheres to the standard semantics regarding task decomposition of HTN planning [R88, R143] and Hybrid Planning [F35, R124].

As a first step in creating a planner that exploits the concept of *planning via logic*, we operationalised the complexity results for HTN plan verification by developing a translation into propositional logic. The encoding (as all other encodings in this thesis) represents all possible decompositions up to a given depth-bound. This work constitutes the first plan verifier for HTN planning [C7], and has led to the development of other verifiers (e.g. by Barták et al. [R48]). Our verifier can be utilised in a wide range of scenarios beyond that of mixed-initiative planning, in particular for post-optimising plans and for local plan repair.

A SAT-based planner can be obtained from the verification encoding by replacing the clauses asserting the solution that should be verified with a formula expressing state transitions. To ensure the completeness of this planner, the bound on the decomposition depth is increased stepwise until a solution has been found. Unfortunately, the use of this first encoding for planning is limited, mostly due to the size of the formula and due to structures in the formula that make it particularly difficult to solve for most SAT solvers. Therefore, we developed a more compact propositional encoding specifically designed for totally-ordered planning domains. It is based on our newly introduced theoretical concept of Path Decomposition Trees (PDTs, [C5]) which are compact representations of all possible decompositions up to a given depth-limit. The resulting planner shows good results, outperforming all currently known techniques on totally-ordered HTN planning problems. We developed two modifications of the encoding, each allowing to handle partial order in the domain [C2, C6]. For one of them, we developed the Solution Order Graph (SOG), a compact representation of the partial order of all solutions up to a given decomposition depth [C2]. SOGs allow for an efficient representation of partially ordered solutions in a propositional formula. The application of SOGs and PDTs is however not limited to SAT-based planning: they are generic representations of the problem, which make structural properties of the problem explicit. As such they may be applied in other algorithms, e.g. in search-based ones, as well as to e.g. improve heuristics. The new encoding for general HTN planning based on an SOG exhibit good performance when compared to state-of-the-art HTN planners.

The so-far presented planning techniques only allow for finding some solution – without a guarantee to its quality. In practical applications short or cheap (according to some cost function) plans are often required. This is especially true if human users are involved as they might spot obvious redundancies in the plan. Finding guaranteed optimal plans in HTN planning via SAT is difficult, as there is no strict correlation between decomposition depth and plan length. As such, it might be possible to derive shorter plans if deeper

decompositions are allowed. Optimal SAT-based HTN planning thus requires to compute an upper bound on the decomposition depth necessary to reach all plans up to a given length. We show how succinct upper bounds can be found [C1]. Our planner using this bounding technique outperforms other optimal HTN planners.

In Chapter 5, we show that the newly developed SAT-based HTN planner can suitably form the basis for addressing the user's instructions within a mixed-initiative assistance system. As noted above, a user's requests can be interpreted as formulae in LTL. LTL constraints can be added to a propositional planning formula easily, as an encoding is already known [R96]. However, this encoding cannot cover full LTL, but only formulae without the temporal *next* operator X. This operator is especially useful when formalising a user's requests to change a plan, as e.g. the request to perform an action as the immediate next step requires this operator. We show how the encoding can be modified to also cover the operator X. In doing so, we develop a new theoretical framework, Partial Evaluation Traces, to reason about LTL formulae [C4]. Using this framework, we not only added support for the operator X, but also improved the base encoding allowing for even faster planning [C4].

Lastly, we show that the developed technologies have been successfully applied in industrial research. The techniques described and developed within this thesis have been applied in a technology transfer project which was jointly conducted with Robert Bosch GmbH. In it, we have created an assistant that guides a novice user through a handicraft Do-It-Yourself project, such as building a wooden key rack [C3, F18]. The planning model in this scenario is highly complex and currently only the newly developed SAT-based planner is able to find plans in an acceptable time frame. In this context, we discuss how different types of knowledge (procedural and static) in such an assistant can be suitably integrated [F33, C10].

2. From Classical to Hierarchical Planning

2.1. Summary

Planning is the means of deciding which actions to take in order to achieve a given objective. This complex cognitive ability is one of the most notable abilities possessed by humans, setting them apart from most other live.[1] The need for planning when producing composite tools may have even influenced the evolution of the human brain [R123]. Planning helps to make strategic long-term decisions, to save resources, time, and energy, and to achieve one's objectives even in an adversarial environment. Naturally, early computer scientists aimed to develop systems that were able to emulate the human ability to plan and to solve problems. Beginning with the general problem solver [R159], research in planning began to focus on the task of selecting a sequence of actions that, if executed, would lead to a desired goal. Solving these problems automatically with a planner requires the problem to be described as an abstraction of the real world, which mirrors its relevant aspects. The idea is that a plan, once found for the abstract model, would also be workable in the real world, provided the planning model was chosen appropriately. As a consequence, a variety of simple and complex planning formalisms was developed, each suited to reflect some features of the real world (e.g. STRIPS [R158], PDDL [R130], PDDL with time [R116, R120], RDDL [R91], and HTN planning [R143, R156] formulated using HDDL [F14]). In these models, planning boils down to combinatorial optimisation.

Planners have been applied in a multitude of practical scenarios, demonstrating one of their key capabilities: being solvers for arbitrary – and not just a few specific – planning domains. For example, planners have been used to optimise logistics [R86], to optimise the transport of plants in automated greenhouses [R90], for determining the reactions needed for the synthesis of organic compounds [R71], for planning experiments in synthetic biology [R54], for generating narratives [R67, R105], for tutoring systems or computer games [R81], and for controlling NASA's rovers on Mars [R100, R112] and the Rosetta space probe [R70].

The results and techniques that we present in this thesis will only be formulated for propositional planning, i.e. planning with the assumption that the state of the world can be represented as a set of atomic logical propositions. The techniques we present are however also applicable to planning problems which are specified using a lifted formalisation (see Section 2.3). Notably, the evaluations we present throughout this thesis

[1] At least on earth.

(Sections 4.2.3, 4.3.3, 4.4.5, and 4.5.2) consider mostly planning problems specified in a lifted fashion. We further assume non-continuous, i.e. discrete, time. We however suppose that the developed techniques can be extended to also handle the above-mentioned more complex types of planning problems, i.e. domains containing continuous state variables and time (see, e.g. Section 4.6).

In this chapter, we introduce classical and hierarchical planning – in the notation used throughout this thesis. We start by giving an introduction into classical planning. Based on it, we describe the extensions hierarchical planning makes. We further introduce theoretical concepts and results by other researchers that are relevant to the techniques and results developed in this thesis. Lastly, we briefly introduce Linear Temporal Logic (LTL), which we will use in Chapters 3 and 5.

Core publications described in this chapter

This chapter serves to introduce relevant concepts for this thesis and thus does not contain any core contribution.

2.2. Classical Planning

The most basic formalism for planning is that of classical planning, which is sometimes also called STRIPS, named after the first planner for the by now standard formalism of classical planning [R158]. In classical planning, states of the world are described in terms of sets of proposition symbols. We denote with L the (finite) set of all proposition symbols, which can e.g. include propositions like `attached_battery7_drill5`. It describes that the `battery7` is attached to the `drill5`. A state is any subset s of L. All propositions $p \in s$ are considered true in s while all others are considered false. Actions provide the means to transform states into each other. Every action a is associated with a precondition $prec(a)$ and an effect $\mathit{eff}(a)$. The precondition $prec(a)$ is any propositional formula over the proposition symbols L. It describes the circumstances under which the action a can be executed. An action a is considered executable in a state s if and only if $s \models prec(a)$. For example the action `attach_battery7_drill5` – which shall attach `battery7` to `drill5` – can only be executed if `battery7` is compatible with `drill5`. Hence, the precondition will contain the proposition symbol `compatible_battery7_drill5`. The effect $\mathit{eff}(a)$ of an action describes how the application of a changes the state to which it is applied. The effect can either be described as a conjunction of positive or negative proposition symbols or equivalently by two sets of proposition symbols: $add(a)$ and $del(a)$. The set $add(a)$ – the adding effects – contains the positively occurring propositions, while $del(a)$ – the deleting effects – contains the negative ones. If a is executed in the state s, the propositions in $del(a)$ are removed from s and those in $add(a)$ are added. More formally, executing a in s results in the state

$(s \setminus del(a)) \cup add(a)$ and is denoted with $\gamma(s, a)$. For a sequence of actions $\pi = a_1 \dots a_n$ applied in a state s, we define the resulting state as

$$\gamma(s, \pi) = \begin{cases} s & \pi = \varepsilon \\ \text{undefined} & \pi = a_1 a_2 \dots a_n \text{ and } s \not\models prec(a_1) \\ \gamma(\gamma(s, a_1), a_2 \dots a_n) & \pi = a_1 a_2 \dots a_n \text{ and } s \models prec(a_1) \end{cases}$$

A planning problem in the classical setting is given in terms of two sets of propositions: s_I and g – the initial state and the goal. A planner's objective is to find a sequence of actions π – called a plan – that if executed in the state s_I will lead to a state $s_g = \gamma(s_I, \pi)$ in which the goal g holds, i.e. $g \subseteq s_g$. Note that this necessarily entails that preconditions of every action in π are satisfied, else s_g would be undefined. From a computational point of view, determining whether an solution to a classical planning problem exists is $\mathbb{PSPACE}$-complete [R147].

There are several extensions to this basic classical formalism. For example planners often allow for conditional effects [R151], which describe effects that are only executed if the state in which an action is applied satisfies a given formula. Conditional effects, as well as other similar extensions, can be compiled into an equivalent model in the presented formalism [R132]. The SAS+ planning formalism [R146] extends the representation of states from a simple boolean representation to a multi-valued representation. In it, a state is an assignment of a value to each state variable. Similar to conditional effects, SAS+ planning problems can be compiled into the presented formalism. For both conditional effects (and similar extensions) and SAS+ it is more efficient to develop planners that can handle them natively. A compilations either increases the size of the problem significantly (for conditional effects up to exponentially many new actions in the number of conditional effects per action are needed) or hide relevant structural information from the planner.

2.3. Lifted Planning

The description of classical planning in the previous section assumed that both the proposition symbols and the actions are mere symbols, e.g. `attached_battery7_drill5`. While such a simplistic model might be adequate for theoretical investigations and the development of planners, it is very impractical for modelling planning domains. The model would have to specify all proposition symbols and actions explicitly, even though they often conform to a pattern. For example, a DIY domain might contain the predicate `attached_b_d` for every battery `b` and drill `d`. The same holds for the `attach_b_d` actions. To allow for specifying such domains compactly, they are commonly specified in a lifted fashion. Here, proposition symbols and actions are not only names, but names with a sequence of parameter variables. For example, we can declare an action `(attach ?b ?d)`.[2] Each variable is assigned to a type, i.e. a set of constants. In the example, the type of `?b` is `battery` – the set of all batteries – while the type of `?d` is `drill` – the set of all drills. The preconditions of actions are described by function-free first order

[2] In planning, variable names are commonly prefixed with question marks, as e.g. in PDDL.

formulae. The add and delete lists of actions similarly contain positive, function-free literals.

The simple propositional representation presented above can be generated from such a lifted representation by enumerating every assignment, i.e. instantiation, of variables to constants of the respective types. Of course, such a simple instantiation procedure will generate actions that can never be executable. To compute only the potentially reachable groundings, several techniques have been developed. These include the planning graph [R140] and the conversion into SAS+ [R101].

Throughout this thesis, we will describe techniques and ideas using a grounded propositional representation of planning problems. In examples and in practical applications, we will however use a lifted representation, both to ease modelling and to make models easier to read and to ease interactions with users. The presented theory and techniques are (with appropriate modifications) nevertheless applicable, since we can create a propositional model from a lifted one via grounding.

2.4. Hierarchical Planning

Classical planning aims solely at finding a sequence of actions from a pool of available actions that transforms the initial state into a goal state. The domain structure and the rationale used when solving these planning problems may not conform with the ways humans would solve the same problem, which tend to solve problems in a hierarchical fashion [R154]. In addition to a portfolio of available elementary actions, humans also perform reasoning over more abstract tasks which are composed of (potentially) a multitude of elementary actions [R111]. The intensity with which humans do this depends on the cognitive difficulty of the problem. While they tend to rely more heavily on abstractions when solving easier and day-to-day tasks, reasoning in complex tasks is done in a manner more similar to forward search on elementary actions [R107]. In any case, it is useful to be able to refer to more abstract courses of action when interacting with human users. This e.g. comes into play when a planning-based system explains its behaviour to the user [R74, R84], where an abstraction hierarchy allows for shorter and more focussed explanations. Lastly, it is often also more intuitive to formulate the aim of a planning problem not in terms of a state to reach, but in terms of tasks to preform. For example, if a user wishes to participate in a conference, the initial and final states of a plan are identical – with the sole exception that the user has less money after the conference. In contrast, it is intuitive to formulate the aim in terms of a high-level task to perform: `(visit ICAPS_2019)`.

2.4.1. Hierarchical Task Network Planning

The most commonly used formalism for planning problems that contains abstract tasks and defines its aims in terms of tasks to perform is Hierarchical Task Network (HTN) planning [R44, R143, R156]. In HTN planning, we distinguish two types of tasks: primitive actions and abstract tasks. Primitive actions are formulated with the same means as actions in classical planning, i.e. preconditions and effects. They represent the lowest

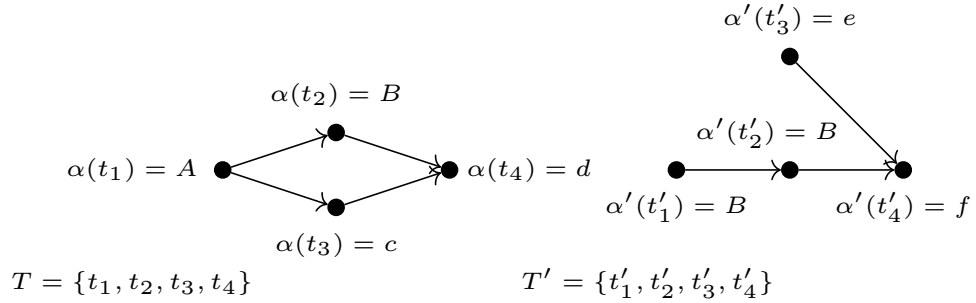


Figure 2.1.: Two example task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ where the order is given by the depicted graphs. Compound task names are indicated with capital letters, while lower-case letters indicate primitive action names. Copied from [C2]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

level of abstraction available, i.e. those activities that can be performed without the need for further refining them. Abstract tasks represent more complex courses of action. Formally, abstract tasks are identified by their names. Whenever we are referring to a task from now on, we mean either an abstract task or a primitive action.

An HTN planning problem connects abstract tasks with primitive actions via decomposition methods. Such a method is a rule $a \to tn$ that provides an option to perform the left hand side abstract task a: by performing the tasks contained in the right hand side tn. The means by which the right hand side tn is described gives HTN planning its name: task networks. A task network is a partially ordered multi-set of tasks. Since task networks are multi-sets, i.e. can contain the same task twice, we have to introduce IDs to distinguish them. Formally, we define a task network as follows.

Definition 1 (Task Network [R88, R143])
Let X be a set of tasks. A task network tn is a triple $(T, \prec, \alpha)$ where

- *T is a set of task identifiers,*
- *$\prec \subseteq T \times T$ is a partial order over T, and*
- *$\alpha : T \to X$ labels each task identifier with a task.*

We denote with TN_X the set of all task networks over a set of tasks X. For a given task network tn, we denote with $T(tn)$ the set of its task identifiers, with $\prec(tn)$ its task ordering, and with $\alpha(tn)$ its task labelling.

Two task networks are depicted in Figure 2.1. Based on the definition of task networks, an HTN planning problem is defined as follows.

Definition 2 (HTN planning problems [R88])
An HTN planning problem is a 6-tuple $\mathcal{P} = (L, A, O, M, a_I, s_I)$ where

- *L is a set of proposition symbols,*
- *A is a set of abstract tasks,*
- *O is a set of primitive actions (or **o**perators),*

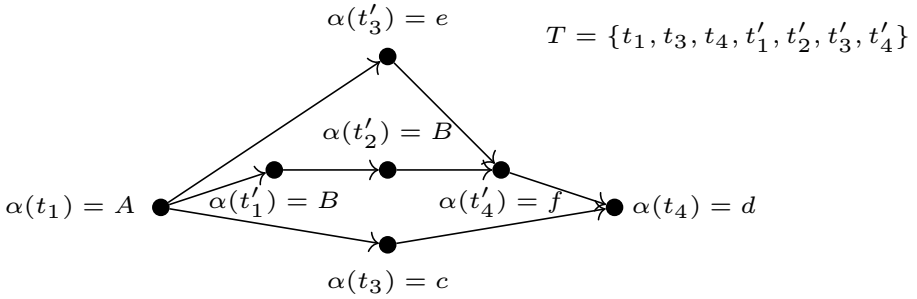


Figure 2.2.: The task network $tn^* = (T^*, \prec^*, \alpha^*)$ resulting from applying the method (B, tn') to t_2 in tn of Fig. 2.1. Copied from [C2]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

- *M is a set of decomposition methods (a, tn_a) where $a \in A$ and $tn_a \in TN_{A \cup O}$,*
- *$a_I \in A$ is the initial abstract task, and*
- *$s_I \subseteq L$ is the initial state*

The problem's decomposition methods describe how abstract tasks can be performed by means of executing more primitive actions. A decomposition method (a, tn_a) (also written as $a \to tn_a$) allows for replacing the abstract task a in a task network tn with the task network tn_a. While doing so, the tasks in tn_a inherit the relative orderings of a in tn.

Definition 3 (Task Decomposition [R88])
Let $\mathcal{P} = (L, A, O, M, a_I, s_I)$ be an HTN planning problem and let $tn = (T, \prec, \alpha) \in TN_{A \cup O}$ be a task network and $t \in T$ a task identifier labelled with the abstract task $a = \alpha(t)$. Furthermore, let $(a, tn_a) \in M$ be a decomposition method and assume that $T \cap T(tn_a) = \emptyset$. If this is not the case, we can rename the task identifiers in tn appropriately.

Decomposing the task identifier t in the task network tn using the method (a, tn_a) results in the task network $tn' = (T', \prec', \alpha')$ where:

$$
\begin{aligned}
T' &= (T \setminus \{t\}) \cup T(tn_a) \\
\alpha' &= (\alpha \setminus \{(t, a)\}) \cup \alpha(tn_a) \\
\prec' &= (\prec \setminus \{(x, t), (t, x) \mid x \in T\}) \cup \prec(tn_a) \cup \\
&\qquad \{(x, t') \mid (x, t) \in \prec, t' \in T(tn_a)\} \cup \{(t', x) \mid (t', x) \in \prec, t' \in T(tn_a)\}
\end{aligned}
$$

We write $tn \to_D tn'$ if there is any decomposition method $m \in M$ and task identifier $t \in T(tn)$ such that applying m to t in tn leads to tn'. Further, $\to_D^*$ denotes the transitive hull of $\to_D$.

If we consider the first task network in Figure 2.1 and apply a method to its task identifier t_2 labelled B whose task network is the second one shown in Figure 2.1, i.e. the method $B \to tn'$, we obtain the task network shown in Figure 2.2.

The objective in an HTN planning problem is defined in terms of an initial abstract task a_I. The objective is to apply decomposition methods to a_I until a task network containing only primitive actions has been reached, which we call a primitive decomposition. The restriction to a single goal task is solely for notational purposes. Any partially-ordered set of tasks can be used as a goal by adding a single additional decomposition method.

Definition 4 (Primitive Decomposition)
Let $\mathcal{P} = (L, A, O, M, a_I, s_I)$ be an HTN planning problem. A task network tn is called a primitive decomposition of $\mathcal{P}$ if $(\{x\}, \emptyset, \{(x, a_I)\})^3 \rightarrow_D^* tn$ holds and $tn \in TN_O$, i.e. if tn contains only primitive actions and can be reached via decomposition from the initial abstract task.

Determining for a given HTN planning problem whether a primitive decomposition exists is trivial and can e.g. be done by a bottom-up marker algorithm. For a primitive decomposition tn to be a solution to a given planning problem $\mathcal{P}$ we also have to show that it is executable. Here, executability means that there is a linearisation of the tasks in tn that is executable in the initial state s_I.

Definition 5 (Executability and Solutions)
Let $\mathcal{P} = (L, A, O, M, a_I, s_I)$ be an HTN planning problem. Let $tn \in TN_O$ be a primitive action network. tn is executable in s_I if there is a linearisation $t_1, \ldots, t_n$ of its task identifiers $T(tn)$ such that $\pi = \alpha(tn)(t_1) \ldots \alpha(tn)(t_n)$ is executable in s_I, i.e. such that $\gamma(s_I, \pi)$ is defined.

A solution to an HTN planning problem, i.e. a plan, is a sequence of primitive actions π such that there is a primitive decomposition tn and π is an executable linearisation of tn. We denote the set of solutions to an HTN planning problem $\mathcal{P}$ as $\mathfrak{S}(\mathcal{P})$.

In the presented form, an HTN planning problem does not pose any requirement to the state that is reached after the last task in the solution is executed. This is not a restriction as we can simulate the requirement posed by a goal state. To do so, we introduce a new initial abstract task a_I^* and a primitive goal task g whose preconditions are the desired goal. We then add a decomposition method decomposing a_I^* into a_I and g with $a_I \prec g$, i.e. $a_I^* \rightarrow a_I g$. Thus g will be the last task in any linearisation of a primitive decomposition and g's preconditions, i.e. the state-based goal, will be fulfilled in any solution.

From the standpoint of complexity, finding solutions to an HTN planning problem is a hard task. Erol et al. [R143] showed that determining whether a solution to an HTN planning problem exists is undecidable.

HTN planning problems bear strong similarities to formal grammars. The process of decomposition is very similar to derivation in formal grammars. Consequently proofs for HTN planning domains sometimes rely on concepts from formal grammars. Höller et al. [F36, F42] studied the connection of solutions in HTN planning and formal languages. From a language point-of-view, HTN planning problems correspond to a class of

[3] Here x is an arbitrary symbol.

languages strictly between the context-free and context-sensitive languages [F42] while classical planning problems can express at most regular languages [F36], showing the high expressivity of HTN planning.

As for classical planning, HTN planning problems are usually specified in a lifted formalism where abstract tasks and decomposition methods are specified in a factored way utilising typed variables. To obtain a purely propositional representation, we can ground the domain and perform reachability checks on the methods and abstract tasks. The planning system PANDA (Planning and Acting in a Network Decomposition Architecture [F29, R75, R124]) provides such a "grounder", which was mainly developed by the author of this thesis. Other "grounders" for HTN planning problems are also available [R59, R76], which often produce larger groundings than PANDA due to its more complex reachability analysis. Especially in HTN planning there are also planners that don't perform grounding before the actually planning process but use the domain in a lifted fashion and ground whenever necessary. These planners include, e.g. the well-known planners SHOP and SHOP2 [R118, R135]. This approach is advantageous whenever it is computationally too expensive to fully ground the planning domain. For them it is in contrast also more difficult to define good heuristics.[4]

2.4.2. Extensions and Restrictions to HTN planning

In principle any extension to classical planning, e.g. conditional effects, can also be added to HTN planning. There are also extensions and alterations that are specific to the hierarchical nature of HTN planning. This includes, e.g. Hybrid Planning which adds preconditions and effects to abstract tasks and allows for causal links in decomposition methods [F35, R124]. Several formulations of HTN planning also allow for state constraints to be present in decomposition methods [R48, R143]. These include e.g. before, after, and prevail constraints, which span between tasks and may refer to the states occurring in a solution between them. The most prominent of these constraints are method preconditions [R118, R135], which must be fulfilled in order to a decomposition method to be applicable.[5] Method preconditions can be compiled into additional primitive actions while the other constraints must – at least to our knowledge – be handled in the planner. We will refrain from presenting such extensions as they would only clutter the notation unnecessarily. The techniques presented in this thesis can however handle method preconditions, before, after, and prevail constraints.

In addition to extensions to the presented formalism, there are also several restrictions to the hierarchy that have been investigated in the past. Each of these restrictions forms a subclass of HTN planning problems, which is decidable and thus is worth investigating as more efficient algorithms to solve them might exist. Most notably, if we restrict the partial order $\prec$ in every method to be a total order, planning becomes significantly easier. First, determining plan-existence for totally-ordered HTN planning problems is "only" $\mathbb{EXPTIME}$-complete [R143]. Second, handling them is also easier from an algorithmic and implementation standpoint. This is e.g. witnessed by the large number of specialised planners for them, which includes SHOP [R135], totSAT [C5], Alford's

[4] Note that neither SHOP nor SHOP2 use heuristics, but blind depth-first search.

[5] For the exact semantics of these preconditions we refer to Nau et al. [R118] and **Behnke** et al. [C7].

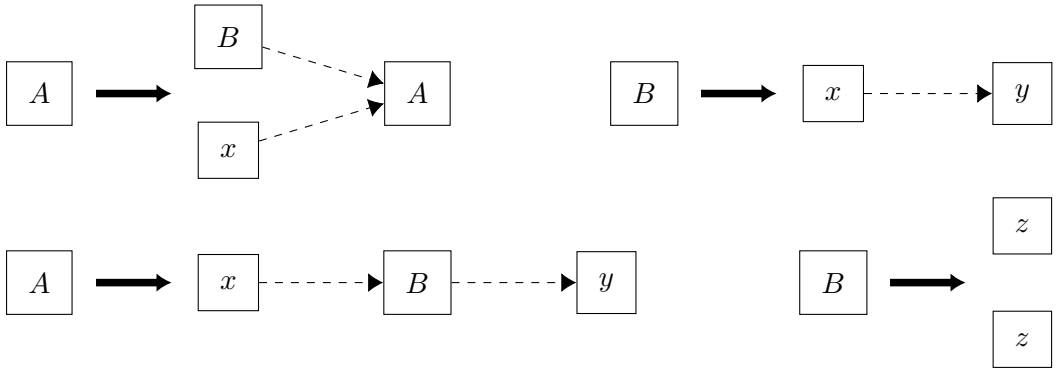


Figure 2.3.: Decomposition methods of a planning problem with abstract tasks $A = \{A, B\}$ and primitive actions $O = \{x, y, z\}$. The initial abstract task is A.

translation [R92], and a planner based on Answer Set Programming [R114]. Theoretical work on HTN planning was concerned with further restrictions, which include

- acyclic planning problems, where the decomposition methods do not allow any recursion through abstract tasks, i.e. it is impossible to derive a task network containing the abstract task a via an arbitrary number of decompositions from a [R143],
- regular planning problems, where each method can contain only a single abstract task and this task must be ordered after all primitive actions in the method [R143],
- tail-recursive planning problems, which forbid certain types of recursion and allow them only through a last abstract task in every method [R68].

Lastly, there is HTN planning with task insertion – or TIHTN – which is not an alteration to the way planning problems are expressed but to the requirements posed to its solution. In TIHTN planning we require a solution – a sequence of primitive actions – to only *contain* the linearisation of a primitive decomposition as a subsequence. The planner is allowed to insert tasks apart from the hierarchy, but still has to decomposed the initial abstract task. This alteration makes HTN planning decidable [R88], more precisely it becomes $\mathbb{NEXPTIME}$-complete [R69]. We will not further consider TIHTN, but note that the planning techniques presented in this thesis can easily be adapted to handle TIHTN planning problems.

2.4.3. Structures for Representing Solutions

To analyse the structure of HTN planning problems and their solutions, several representations and abstractions have been developed. These include e.g. the Task Decomposition Graph, which encodes the decomposition methods of an HTN planning problem in a graph structure [R75].

For this thesis, the most important structures are Decomposition Trees [R88]. They represent the decompositions that were applied to obtain a solution π. In representing decompositions of an HTN planning problem, they are related to Task Decomposition

Trees [R89], Task Decomposition Graphs [F29, R75, R82], and Planning Trees [R134]. These three concepts differ sharply from that of Decomposition Trees, as they represent an abstraction of all possible decompositions in an HTN planning problem. A Decomposition Tree on the other hand represents those decompositions – without any abstraction – that have led to a concrete solution π.

The root of a Decomposition Tree is the initial abstract task a_I. Every task t occurring during the decomposition process is represented by a node v in the Decomposition Tree. Methods in turn correspond to edges, adding an edge from v to child v' for every task in a method's task network. The leafs of the Decomposition Tree then correspond to the task network resulting from applying all decompositions. These leafs are also called the tree's yield.

Definition 6 (Decomposition Tree [R88])
Let $\mathcal{P} = (L, A, O, M, a_I, s_I)$ be an HTN planning problem. A Decomposition Tree (DT) is a 5-tuple $\mathcal{T} = (V, E, \alpha, \beta, \prec)$, where

- *(V, E) is a directed tree with the root node r and inner nodes I.*
- *$\alpha : V \to A \cup O$ labels each node with a task. The label of each inner node is in A, the label of each leaf is in O, and the label of the root node r is a_I, i.e. $\alpha(r) = a_I$.*
- *$\beta : I \to M$ labels each inner node v with a method $\beta(v) = (a, tn_a)$ such that $\alpha(v) = a$. Further v must have $|T(tn_a)|$ children $C = c_1, \ldots, c_{|T(tn_a)|}$ and there must be an isomorphism $\phi_v : T(tn_a) \to C$ such that $\alpha(\phi_v(t_i)) = \alpha(tn_a)(t_i)$ for all $i \in \{1, \ldots, |T(tn_a)|\}$.*
- *$\prec$ is a partial order on the nodes of v. For every inner node v with $\beta(v) = (a, tn_a)$ it contains the order $\phi_v(t_1) \prec \phi_v(t_2)$ if $t_1 \prec_{(tn_a)} t_2$ for any $t_1, t_2 \in T(tn_a)$. It further contains for every order $v_1 \prec v_2$ the order $c_1 \prec v_2$ and $v_1 \prec c_2$ for any $(v_1, c_1), (v_2, c_2) \in E$. Lastly, $\prec$ may not contain any other ordering apart from those implied by these two rules and transitivity.*

The depth of a Decomposition Tree is the depth of (V, E). The yield of a Decomposition Tree is the task network $yield(\mathcal{T}) = (T_y, \alpha_y, \prec_y)$ where

- *T_y is the set of leafs of $\mathcal{T}$,*
- *$\alpha_y(t) = \alpha(t)$, and*
- *$\prec_y = \{(t_1, t_2) \in \alpha \mid t_1, t_2 \in T_y\}$.*

To illustrate the concept of a Decomposition Tree, consider an HTN planning problem with the decomposition methods shown in Figure 2.3. In Figure 2.4, we show a Decomposition Tree for this domain. The yield of this Decomposition Tree consists of its seven leafs and their relative ordering. It is depicted in Figure 2.5. A valid linearisation of this yield would, e.g. be $xxyxzzy$.

Geier and Bercher [R88] showed that every solution π to an HTN planning problem $\mathcal{P}$ is the linearisation of the yield of a Decomposition Tree for $\mathcal{P}$. The contrary is clear by definition as any executable linearisation of $yield(\mathcal{T})$ will be a solution to $\mathcal{P}$.

Theorem 1 (Decomposition Trees and HTN solutions [R88])
Let $\mathcal{P}$ be an HTN planning problem and π a solution to that planning problem.

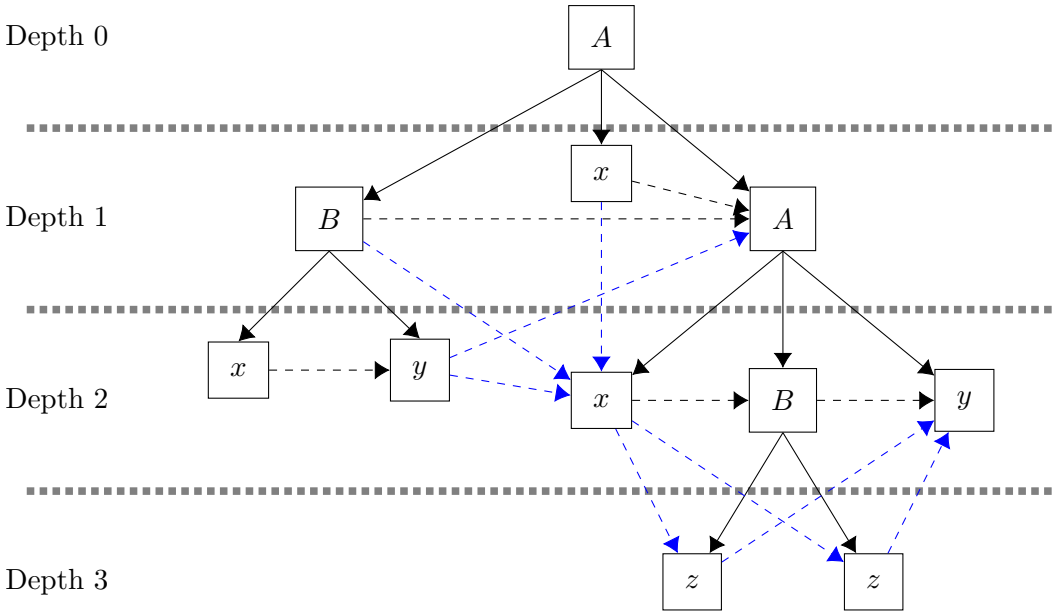


Figure 2.4.: A Decomposition Tree for the HTN planning problem shown in Figure 2.3. Dashed arrows indicate the order $\prec$, where black arrows indicate order contained in the problem's methods and blue order implied via decomposition. Transitively implied ordering constraints are not shown.

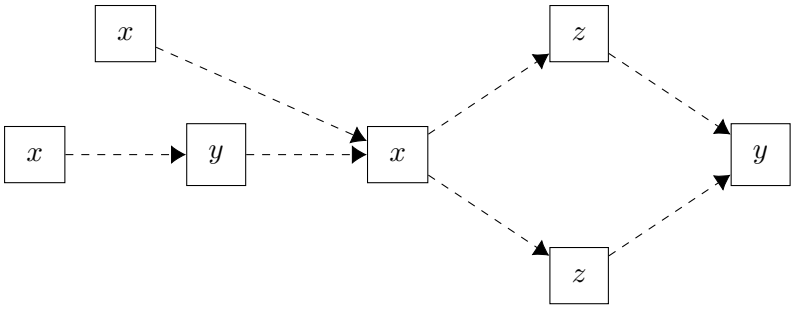


Figure 2.5.: The yield of the Decomposition Tree shown in Figure 2.4.

Then there exists a Decomposition Tree $\mathcal{T}$ such that π is a linearisation of yield($\mathcal{T}$).

2.5. Algorithms for Solving Planning Problems

Due to their computational complexity – classical planning is in general $\mathbb{PSPACE}$-complete and HTN planning is undecidable – there are no generally efficient algorithms, i.e. algorithms that run in (only) polynomial time, for solving these problems.[6] In the past, several techniques for solving planning problems have been developed, which aim at solving them in a practically efficient manner. Practically efficient here means, that problem instances occurring in the real world will be solved within an acceptable time frame. How this time frame looks like depends on application, but it can range from seconds to hours or days – as compared to "more time than the universe has already existed". Witnessed by the increase in performance over the last two decades and their application to real-world problems, these approaches have been widely successful.

The most common way to solve planning problems is via search. This search is either performed in the space of plans or in the space of states (respectively the space of progressions for HTN planning problems [F20, R118]). In plan space search, search is performed over incomplete plans and successor nodes are created by applying modifications to them [R150]. Modifications aim at removing a flaw from the plan, i.e. a property that prohibits the incomplete plan from being a solution, e.g. an unfulfilled precondition. The search ends as soon as a plan without a flaw is found. In state space search, the nodes of the search are states, starting from the initial state. Successors are generated by applying applicable actions to the current state. The search terminates if a state has been reached in which the goal is true. Both plan space and state space search can be improved with a variety of techniques, e.g. heuristic search using A* or greedy search, pruning techniques, (at least state space search) by bidirectional search, or orbit search.

Another search-based technique, which differs in its general principles from state space and plan space search, is symbolic search. Here one considers the set of all states with a distance of at most k units from the initial state. These are represent by a Binary Decision Diagram (BDD). To compute the same set for the distance $k + 1$, we can apply operations to the BDD. This allows for considering all reachable states at once, thus speeding up the computation. Interestingly, this technique can be extended to incorporate some types of heuristics [R66].

Lastly, there are planning approaches that transform the planning problem into another equivalent problem and use ready-to-use solvers for that problem. Any approach that solves planning via translation into another problem has one fundamental advantage compared to search-based techniques: it benefits automatically from progress made in solving the target problem. This allows for improvement of planner performance without changing the planner at all. The best known type of these approaches is planning via SAT – the satisfiability problem for propositional logic.

Definition 7 (SAT)
Given a propositional formula $\mathcal{F}$ in conjunctive normal form (CNF), SAT is the problem

[6] However there might be for classical planning if $\mathbb{P} = \mathbb{PSPACE}$, which is fairly unlikely.

of determining whether a valuation β that satisfies $\mathcal{F}$ exists, i.e. $\beta \models \mathcal{F}$, and if so to output β.

The problem SAT is $\mathbb{NP}$-complete [R157]. The core idea in SAT-based planning is to construct a propositional formula that is satisfiable if and only if the planning problem has a solution. Based on the satisfying valuation β, it is then possible to extract that solution. Even for classical planning there is a slight theoretical problem: while SAT is $\mathbb{NP}$-complete, classical planning is $\mathbb{PSPACE}$-complete. As such, we can't expect that there is such a transformation of polynomial size.[7] To circumvent this problem and allow for SAT-based planning nevertheless, we are not constructing a propositional formula that is satisfiable if the problem has any solution, but only if it has a solution with specific properties, such as adherence to some bound. We then choose this property such that the plan existence problem becomes $\mathbb{NP}$-complete. Completeness is usually achieved by an iteration over the bound.

SAT-based planners draw from the high efficiency of modern SAT solvers. Translations into propositional logic are also practical for extensions, as their encodings are usually easily extendable, e.g. to handle Linear Temporal Logic [R96]. Further, using SAT Modulo Theory solvers even allows to support numerical state variables and time [R62]. This could even be extended to cover Planning Modulo Theories that, e.g. allow for complex geometrical reasoning in planning actions [R83].

For classical planning, the typical restriction posed on the plans described in the propositional formula is a length bound [R145]. Given a length bound ℓ, we construct a formula that is satisfiable if and only if a plan of length at most ℓ exists. If we assume a unary encoding[8] of ℓ the plan existence problem becomes $\mathbb{NP}$-complete.

The encoding is based on representing the plan as a sequences of timesteps. For each timestep t ($0 \leq t \leq \ell$) and proposition symbol p, the decision variable $p@t$ represents that p is true in the state at time t [R145]. Similarly, for every timestep t ($0 \leq t < \ell$) and action a, the decision variable $a@t$ represents that a is executed at time t. The propositional formula now asserts that

1. at $t = 0$ the decision variables $p@0$ exactly represent the initial state,
2. the goal is true at $t = \ell$,
3. the preconditions of each applied action are fulfilled in the state in which it is executed,
4. the effects of each applied action are executed,
5. proposition symbols not affected by applied actions retain their truth from state to state, and
6. at most one action can be executed at each timestep.

The commonly used translation of these rules into a propositional formula was introduced by Kautz and Selman [R145]. In order to obtain a complete planner based on this encoding, we have to call the SAT solver multiple times. We start with length bound

[7] Again, unless $\mathbb{NP} = \mathbb{PSPACE}$.

[8] Note: from a complexity theory point-of-view, this is not an acceptable assumption. However since we are interested in the *practical* usability of the approach alone, it is acceptable to use an encoding that is polynomial in ℓ but exponential in $\log \ell$.

$\ell = 1$, construct the formula, and run the solver. If it returns a satisfying valuation, we can extract the plan. If the solver shows that there is no satisfying valuation, we know that any plan for the problem must have at least the length 2. We then increment ℓ by one an repeat the process. Several improvements to this simple scheme have been proposed. This includes different step sizes, i.e. increases by more than one [R78], the interleaved execution of SAT solvers [R104], and the use of incremental SAT solving [R58].

In addition to technical improvements in the evaluation of the formula, there has also been progress in making the formula more amenable to SAT solvers. Most of the clauses resulting from encoding the assertions 1-5 are Horn clauses, i.e. they contain at most one positive literal. The only exception are those clauses that pertain to negative preconditions and delete effects. Horn clauses can be treated efficiently by SAT solvers and a formula containing only Horn clauses can be solved in polynomial time. Encoding the assertion 6 breaks this pattern – which seems somewhat unnatural, as does the fifth axiom of geometry [C161, Book I, Proposition V]. The constraint can, however, not be dropped entirely. Doing so would allow for executing actions in parallel as long as their preconditions are fulfilled in the same state. Consider – as an example – two action, both having $a \wedge b$ as their precondition and one having $\neg a$, the other $\neg b$ as its effect. Both actions are applicable in the state $\{a, b\}$ but executing one necessarily disables the execution of the other. It is not possible to sequentialise them such that the resulting plan would be executable.

Already the encoding of Kautz and Selman [R145] accounts for a limited amount of safe parallelism, nowadays called $\forall$-step parallelism [R104]. As of now, the encoding based on the $\exists$-step semantics for parallel execution of actions allows for the most parallelism [R104]. Its aim is to allow parallel execution of actions whenever a linearisation of the actions exists that is executable. Deciding this property is unfortunately $\mathbb{NP}$-complete [R104]. Rintanen et al. [R104] presented an approximation that pre-computes an ordering of all actions and allows any subset of actions to be executed in parallel if they are executable in this precomputed order. This property can be compactly encoded into propositional logic via *chains* [R104]. For both encoding chains and determining an appropriate order of all tasks, the Disabling Graph (DG) is used. The DG's nodes are the actions and an edge (a_1, a_2) denotes that executing a_1 may falsify a_2's preconditions. If the DG is acyclic, it is safe to execute all actions in a reverse topological ordering, as none will disable an action occurring later on in the order. If the DG contains cycles, actions are executed in reverse topological order of the strongly connected components and in random order within the components.

Most planners for HTN planning problems used some fashion of blind search, e.g. SHOP and SHOP2. Recently, search-based HTN planners began to incorporate heuristics, e.g. PANDA [F20, F29, R75], and pruning techniques, e.g. FAPE [R61, R76], which provided significant improvements in solving time for HTN planning problems.

Similar to classical planning, translation-based approaches for HTN planning have also been introduced in the past. For totally-ordered HTN planning problems, Dix et al. proposed a translation into Answer Set Programming [R114]. Alford et al. [F34, R92] showed that HTN planning problems can be translated into classical planning problems.

Similar to the encoding of classical planning into propositional logic, their translation requires a bound. In this case, the bound is the so-called progression bound – the maximum size of an intermediate task network under progression. For tail-recursive HTN planning problems, an upper progression bound can always be computed in polynomial time, i.e. a bound k such that any solution has a progression trace where the largest intermediate task network has size k. For arbitrary HTN planning problems, this bound cannot exist – else we would have obtained a decision procedure for an undecidable problem. Mali and Kambhampati [R138] proposed an encoding of hierarchical planning into propositional logic. Their notion of hierarchical planning, however, differs significantly from the presented, and by now established, HTN formalism. Notably, their formalism does not include an initial abstract task, but specifies the aim in terms of a goal state. As such, abstract tasks can be freely inserted into the plan. Such planning problems are classified as decompositional planning, where abstract tasks and the hierarchy do not specify the problem, but assist in solving it [R73, R137, R149]. Further they allow for task-sharing [R60], i.e. a task that has been derived twice via decomposition must be executed only once. Lastly, they assume that the decomposition hierarchy is acyclic. These differences make their encoding not useable for HTN planning problems. To the best of the knowledge of the author, there is no encoding of (standard) HTN planning into propositional logic prior to this work.

2.6. Linear Temporal Logic

Parts of this thesis also deal with handling planning in conjunction with (finite) Linear Temporal Logic (LTL) [R155]. LTL is a modal extension of classical propositional logic. It is defined based on a set of primitive propositions A which by itself can be formulated in any complex logical language. For the purposes of planning, we use the propositional logic over the set of proposition symbols L and primitive actions O. LTL makes statements over sequences of states over which propositions in A can be evaluated, i.e. we assume a discrete view on time. Such a sequence of states is also called a trace. In the case of planning, a state in the LTL sense is a state plus the action which is executed in it. For the last state of a plan there is no such action. Technically, we can add a dummy action `end` that signifies the end of the plan.

Based on the set A, we define the following expressions as syntactically valid LTL formulae T:

$$a \in A \mid t_1 \wedge t_2 \mid t_1 \vee t_2 \mid \neg t \mid Xt \mid Et \mid Gt \mid t_1 U t_2$$

The logical connectors $\neg$, $\wedge$, and $\vee$ have the usual semantics. X, E, G, and U are called temporal operators. The temporal operator X refers to the next timestep and E to some point of time in the future. Gt denotes that t should hold from now on forever and $t_1 U t_2$ denotes that t_1 holds until t_2 holds for the first time or that it will eventually hold.[9] Conceptually, E corresponds to the modal operator $\diamond$ and G to $\Box$. More formally, a temporal formula T is evaluated over a sequence of states $s_1, \ldots, s_n$. The truth of a

[9] LTL is usually formulated over infinite traces were $t_1 U t_2$ is also satisfied if t_1 holds forever.

formula $[[\phi]](s_1, \ldots, s_n)$ is defined as:

$$
\begin{aligned}
[[a]](s_1, \ldots, s_n) &= s_1 \models a && \text{iff } a \in A \\
[[f \wedge g]](s_1, \ldots, s_n) &= [[f]](s_1, \ldots, s_n) \wedge [[g]](s_1, \ldots, s_n) \\
[[f \vee g]](s_1, \ldots, s_n) &= [[f]](s_1, \ldots, s_n) \vee [[g]](s_1, \ldots, s_n) \\
[[\neg f]](s_1, \ldots, s_n) &= \neg [[f]](s_1, \ldots, s_n) \\
[[Xf]](s_1, \ldots, s_n) &= [[f]](s_2, \ldots, s_n) && \text{iff } n > 1 \\
[[Xf]](s_1, \ldots, s_n) &= \bot && \text{iff } n = 1 \\
[[Ef]](s_1, \ldots, s_n) &= [[f]](s_1, \ldots, s_n) \vee [[f]](s_2, \ldots, s_n) && \text{iff } n > 1 \\
[[Ef]](s_1, \ldots, s_n) &= [[f]](s_1) && \text{iff } n = 1 \\
[[Gf]](s_1, \ldots, s_n) &= [[f]](s_1, \ldots, s_n) \wedge [[f]](s_2, \ldots, s_n) && \text{iff } n > 1 \\
[[Gf]](s_1, \ldots, s_n) &= [[f]](s_1) && \text{iff } n = 1 \\
[[fUg]](s_1, \ldots, s_n) &= [[g]](s_1, \ldots, s_n) \vee ([[f]](s_1, \ldots, s_n) \wedge [[fUg]](s_2, \ldots, s_n)) && \text{iff } n > 1 \\
[[fUg]](s_1, \ldots, s_n) &= [[g]](s_1) && \text{iff } n = 1
\end{aligned}
$$

For simplicity, we denote for a plan π and an LTL formula ϕ with $[[\phi]](\pi)$ the truth of ϕ over the state trace $(s_1, \ldots, s_n)$ induced by π, i.e. $[[\phi]](s_1, \ldots, s_n)$.

3. From Blackbox to Whitebox Planning

3.1. Summary

Planning-based systems provide a wide range of useful assistance by exploiting a planner's abilities to quickly and effectively adapt to new, unforeseen, and challenging problems. Based on a generated plan, these systems can recommend a course of action to their human users. We start our investigation by surveying some of the existing techniques for providing such assistance. Some of them simply present the planner's solution to the user, without allowing the user any influence over it, i.e. act in a blackbox fashion. In contrast, *Mixed-Initiative Planning Systems* enable the user to exert influence and control over the planning process by making the planner's decision making process collaborative. This way, they allow the user to better understand and accept the planner's decisions, thus improve his satisfaction with the assistance system, and transform the assistant into a whitebox system.

We will identify which capabilities a planner must possess in order to be used successfully within a mixed-initiative planning-based assistant. First, the planner must be able to find plans as quickly as possible in order to create a system that is reactive. Second, the planner must be able to change a once found plan in accordance with the instructions of the user.

To understand the difficulties involved in changing plans with respect to such requests, we study the computational complexity of the involved decision problems. As a first step in the analysis, we show that HTN plan verification is $\mathbb{NP}$-complete [C9]. Next, we show that even the most elementary requests to change a plan (adding, removing, exchanging, re-ordering actions) are as difficult as planning itself [C8]. To not only cover these elementary types of requests, but general ones posed by the user, a suitable representation is required. We argue that Linear Temporal Logic (LTL) is a suitable formalism to express the requests posed by the user to change plans. Lastly, we show that the computational complexity of changing plans in accordance with a given LTL formula is as hard as the elementary requests. This investigation is – to the best of our knowledge – the first complexity-theoretic analysis related to mixed-initiative hierarchical planning.

The theoretical results of this chapter – which pertain to the second requirement posed to a planner – have led to the development of an HTN planner that currently outperforms all its competitors (see Chapter 4) and thus also address the first requirement.

Core publications described in this chapter

[C8] **Gregor Behnke**, Daniel Höller, Pascal Bercher, and Susanne Biundo. "Change the Plan – How hard can that be?" *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016).* AAAI Press, 2016, pp. 38–46

[C9] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition". *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015).* AAAI Press, 2015, pp. 25–33

3.2. Planning-based Assistance

Planning-based systems have been deployed in a wide variety of scenarios over the last three decades, especially in those practical applications where there is the need for solving moderately to highly complex combinatorial problems. Planning-based assistance systems provide suitable help in these scenarios, by quickly, effectively, and automatically solving these combinatorial problems for the system's users.

In any deployed planning-based system a human user or operator has – at least in some way – contact to the system and the planner. In some scenarios this contact is tangent, e.g. in logistics [R86] or automated greenhouses [R90], where the planner achieves some technical objective that is of interested to its human user. In these technical types of scenarios, it is generally sufficient to simply find a plan which is subsequently executed without further user of the planner.[1]

In contrast, there is a multitude of scenarios in which planners provide assistance directly to their users. This includes e.g. composing a fitness training plan [F27], figuring out how to connect the devices of a home theatre system [F40, R74], or determining how to use electric tools to complete a Do-It-Yourself home improvement project [C3]. The planner alleviates its user from the tedious and difficult task of determining a goal-leading course of action in a complex environment. Instead, the users can focus on higher level activities, on controlling the system, or on tasks that can only be performed by humans. There are several scenarios in which planners have been successfully used in this way. They however differ in the way the user and the planner interact and in the way in which the user has influence and control over the planning process. In this section we will survey some of the existing planning systems that assist their human users.

[1] Note that in domains with a more complex physical environment, a coupling between planning and acting might be necessary [R45].

3.2.1. Black-box Planning

Some planning-based assistance systems treat the planner as a black-box system, i.e. as a system into which inspection is not possible and whose inner workings cannot be analysed, modified, or influenced. For example Bercher et al. [F43, F40, R74] presented a planning-based assistant which supports a human user in setting up a home theatre system. Based on the currently available devices, cables, and connectors, the assistant finds a plan to connect all devices appropriately. The plan is transformed into a sequence of instructions and presented to the user. The user, however, cannot influence the generated plan in any way. The only circumstance under which the assistant is able to alter its plan is if the execution of a plan step has failed. Then the assistant will adapt its plan to the changed situation using plan repair.

The XPLAN system [R54] automatically plans laboratory experiments for tests of synthetic biology. Its plans are executed by lab assistants and the results are used by bio-engineers. XPLAN's users solely provide the input for the planner and execute the produced plan. Interaction with the planner is not possible.

Assistance systems like Bercher et al.'s home theatre assistant or XPLAN base their assistance on providing the user with a set of instructions (or performing them themselves) that solve the user's current problem. Their core assumption is that the user will be satisfied with the plan – no matter how it looks like – provided that it solves the user's problem. They lack the ability to react to a user, who might be dissatisfied with the produced plan, even though it solves his problem. As a result of their dissatisfaction, users might abort individual interactions with the assistant or stop using it entirely.

3.2.2. White-box Planning

While some assistance systems use planners in a black-box fashion, many practical applications require that the planner is (in some way) exposed to the user. This results in a white-box planning-based assistant. The planner should allow the user to exert influence and, if possible, even control over the planning process. Thus, the planner should take the user's wishes into account and, if possible, let the user shape the planning process to its will. The assumption is that a user with control over the planning process will be satisfied with its results [F26]. Planning that allows for such an integration of the user into the planning process is called *mixed-initiative planning* [R85].

One of the first mixed-initiative planning systems was TRAINS [R144], which assisted in planning for a transportation task. Its main focus was not on the mechanics of planning necessary for such a system, but on dialogue and interactional aspects. Interaction focusses on jointly altering a current – unfinished – plan until it achieves the goal of the transportation task, as well as satisfying the user. The planner in TRAINS was a handcrafted path-finding algorithm written specifically for this domain. Later, TRAINS was extended to TRIPS [R136], which demonstrates how to assist human crisis-managers by jointly developing plans for disaster relief, e.g. for an evacuation. Like TRAINS, TRIPS focusses on interactional aspects of mixed-initiative planning as well as questions of architecture, while using specialised programs to solve the planning aspects of the domain.

Notably, they propose to separate the components of a mixed-initiative planning systems into three parts: input/output components, a dialogue component, and a reasoning component. The assistant we present in Sec. 5.3 uses the same architectural principles.

Mixed-Initiative Cased-Based Planning (MI-CBP) [R142] was – to the best of the author's knowledge – the first mixed-initiative planning system that was applicable to general planning problems and not specialised to a single domain. It bases its user interaction upon case-based reasoning and not upon search as used by most modern planners. Whenever the user inputs a new goal into MI-CBP, it uses its case-based reasoner to retrieve a plan for a similar situation contained in its knowledge base. The user is then asked to adapt this retrieved plan such that it will solve the problem at hand. MI-CBP supports the user in this by providing him with plausible modifications.

SIPE-2 is a domain independent plan-space planning system that supports user interaction during the planning process [R133]. In SIPE-2, the user can take full control of the planning process and subsequently performs all decisions (adding actions, ordering, causal relations) during the planning process himself. In addition, the user can hand control back to SIPE-2, which continues the planning process until a solution has been found without any opportunity for the user to take control again. When continuing the planning process, SIPE-2 will refine the partial plan the user has constructed, i.e. it will adhere to the user's prior instructions. In essence, SIPE-2 allows the user to perform parts of the planning process manually and provides the user with the option of completing the constructed partial plan automatically.

PASSAT is a mixed-initiative planning system solving HTN planning problems and consequently uses a general HTN planner in its core [R122]. PASSAT's main application domain is the planning of missions of military special operations forces. It presents its user during interaction with a current, unfinished decomposition tree and points out which tasks, i.e. leafs, in the tree must be refined further, as well as causal errors in the plan. PASSAT's user can either refine an undecomposed task or use a plan-repair mechanism to resolve causal errors. Lastly, PASSAT offers the ability for plan sketching [R117], where the user provides tasks (primitive and abstract) that must be contained in the final plan. PASSAT determines which refinements of the initial abstract task can suitably be used to generate a plan containing these tasks.[2] This is designed to combine top-down and bottom-up planning techniques. PASSAT further assumes that its domain model is not necessarily correct or complete and thus allows the user to change it by dropping or adding constraints or tasks.

Nau et al. [R131] created a planning-based system that assists engineers in creating microwave modules (transmitters and receivers). The planner generates a "Process Plan", i.e. a plan determining how a specific part is manufactured and added to the microwave module. Here, the planner does not generate plans for assembling the whole module, but only for individual parts to be added to the module. It further not only generates a single plan, but multiple alternative plans for adding a specific part to the module. The engineer can then select for each of the parts a plan to assemble it and thus can create a plan for assembling the whole module. He can further exchange parts for other parts fulfilling an equivalent functionality. In this, the user is assisted by a trade-off analysis

[2] This problem bears strong similarities to HTN plan recognition [F19].

module, which helps him to optimise his selection of sub-plans – in order to create a final module and a full production plan for it.

Muñoz-Avila et al. [R129] presented the mixed-initiative planning system SiN. As PASSAT, it is based on HTN planning and focusses on the military application of non-combatant evacuation. SiN combines the HTN planner SHOP [R135] with NaCoDAE [R141]. NoCoDAE uses a case-based reasoner to retrieve cases matching the user's description. The focus of SiN are domains with incomplete knowledge – which are very common in military applications due to unknown situation in enemy territory [R162, Chapter 12]. SiN assumes that SHOP cannot find a solution to the posed problem based on the information it has (i.e. the known state). Interaction with the user is initiated whenever a situation occurs in which SHOP cannot continue its search without backtracking. In most cases, this is the case if an abstract task which SHOP cannot decompose[3] has been reached. Control of the planning process is then ceded to NaCoDAE which initiates a dialogue with the user, who decides how to proceed. NaCoDAE holds a set of standard operating procedures, one of which is selected via case-based reasoning from the user's answers to NaCoDAEs questions. This procedure is interpreted as a decomposition method and executed, allowing SHOP to resume planning.

TRAINS/TRIPS, MI-CBP, SIPE-2, and PASSAT have one central feature in common: all of them integrate the user into the process of *finding* a solution. In them, the user asserts control over the planning process by taking decisions a black-box planner would perform during search. As a result, these mixed-initiative planners present incomplete plans to their users and ask how they should be modified. The user is then asked for his opinions on possible modifications and/or is allowed to instruct the planner to perform more or less complex operations on the plan, e.g. "send the trains from Chicago to Detroit" in TRAINS [R144]. Once a solution is found, the planning process is finished and it is assumed that the user will be satisfied with this plan. We call this type of integration of the user into the planning process *search-based*.

In contrast, the probably best known mixed-initiative planning system MAPGEN [R100, R112] uses a different approach. MAPGEN was designed to create the plan for operating NASA's Mars rovers Spirit and Opportunity. ENSEMBLE, the successor of MAPGEN, is used to create the operation plans for the rover Curiosity [R100]. MAPGEN's users are always presented with a causally sound plan, i.e. a plan that satisfies all flight and mission rules. These plans could be used as operation plans for the rover, i.e. are solutions to the planning problem at hand. The main objective of MAPGEN is to assist in the mediation between different groups of scientists with different objectives. Each of them has a set of scientific tasks to perform, but it is never possible to perform all of them in a single day. Thus MAPGEN proposes a selection, which can afterwards be altered by the scientists to achieve scientifically optimal results. MAPGEN allows both for manual manipulation of the plan – after which it further modifies the plan to ensure causal correctness, as well as posing additional constraints, which are then automatically fulfilled by the planner. These constraints can include additional goal, as well as temporal constraints over the plan. To be able to cope with these manipulations and inputs from the user, MAPGEN uses an efficient temporal planner at its core. The

[3]SHOP's domains can contain method preconditions which may restrict the applicability of decomposition methods.

planning model itself is however classical, i.e. non-hierarchical.

MAPGEN interacts with its users only based on solutions and allows its users to move from solution to solution via modifications of the plan or instructions to the planner. Here the planning process is finished once a solution is found that is satisfactory to MAPGEN's users. We call this style of interaction *solution-based*. The distinction between search-based integration and solution-based integration was already observed by Muñoz-Avila et al. [R129].

We presume that with the advent of highly efficient planning system like FF [R128] and Fast Downward [R101] for classical planning, and e.g. PANDA [C2, F20, F29] for hierarchical planning, solution-based interaction will become more popular. The main disadvantage of search-based interaction is that it forces the user to consider deeply planning-related questions, e.g. how and why a causal error occurs, which modifications to a plan are allowed, and has to understand partially completed plans. Notably, search-based interaction might (has to) present causally flawed plans to the user, which is hard for him to understand and must be remedied [F32, F41]. In contrast, solution-based interaction always presents complete and causally correct plans to the user. Using completed plans, it is also (presumably) easier to explain the rationale behind the plan to the user in order to help the user to better understand it. Lastly, solution-based interaction focusses on the *final* plan that the user will execute or be assisted with. This allows him to specifically address issues he has with the actual final plan, instead of issues with intermediate uncompleted plans.

As we deem solution-based interaction strategies more promising for future applications, we focus in this thesis on techniques for mixed-initiative planning that uses a solution-based interaction. The techniques we present are however also applicable for search-based interaction techniques.

3.3. The Planner in Mixed-Initiative Planning

Mixed-initiative planning is, on the system's side, not enabled by the abilities of a single system component. Interaction, dialogue, and planning components must be able to handle the tasks they have to fulfil in a mixed-initiative planning setting. Rightly, a lot of focus was laid on interactional and dialogical aspects of mixed-initiative planning [F32, F41, R136, R144], as without them humans will refrain from using mixed-initiative planners due to their unusable interaction behaviour. In this thesis, we want to focus on the tasks the planner itself has to perform in a mixed-initiative planning setting. For that we start by discussing the objectives of the planner in both search-based and solution-based mixed-initiative interactions.

For search-based interaction, we argue that a depth-first search style interaction with the user is necessary in order to keep him attentive and not to overstrain his cognitive load [F32, F41]. Such an interaction strategy is e.g. used by PASSAT [R122]. When using a depth-first-style interaction, a severe problem arises: dead ends in the search space. Dead ends are states of the search from which it is not possible to reach a goal or a causally completed and error-free plan. If the user controls the search, he might

inadvertently modify the plan in an way that precludes it from being a solution. There are two ways of solving this problem. We either allow the user to "back-up" from such a dead end plan, or we prohibit the user from ever choosing an option that will lead to one.

For the former, we would allow the user to perform any modification until an unresolvable flaw is present in the current plan. If so, the planner has to notify the user – through the dialogue manager – and ask for instructions. In the simplest case, the user will ask for a plan that is solvable to be presented. At this point, we have to ensure expectation conformity to keep the user attentive. This means that we have to find a solution that is as similar as possible to the current plan, but can still be refined into a solution [F32, F41]. The planner can e.g. solve this problem by planning while optimising a plan-distance metric.[4] As such, the requirement to the planner is to find solutions quickly, i.e. to be fast.

If we use the second means to avoid the problems of a depth-first search-based integration, we cannot allow the user to perform changes to the plan that would lead to a dead-end state. For that, we have to be able to determine whether a given plan can be refined into a solution. Answering this question is in its core a planning problem. In the interaction, we would have to be able to perform these tests in (near) real-time, i.e. a fast planner is required. We constructed a planning-based assistant – SLOTH – using this interaction scheme [F27, F32]. SLOTH assists its user in planning an individualised workout for a given set of training objectives. In it, we perform a depth-first search and present the user with options to refine his current plan at every step. We use the planner to exclude all refinements that cannot lead to solutions, e.g. due to violation of general training rules.

In both search-based and solution-based interaction, users will commonly request alterations to the plan. For solution-based interaction, requesting to *change* the plan is the main type of request posed to the planner. In search-based interaction, the user is usually presented with a set of possible modifications (or refinements) to the plan. These modifications are usually those that the planner itself would perform to reach the next search node. Applying these presented modifications can be easily implemented in the planner, as it only has to apply them. But the user – on his own initiative – might instruct the planner to perform a more complex *change* to the plan. In both cases, the planner must be able to find a plan that satisfies the user's request.[5] Thus as a minimum requirement, the planner must be able to find plans that satisfy a given request from the user. If so, we can again optimise a distance metric from the previously shown plan in order to ensure expectation-conforming changes. Answering the requests of a user again requires to have a planner that produces solutions quickly – i.e. a fast planner – as the modification has to be performed within a time-frame acceptable to the user.

In SLOTH, we added a rudimentary handling of these requests, namely only a single type: exchanging exercises for other exercises. Note that this might imply to change more than one exercise as the resulting plan must adhere to the general fitness training rules (e.g. that a trained muscle must be warmed-up first).

[4] For a SAT-based planner one would binary-search the optimum. See Section 4.5.

[5] Or be able to explain why this is not possible. This is an avenue for future research.

To sum up, for any kind of mixed-initiative interaction, we need a fast planner and one that is able to find plans satisfying a user's requests to change a plan. We will address the need for a fast HTN planner in Section 4. In it, we will show how HTN planning problems can be translated into propositional logic. Using this translation, we can obtain the currently fastest known HTN planner.

3.4. Changing Plans

As we have stated, one key ability of the planner in a mixed-initiative planning system is to alter a plan according to the user's instructions. When enabling the planner to handle these instructions, we face two problems: (1) how do these instructions look like, i.e. how can they be formalised and (2) how can the planner find plans adhering to these formalised instructions?

3.4.1. Requests to Change a Plan

We start by studying potential requests uttered by users [C8]. As a first characteristic of any request, it describes the change that the user wants to see implemented to the plan by the planner. At first glance, we can distinguish four general types of these changes, each pertaining to a mandated change to the actions contained in a current plan presented to the user.

- Adding an action or actions to the current plan
- Removing an action or actions from the current plan
- Replacing an action in the current plan by others
- Reordering an action in the current plan

Naturally, users might not restrict their instructions to these four simple patterns. We however argue that they are prototypical for more complex requests made by users and are thus worth studying. We will discuss how more complex requests to change a plan can be interpreted and represented formally in Section 3.4.2.

If we consider these four requests, applying them to a given plan seems to be rather trivial, as we "only" have to add, remove, replace, or reorder actions. At a closer look, there is a problem with this simple view on implementing these four requests. If the user instructs a mixed-initiative planning system to change a plan presented to him, he has certain expectations on how the result looks like and which properties it must possess. For a solution-based interaction, we obviously have to ensure that the changed plan is still a solution to the planning problem and e.g. not causally flawed, or is missing actions enforced to be in the plan via the HTN domain structure. For a search-based interaction, the planner should likewise find a plan that both fulfils the request and can still be refined into a solution. If the planner would merely implement the requested change, there would be a risk that the new plan cannot be refined into a solution, i.e. is a dead-end in the search space. Presenting such a plan to the user would demonstrate incompetence of the planner to the user, as he might think that the planner should

have seen that the new plan is a dead-end and should have acted accordingly. As such, independent of the interaction strategy, the planner must ensure that the changed plan is either a solution or can still be refined into a solution.

Further, the change requested by the user might still allow for a plan to be found, but the plan might be significantly longer or its actions may be more complicated or more dangerous to be performed. In these cases, the planner should be able to inform the assistant's user about this fact and may even advise him to not use the altered plan, but the original one instead. This would, to be persuasive for the user, require an explanation why e.g. a shorter or less complicated solution cannot be found when adhering to the user's request. Such explanations are out of the scope of this thesis and may be considered in future work.

Thus the planner has to be allowed to alter the plan apart from the change directly requested to by the user. To see why this is necessary, consider as an example a plan for sawing a wooden plank into two pieces using a hand saw. This plan may contain the action `saw_by_hand(...)`. Now a user might want to alter the plan so that the plank is sawn using an electric saw instead of manually. If the planner solely replaces `saw_by_hand(...)` with the corresponding action `saw_with_saw(...)` the plan will contain three types of flaws. First, if we only replace the action, the electric saw might not be set up correctly due to missing actions, e.g. those inserting batteries or setting the saws switches to the correct positions. Second, the plan might still contain actions that pertain to the manual saw, e.g. those "setting it up" by inserting a sawing blade or by attaching the plank in a specific way to the table. These actions are superfluous and should be removed, else the planner might be viewed as incompetent. Third, using an electric saw instead of a manual one might influence other actions in the plan. For example we might have to sand the cut edge which was not necessary when using a manual saw. Also if the user has only a single battery, the battery has to be removed from the saw and has to attached to the drill if it is used afterwards, which was not necessary before. This third type of changes, i.e. those necessary to apparently unrelated parts of the plan, becomes even more apparent if we consider a transportation scenario with delivery-deadlines. If we change the route at one point in time, we might have to alter the remaining parts of the route to ensure the deadlines are met.

The reason for these flaws is that the user's request has been implemented word-by-word, i.e. if the user instructs the planner to replace action A with action B, we assumed that the planner does exactly this operation. Note that executing these kinds of requests seems trivial, which we will show is not the case for HTN planning in Section 3.5.2. In order to prevent the above shown flaws in plans, the planner has to be allowed to perform additional changes to the plan next to the change the user requested explicitly. Which and how many of these additional changes are allowed thus forms the second dimension of a user's request.

In practice, we would most of the time like to optimise a distance metric between the plan to which a change should be applied and the plan resulting from that change. Naturally, one might imagine even more complex models of which changes can be allowed. For example, one might associate different costs with changing different actions, or penalises changes that occur "near" to each other in the plan less than changes that

are unconnected to each other. Further, users will typically not state this secondary requirement for a change explicitly. Instead, the planner will have to make assumptions based on the domain and the current situation which metric should be optimised, if any. We deem this question to be out of scope of this thesis and have thus not investigated it further.

In order to study these optimisation problems theoretically, we can instead study decision-theoretic versions of the optimisation. From the computation-theoretical perspective, we can distinguish three types of additionally allowed changes [C8]:

- No additional changes are allowed
- Up to k additional changes are allowed
- Any additional changes are allowed

Notably, minimising the number of changes is related to the second restriction, as answering it is necessary for optimisation. Thus studying the second type of restriction is the computation theoretic way of studying the question of minimising the number of changes.

3.4.2. Complex User requests

In the previous section, we noted that one might imagine requests to change a current plan that pose more complex requirements than those four basic typed we presented. For example, the user in a DIY setting might ask the planner to ensure that whenever he cuts a surface it should be sanded afterwards. We could handle this constraint by translating it into several requests to add the `sand(...)` action for every cut surface. This kind of treatment has however two significant disadvantages. First, we have to keep track of this constraint when further changes to the plan are made as we have to insert sanding actions if we are cutting additional surfaces. Second, we would have to develop such a specialised treatment (i.e. translation into elementary requests) for every possible request uttered by the user. This is highly inefficient and, due to its inevitable redundancy, prone to errors.

As such, it would be beneficial to have a unified representation for all requests that might be posed by the user. This representation would enable handling all requests in a uniform way by a single algorithm and thus removing redundancy. Further, adhering to previously posed constraints when handling subsequent requests by the user is easier, as all of them have the same format and can be more easily integrated.

The unified representation of change requests should be able to express arbitrary logical connections between both actions and state features. Further, the user should be able to pose requirements with respect to the order of actions and their dependency over time. Linear temporal logic (LTL) [R155] is a suitable formalism to express these kinds of restrictions. LTL has been used in wide variety of scenarios to formulate and formalise restrictions to processes with timing aspects such as workflow management [R103]. Further, model checking systems like NuSMV [R119] use LTL to specify desired and undesired system behaviour. Notably, LTL is used to specify the semantics of plan constraints and preferences in PDDL, which is the standard description language for classical plan-

ning problems [R109].

Consider the simple types of requests mentioned in the previous section. Each of them can be easily transformed into an LTL formula. Adding an action a corresponds to the LTL requirement Ea. Removing an action a corresponds to $G\neg a$, changing an action a to b corresponds to $Eb \wedge G\neg a$, and ordering action a before b to $E(a \wedge XEb)$.

Unfortunately, it is highly improbable that the users of a mixed-initiative planning system will be able to formulate their requests to the system in terms of an LTL formula they would like to see fulfilled. As such, the restrictions formulated by the user in natural language have to be converted into corresponding LTL formulae. Nikora and Balcom [R94] presented a technique to extract LTL requirements from natural language using a combination of machine learning and pattern-based techniques. For the DIY assistant we describe in Section 5.3, we have used a similar technique tailored to the specifics of the DIY domain. The translation was implemented by a Master's student [R53].

3.5. The Computational Complexity of Changing Plans

In the previous section, we analysed how the requests of a user to change a given plan can be formalised using Linear Temporal Logic. In order to be able to develop planning techniques which are suitable to address them, we first have to understand the theoretical aspects of adhering to requests by the user.

3.5.1. Complexity of Plan Verification

We start our investigation with a request that has – so far – not been discussed explicitly. We however start with it, as the results from its investigation from an important cornerstone of further theoretical as well as practical work in this thesis.

Consider a situation where the user is wholly discontent with the plan that the planner has presented to him. Instead of asking the planner for specific changes, he might simply provide the planner with a sequence of actions that he wants to perform (or to be performed). In this situation, the planner has to verify that the plan the user has provided is actually a solution to the planning problem. For HTN planning this means that the plan has to be executable and that it can be obtained from the initial abstract task via decomposition.

Note that in this scenario, we assume that the user does only provide the primitive actions, i.e. the plan itself. He does not provide a decomposition for this plan. Especially in a mixed-initiative setting this is a reasonable assumption, as the user will only provide the steps he has to undertake. Further, we do not expect that the user is able to name the specific methods applied in the planner's model to obtain these actions.

Formally, the plan verification problem asks the following.

Definition 8 (Plan Verification)
Given a planning problem $\mathcal{P}$ and a sequence of actions π. The plan verification problem is to decide whether $\pi \in \mathfrak{S}(\mathcal{P})$.

The first insight into the complexity of plan verification gave an investigation of the connection between planning problems and formal grammars [F42]. The set of solutions $\mathfrak{S}(\mathcal{P})$ to a planning problem $\mathcal{P}$ can be interpreted as a formal language over the set of actions O. We showed that this language lies strictly between the context free and the context sensitive languages [F42]. Taking the analogy of parsing words for these two languages, the complexity of plan verification should lie between $\mathbb{P}$ (parsing for context free languages) and $\mathbb{PSPACE}$ (parsing for context sensitive languages).

Based upon this initial intuition, we showed that the plan verification problem is $\mathbb{NP}$-complete [C9]. Contrary to most proofs for $\mathbb{NP}$-completeness, the difficulty here lies in proving that the plan verification problem can be solved in non-deterministic polynomial time. One might assume that a simple guess-and-check algorithm would suffice, where we apply decomposition methods until the task network tn to be verified is found. This is however not sufficient, as task networks of size n exist which require an exponential number of decompositions in n to reach them from the initial abstract task. This is caused by the presence of so-called ε-decompositions, methods that allow to decompose an abstract task into an empty task network, i.e. one without tasks in it. If we consider a planning problem with abstract tasks a_i for $i \in \{0, ..., n\}$ such that a_0 can be decomposed into an empty task network and every other a_i is decomposed into a task network containing two copies of a_{i-1}, we can see that every primitive decomposition in this domain requires $2^{n+1} - 1$ applications of decomposition methods. These ε-methods are not just idiosyncrasies of the HTN formalism, but occur naturally in practice. Consider an abstract task `drill_screw(?o,?s)` in the DIY domain that will drill the screw `?s` into the object `?o`. The decomposition method for it will decompose it into a sequence of abstract tasks that first configure the drill appropriately (insert battery and screw bit, put switches into correct position) and second perform the actual drilling. Such a task sequence could, e.g. be `attach_battery`, `attach_bit`, `configure_gear_switch`, `configure_direction_switch`, and `drill`. The tasks setting up the drill must not be executed if the drill is already configured correctly, e.g. from a previous `drill_screw` task for another screw and object. These tasks can be accomplished by simply doing nothing, i.e. by a ε-decomposition. Instead of using ε-decompositions, we could have a separate decomposition method for `drill_screw` for every omitted subset of preparatory steps. This would however lead to both an exponential amount of additional methods and second add significant redundancies into the model. In fact, every planning problem with ε-decomposition methods can be compiled into a model without such methods[6] by added an exponential number of additional new decomposition methods [F42].

As another example, consider a planning problem where packages are delivered via trucks. The abstract task `goTo(?t,?l)` describes moving the truck `?t` to location `?l`. If the truck `?t` is already located at `?l`, this task can obviously be achieved without doing anything. Consequently there should be a decomposition method that decomposes `goTo(?t,?l)` into the empty task network.

To show $\mathbb{NP}$-membership of the plan verification problem, we introduced a mechanism to handle these ε-methods efficiently during the decomposition process. More specifically, we showed that every task network of size n can be obtained by $2 \cdot |A| \cdot n$ decompositions

[6] Except one, if the initial abstract task can be decomposed into an empty plan. This is similar to the Chomsky Normal Form of formal grammars.

plus (an up to exponential amount of) decompositions that only lead to the removal of tasks from the current task network [C9]. We can determine for every abstract task whether it can be decomposed into an empty task network in polynomial time. Further, the size of every intermediate task network after abstract tasks have been removed is at most n. These two results allow us to simulate the necessary exponential number of decompositions in polynomial time, showing $\mathbb{NP}$-membership.

$\mathbb{NP}$-hardness can be proven via a reduction from the vertex cover problem. Roughly speaking, we use an abstract task for every edge to choose a vertex cover and then use the plan to be verified to restrict the number nodes in the cover [C9]. The mechanics of the hardness proof was inspired by the work of Barton [R152] on ID/LP grammars.

Theorem 2 (Complexity of Plan Verification [C9])
Deciding the problem of plan verification is $\mathbb{NP}$*-complete.*

3.5.2. Complexity of Changing Plans

Next, we investigated the four elementary types of change requests we presented in Section 3.4.1. We showed that the computational complexity of the requests does not depend on the type of the request, bust mostly on the amount of additional changes that the planner is allowed to make in order to implement the change [C8].

If the planner is not allowed to perform additional changes, but only to implement the user's request word-by-word, we can use the complexity result for plan verification and show $\mathbb{NP}$-completeness for the four types of requests [C8]. If the planner is allowed an arbitrary amount of changes, we can reduce the HTN plan existence problem to answering the request [C8]. Since HTN plan existence is undecidable [R143], answering the four basic types of change requests with unlimited additional changes is undecidable. Lastly, we showed that answering the four requests with a bound of k additional changes is $\mathbb{NEXPTIME}$-complete, based on a reduction from the plan existence problem for acyclic HTN planning problems.

These complexity results however cover only the four elementary types of requests to change the plan. Since we propose to use LTL as the unified representation of user requests to change the plan, we should also investigate the computational complexity of changing plans with respect to a change request formulated as an LTL formula ϕ. As such, we should define the decision problem(s), depending on the amount of additional changes allowed to be performed. For the problem without allowed changes, it is unclear how to define it in the first place, as an arbitrary LTL formula does not directly imply changes. As such, we will only consider k-limited changes and unrestricted changes.

Definition 9 (k-LTL-Change)
Let $\mathcal{P}$ be an HTN planning problem and ϕ be an LTL formula over $\mathcal{P}$'s proposition symbols and actions. Let $\pi \in \mathfrak{S}(\mathcal{P})$ be a solution to $\mathcal{P}$.

k-LTL-Change is the problem of deciding whether a solution π^* to $\mathcal{P}$ exists for which $[[\phi]](\pi^*) = \top$, such that π^* can be obtained from π by adding and removing at most k actions. t

ANY-LTL-CHANGE *is the problem of deciding whether a solution* π^* *to* $\mathcal{P}$ *exists for which* $[[\phi]](\pi^*) = \top$.

Note that the new solution π^* for ANY-LTL-CHANGE has no connection to the old solution π. This is a necessary consequence of considering any amount of changes to the original plan. The only information contained in the original plan is that the problem $\mathcal{P}$ is in principle solvable.

We now show that the k-LTL-CHANGE problem is $\mathbb{NEXPTIME}$-complete, as were the four basic types of requests.

Theorem 3
k-LTL-CHANGE *is* $\mathbb{NEXPTIME}$*-complete.*

Proof. Membership: Let $\mathcal{P}$ be an HTN planning problem. Since k is encoded logarithmically, we can perform up to k additions and deletions of actions in non-deterministic exponential time. Let π^* be the resulting plan. Afterwards, we can in linear time in $|\pi^*|$, i.e. exponential time in the input, evaluate the truth of ϕ based on the rules presented in Section 2.6. If π^* does not satisfy ϕ we return false. Lastly, we run the $\mathbb{NP}$ algorithm for plan verification to check whether π^* is a solution to $\mathcal{P}$ or not. This remains an $\mathbb{NEXPTIME}$ algorithm, as we simply return the result of the plan verification algorithm.

Hardness: We reduce from the plan existence problem for acyclic HTN planning problem, which is known to be $\mathbb{NEXPTIME}$-complete [R68]. Let $\mathcal{P} = (L, A, O, M, a_I, s_I)$ be an acyclic HTN planning problem. We construct a new planning problem in the following way: create a new primitive action p that is not contained in $\mathcal{P}$. p has neither preconditions nor effects, i.e. is always applicable. Next, we introduce a new abstract task a^* that is not contained in $\mathcal{P}$. We then add two decomposition method for a^*. One method decomposes a^* into a task network containing only p and the other into a task network containing only a_I.

$\pi = (p)$ is clearly a solution to the transformed problem. For a given acyclic HTN planning problem, the maximum length of a plan derivable via decomposition is $\Delta^{|A|}$ where Δ is the maximum number of subtasks in a method. We thus pose the request of changing $\pi = (p)$ with the formula $\phi = G\neg p$ and up to $k = 1 + \Delta^{|A|}$ allowed changes.

If $\mathcal{P}$ has a solution, then it contains at most $\Delta^{|A|}$ actions. To obtain it from π, we remove p and add the solution, amounting to at most $1 + \Delta^{|A|}$ operations. This solution also satisfies ϕ as it cannot contain p.

If there is a changed plan π^* satisfying ϕ, obtainable with at most k changes from π, it is clearly a solution to $\mathcal{P}$. □

Next we show that ANY-LTL-CHANGE is undecidable.

Theorem 4
ANY-LTL-CHANGE *is undecidable.*

Proof. We reduce from the plan existence problem for HTN planning, which is known to be undecidable [R143]. Let $\mathcal{P} = (L, A, O, M, a_I, s_I)$ be an HTN planning problem.

We construct a new planning problem in the following way: create a new primitive action p that is not contained in $\mathcal{P}$. p has neither preconditions nor effects, i.e. is always applicable. Next, we introduce a new abstract task a^* that is not contained in $\mathcal{P}$. We then add two decomposition method for a^*. One method decomposes a^* into a task network containing only p and the other into a task network containing only a_I.

$\pi = (p)$ is clearly a solution to the transformed problem. We pose the request of changing $\pi = (p)$ with the formula $\phi = G\neg p$.

If $\mathcal{P}$ has a solution, it will also be a solution to the transformed problem. This solution also satisfies ϕ as it cannot contain p.

If the call to Any-LTL-Change returns true, there is a plan not containing p. This plan is a solution to the original planning problem $\mathcal{P}$. □

These two results show that answering change requests formulated in LTL are as hard as answering the four types elementary change requests. Further, we can deduce that optimising the number of changes is undecidable in the sense that there is no terminating algorithm that can output the minimum number of required changes. If there would be such an algorithm, it could be used to decide the undecidable Any-LTL-Change problem.

Lastly, we can draw conclusions from these results for the design of algorithms for handling a user's request to change the plan. The question arises whether it is sensible to develop specialised algorithms for handling change requests, or whether is more sensible to integrate handling them into a general planning algorithm. A specialised algorithm is (in the first instance) only reasonable if changing the plan is easier than planning itself, else e.g. a compilation-based approach is more sensible as a first option. Since adhering to change requests is undecidable, this is not the case. I.e. we should, at first, integrate answering change requests into the planning process itself. We will describe in Section 5.2 how we can integrate requests formulated in LTL into the planning techniques we present in the next section. There we will further describe improvements to the state of the art in SAT-based planning with constraints in LTL.

4. From Verification to Planning

4.1. Summary

In this chapter, we describe how the theoretical results of the previous chapter have led to the development of the first HTN plan verifier, and further to a highly efficient HTN planning technique. Both the verifier and the planning technique are based on a translation into propositional logic. The resulting formulae are solved by off-the-shelf SAT solvers. Thus, both the verifier and the planner will automatically profit from any improvements made to SAT solvers in the future.

As we showed in the previous chapter, determining whether a given plan is a solution to an HTN planning problem is $\mathbb{NP}$-complete. Thus a translation of the problem into the equally $\mathbb{NP}$-complete boolean satisfiability problem is a suitable means to solve the problem. We start by presenting such an encoding and show that it is able to verify plans for common benchmark instances [C7].

One could easily extend this encoding for plan verification to not only be able to verify plans, but also to find them. There is however one theoretical issue with this extension: There cannot be a single propositional formula that is satisfiable if and only if a given HTN planning problem has a solution. This is due to the fact that plan existence for HTN planning is undecidable [R88, R143] – even for variants [F35]. To circumvent this issue, we consider plan existence only for plans that have a limited decomposition depth K. To nevertheless achieve completeness, we iterate over the bound until a plan has been found.

A naive adaptation of the verification encoding for planning proved to be too inefficient in practice. Thus we developed a specialised encoding for the class of totally-ordered planning problems [C5]. Due to their specific structure, we were able to significantly reduce the size of the encoding. This in turn enabled SAT solvers to solve the translated problems efficiently. The reduction in size is achieved by representing all (depth-limited) decomposition trees of the planning problem in a single common super-tree, the *Path Decomposition Tree* (PDT). A PDT contains all possible, K-depth-limited decomposition trees as its rooted subtrees. The propositional encoding then represents a decomposition tree as a subtree of the PDT.

Next, we show that this compact encoding for totally-ordered HTN planning problems can form the basis for an encoding for general, i.e. partially-ordered, HTN planning which is similarly compact. We introduce an encoding that explicitly tracks the partial

order imposed by methods within the propositional formula [C6]. This encoding requires the SAT solver to explicitly reason about the order contained in the planning domain's methods, which takes considerable effort. To alleviate the SAT solver from this reasoning, we introduce a method to handle these ordering constraints externally, i.e. before the propositional formula is constructed. For this, we modify the construction of the PDT so that a compact representation of the order in all primitive action networks that can be derived via K-depth-limited decomposition can be extracted [C2]. Technically, every derivable primitive action network will be an induced subgraph of the extracted ordering – called the *Solution Order Graph* (SOG). Exploiting the information in the SOG, the ordering constraints imposed on the solution do not depend on the applied methods anymore. This allows for an asymptotically more compact representation of order, which greatly improves the efficiency of the planner based on it.

The three encodings presented so far have only been developed for satisficing planning, i.e. for the task of finding *any* plan. In practice it is however often necessary to not only find *some* plan, but the shortest or a reasonably short plan. We show that the presented propositional encodings can also be utilised to find such optimal (or near optimal) solutions. We start with finding some satisficing solution, and optimise the plan length with additional calls to the SAT solver thereafter. To be able to perform this optimisation, we have to determine a maximum decomposition depth $K(\ell)$ such that all plans of length ℓ have a decomposition depth of less than $K(\ell)$. During the development of the plan verifier, we have already proposed three algorithms to compute an upper bound on $K(\ell)$ [C7]. These bounds were however unnecessarily high. We present a new algorithm to compute a succinct upper bound, which is used in our optimal SAT-based HTN planner [C1].

Lastly, we show that a planner based on these new encodings outperforms the current state-of-the-art in HTN planning. We compare the new planner against a wide range of competitors: PANDApro [F20], HTN2STRIPS [F34], PANDA [F29, R75], FAPE [R61, R76], SHOP2 [R118], and HTN2ASP [R114]. The evaluation is conducted on a set of benchmark domains which includes domains that were used in the evaluations of other planners, but also new domains. Based on the evaluation results, we can claim that the new planner is an important step towards fulfilling the first requirement we set out in Section 3.3: having a fast planner.

Core publications described in this chapter

[C1] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "Finding Optimal Solutions in HTN Planning – A SAT-based Approach". *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019).* IJCAI, 2019, pp. 5500–5508. DOI: 10.24963/ijcai.2019/764

[C2] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "Bringing Order to Chaos – A Compact Representation of Partial Order in SAT-based HTN Planning". *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019).* AAAI Press, 2019, pp. 7520–7529. DOI: 10.1609/aaai.v33i01.33017520

[C5] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "totSAT – Totally-Ordered Hierarchical Planning through SAT". *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*. AAAI Press, 2018, pp. 6110–6118

[C6] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "Tracking Branches in Trees – A Propositional Encoding for solving Partially-Ordered HTN Planning Problems". *Proceedings of the 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*. IEEE Computer Society, 2018, pp. 73–80. DOI: 10.1109/ICTAI.2018.00022

[C7] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "This is a solution! (... but is it though?) – Verifying solutions of hierarchical planning problems". *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press, 2017, pp. 20–28

4.2. Translating Plan Verification into SAT Formulae

Plan verification is the problem of deciding for a given plan whether it is a solution to a given HTN planning problem. This problem plays a central role, not only in mixed-initiative planning, but in planning in general. For example, local search techniques [R121] alter a current plan until it is a solution to the planning problem. If we were to apply this technique to HTN planning, we would have had to check, for every altered plan, whether it is reachable via decomposition from the initial abstract task. An independent plan verifier is also a useful tool for planning researchers in order to check that their planners work correctly. Plan verification may play a role in forthcoming HTN planning competitions to check whether the plans output by the planners are actually valid [F13]. The general interest in HTN plan verification is also emphasised by the fact that after the publication of the first HTN verifier by us, Barták et al. [R48] have recently published a second independent HTN plan verifier.

We showed in Section 3.5.1 that the HTN plan verification problem is $\mathbb{NP}$-complete. Due to the problem's complexity, a translation into another $\mathbb{NP}$-complete problem for which efficient solvers already exist seems a reasonable choice. Among the $\mathbb{NP}$-complete problems, the satisfiability problem for propositional logic (SAT, see Definition 7) has both ready-to-use and highly efficient solvers.

When using SAT to solve the plan verification problem, we have to construct a propositional formula $\mathcal{F}$ that is satisfiable if and only if a given plan π is a solution to a planning problem $\mathcal{P}$. In HTN planning a solution has to fulfill two requirements (see Definition 5): It has to be executable and it has to be obtainable via decomposition from the initial abstract task. The former can be verified easily with existing tools, e.g. VAL [R108] or the formally verified validator for PDDL [R47]. Thus what remains for a translation into propositional logic is to determine whether it is possible to obtain a plan π via decomposition from the initial abstract task.

The created propositional formula $\mathcal{F}$ must – at least in some way – encode the decompositions that were used to obtain tn. As we argued in Section 3.5.1, there are cases

where it is necessary to apply an exponential number of decomposition methods in order to obtain a primitive decomposition of an abstract task. Most of these decompositions however will not add new primitive actions to the task network. We showed that at most $2|A||\pi|$ "productive" decomposition methods are necessary to obtain any solution π – provided some decomposition exists at all [C9]. Determining for an abstract task whether it can be decomposed into the empty task network can be done in polynomial time (see Section 3.5.1). If our encoding allows for the removal of these tasks from a current task network, $2|A||\pi|$ is a theoretical upper bound for the number of decompositions the encoding needs to consider when verifying π.

Next, we describe our encoding for plan verification. Using this encoding together with the theoretical upper limit on the number of decompositions $2|A||\pi|$ is practically impossible, as it will usually be far too high for the formula to be constructable. Thus we propose two methods to compute better bounds. Note that in Section 4.5 we will present another method to compute these bounds, which is empirically significantly better than the two methods presented in this section. The newer method to compute the bound was not included in the evaluation, however.

4.2.1. Encoding Decomposition

When encoding the process of decomposition into propositional logic, restricting the number of allowed decompositions directly is quite cumbersome. Instead, we only restrict the depth of the Decomposition Tree corresponding to the applied decompositions. We denote this depth bound with K. If we are given a bound B on the number of decompositions to apply, we can set $K = B$. This might allow the encoding to perform more decompositions than B, but ensures that any combination of $B = K$ decomposition methods can be applied. Note that this is sufficient for correctness. As our encoding uses the decomposition depth K as a bound, we will aim to compute succinct values for K instead of B in the next section.

A satisfying valuation of the formula will represent a Decomposition Tree T via its decision variables, whose yield is the plan π. Such a tree T exists if and only if π can be obtained via decomposition from the initial abstract task [R88]. We can arrange the vertices of each Decomposition Tree T into layers where edges of the tree connect nodes in adjacent layers. Then the number of necessary layers is equal to the maximum depth K of any Decomposition Tree we want to consider. We can further view every layer of the Decomposition Tree as a sequence of tasks and limit its length – using the results of Section 3.5.1 – to $|\pi|$. In total we can assign each node of any Decomposition Tree T to a point in a matrix of size $K \times |\pi|$. Figure 4.1 contains a visual depiction of such an assignment. To further ease the construction of the formula, we repeat leafs of a Decomposition Tree, i.e. primitive actions, in the layer below them. This ensures that the yield of the Decomposition Tree will be completely present in the last layer.

Each point in the matrix is represented by its coordinates (l, p) where l identifies the layer and p the position within the sequence of possible tasks in the layer. To encode the Decomposition tree, we introduce for every point (l, p) and for every task $a \in A \cup O$ a decision variable $a^{(l,p)}$. If $a^{(l,p)}$ is true, there will be a corresponding node v in the

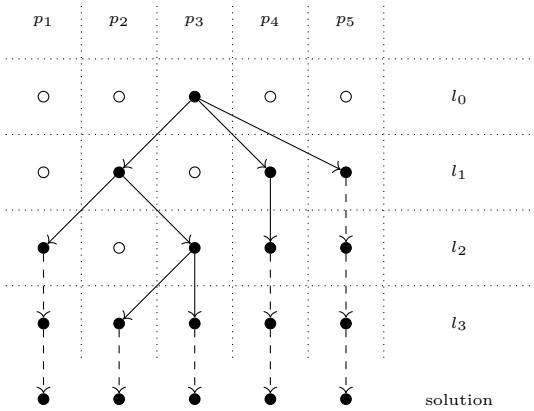


Figure 4.1.: A structural depiction of the plan verification formula, where tasks are arranged in a sequence of layers and are assigned to a position in each layer. A selected Decomposition Tree is shown in black. Decomposition is shown with straight arrows, while inheritance of primitive actions is shown with dashed arrows. Constructed after a Figure in [C7]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

Decomposition tree whose label, i.e. $\alpha(v)$, will be a. Next, we introduce for every two positions p and p' in every layer l (except for the last) the decision variable $\mathbf{e}^{(l,p)}_{(l+1,p')}$, which if true represents an edge in the Decomposition Tree between (l,p) and $(l+1,p')$. To fully represent Decomposition Trees, we further introduce for every point in the matrix and method m the decision variable $m^{(l,p)}$, indicating that $\beta(v) = m$. Lastly, we add decision variables $\mathbf{b}^{(l,p)}_{(l,p')}$ for every pair of positions p, p' in each layer l encoding the relative order of the nodes in each layer. The formula asserts the properties required in the definition of a Decomposition Tree (Definition 6). The formula asserts that

1. every point in the matrix is labelled with at most one $a^{(l,p)}$ and if and only if it is labelled with an abstract task, then at most one of the $m^{(l,p)}$ is true.
2. for every $m^{(l,p)}$ that is true, each subtask t of m will occur in the layer $l+1$ or t can be decomposed into the empty task network.
3. every primitive action is repeated at the same position in the next layer.
4. the order of tasks in layer l is inherited to its subtasks in layer $l+1$.
5. the order in each layer is valid, i.e. transitively closed and acyclic.
6. every point in the matrix has at most one incoming edge.
7. if a point in the matrix is labelled with a task, it has either an incoming edge or is at layer 0. In layer 0 only one position can be labelled with a task.
8. for every edge $\mathbf{e}^{(l,p)}_{(l+1,p')}$ both adjacent positions (l,p) and $(l+1,p')$ are labelled with a task.

We omit the details of the encoding at this point for the sake of brevity. Details can be found in the corresponding paper [C7]. We shown that our encoding is correct and complete, i.e. that every satisfying valuation represents a Decomposition Tree of height $\leq K$ and that for every such Decomposition Tree a satisfying valuation exists [C 7, Theorem 1].

As the last step in the encoding, we add clauses to the last layer ensuring that the order represented by the $\mathbf{b}^{(l,p)}_{(l,p')}$ atoms is in line with the natural ordering of the positions, i.e. $\neg\mathbf{b}^{(l,p)}_{(l,p')}$ for all $p > p'$, and clauses asserting that the positions in the last layer are labelled with the solution to be verified.

We want to note that the construction of the formula does not require π to contain only primitive actions. This formula could thus also be applied to (partial-)plans still containing abstract tasks and verify whether they can be derived via decomposition from the initial abstract task. This ability is e.g. useful for search-based integrations of the user, as they will deal with plans containing abstract tasks frequently.

4.2.2. Computing Bounds for the Tree Height

The presented translation is based on the fact that there is an upper bound K to the height of Decomposition Trees leading to a given plan π.[1] Based on our theoretical results, K can be set to $2|A||\pi|$ [C9]. We show that this bound is far too high in practice and can be lowered significantly, thereby improving the efficiency of the verifier.

The theoretical bound considers solely the worst possible case for decompositions and does this without taking the specific structure of the planning problem into account. More specifically, all abstract tasks are viewed as interchangeable and it is assumed that every abstract task has only methods leading to the maximum bound, i.e. methods leading to tasks with high minimum decomposition depth. This is often not the case in typical planning problems. We developed two methods to extract a more succinct bound of which the third dominates the two others, i.e. it is always smaller or equal to them, and should thus be preferred. These bounding techniques exploit specific structures present in the planning problem.

As a first observation, there is a significant proportion of planning problems that – after grounding – are acyclic (see Section 2.4.2). For them, we construct the Task Transition Graph (TTG) $\mathcal{T}$ whose nodes are the problem's tasks (primitive and abstract). $\mathcal{T}$ contains the directed edge (a, b) whenever there is a decomposition method for a whose task network contains b. The planning problem is acyclic if and only if $\mathcal{T}$ is acyclic. Obviously, we can use the length of the longest path in $\mathcal{T}$ as an upper bound for height of any decomposition tree in this domain.

For the second method, we consider planning problems in which all decomposition methods contain at least two subtasks. In such problems, we can apply at most $|\pi| - 1$ decompositions until we have reached a plan with the same size as π. Any further applied method will ensure that any resulting task network cannot be equal to π as it contains more tasks. Any HTN planning problem can be compiled into a solution-

[1] Or alternatively an upper bound B on the number of applied decompositions.

domain	number of instances	K_{theo}		K_{unit}		K_{TTG}	
		min	max	min	max	min	max
UM-Translog ●	21	70	1258	5	37	3	6
Satellite ●	22	20	510	5	17	1	4
SmartPhone ●	3	132	324	11	15	3	∞
Woodworking ●	5	12	48	2	4	1	2
Monroe ●	50	198	11032	∞	∞	4	8

Table 4.1.: Number of instances per test domain. Minima and maxima for each method to compute a hight bound. Colours refer to the runtimes in Figure 4.2. Infinity represents instances in which either the computation is not applicable (SmartPhone) or timed out (Monroe). K_{theo} denotes the theoretical upper bound $2|A||\pi|$, K_{unit} the bound computed for the compiled domain where all methods have at least two subtasks, and K_{TTG} is the bound applicable only to problems with acyclic grounding. Abridged version from [C7]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

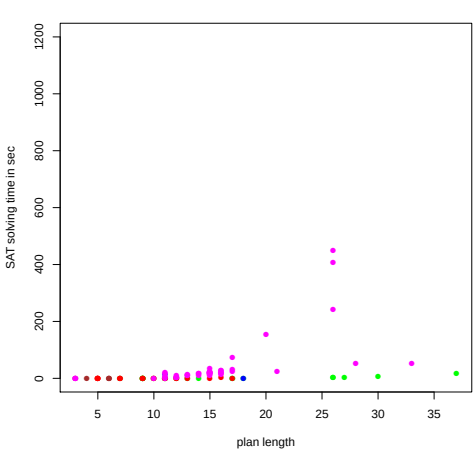
equivalent planning problem where all methods at least two subtasks [F42].[2] However an exponential increase in size might be necessary. If possible within the given time limit, we compile the given HTN planning problem into one where all methods have at least two subtasks. If not, we do not use this second method. Since we alter the domain structure by this compilation, we used the altered domain for verification if the computed bound was the smallest and was thus used for verification.

4.2.3. Empirical Evaluation

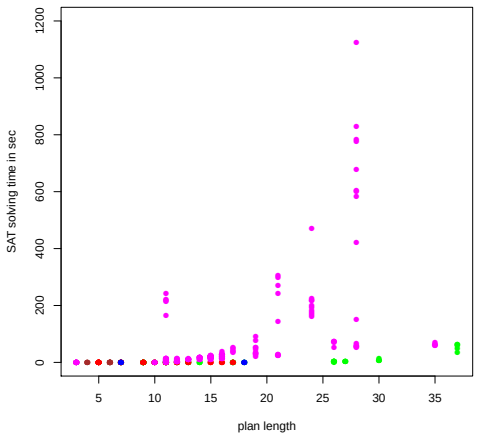
To ascertain the empirical performance of our plan verifier, we tested it on several planning problems with (at the time of publication) known solutions. The experiments were conducted on an Intel Xeon E5-2660 with 503 GB available RAM. We used instances from the domains UM-Translog, Satellite, SmartPhone, and Woodworking. These domains were e.g. used in the evaluation of the HTN planner by Bercher et al. [R75]. We added 50 randomly selected plans form the Monroe corpus [R106]. This corpus was developed to test plan and goal recognition algorithms and contains in addition to the plan to be recognised, the HTN planning domain with which the plans were generated. Planning in the Monroe domain is rather trivial, while determining for any given sequence of actions whether it is a solution is difficult. Note especially that the plan verification problem is equivalent to the plan goal recognition problem for HTN planning if we know that the whole plan was observed [C9]. In total we considered 101 planning problems with their known solutions.

As a first step, we computed the height bound K with all three methods (theoretical

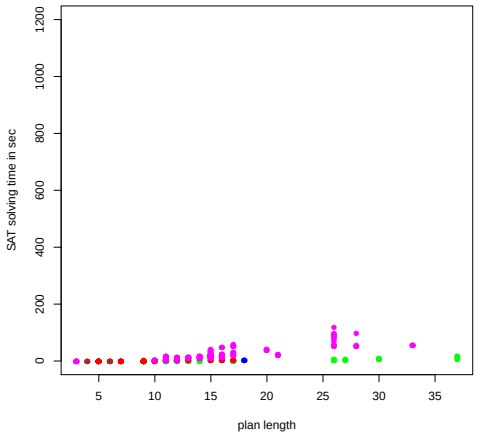
[2]Note that this paper describes only a method to ensure that all methods contain at least one sub task. This method can however be easily expanded to achieve the required property. Further, technically there can be methods with fewer than two subtasks, as long as they decompose the initial abstract task and no method contains the initial abstract task as a subtask.



(a) Runtime on actual solutions.



(b) Runtime on non-solutions, generated by replacing a single action in a solution.



(c) Runtime on non-solutions, generated by random-walking.

Figure 4.2.: Runtimes of plan verifier on test instances. All Figures are copied from [C7]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

limit, TTG, and at-least-2-subtasks). The results are shown in Table 4.1. Note that the bound based on our theoretical results is always significantly higher than those computed by the other two methods. Especially the minima are higher than the maxima for all other methods, if they exist. Next, the unit-decomposition expansion timed out[3] on all instances of the Monroe domain, but led to comparably small bounds for all other instances. After grounding, only some instances of the SmartPhone domain contained cycles, i.e. the TTG-based method was usable in all other instances. In all other instances, the TTG method was an improvement over the unit-method or the theoretical upper bound (for Monroe).

To test the efficiency of the presented encoding, we have not only considered plans to be verified, but also non-plans which should be refuted. We added two types of non-solutions to the test set. One was generated from solutions by randomly replacing an action in the plan with another action from the domain. The second was generated by randomly applying applicable actions until a plan of the same length as the known solution was found. For both types, we generated five non-solutions for each instance, i.e. 505 instances for each type.

For all solutions our encoding was able to verify them. The maximum runtime for verifying a solution was 451 seconds. However, the runtime exceeded 100 seconds in only four instances. The runtimes per instance are shown in Figure 4.2.

The runtime behaviour of our verifier on the almost-solutions – where we randomly replaced an action in the solution with another action – is shown in Figure 4.2b. For some instances runtime increased significantly (the maximum runtime was 1123 seconds), while all were still refuted correctly. We suppose that this is due to the fact that the resulting formula is "almost satisfiable" as the plan to verify differs only in one action from a solution. In general, propositional formulae that are almost satisfiable (i.e. satisfiable if only a few or just a single clause were to be dropped) are considered computationally hard. Of the 505 plans generated by random walking, 501 were classified as non-solutions, with the highest runtime being less than 120 seconds. The four remaining plans were classified as solutions. We manually checked that this classification was indeed correct. The runtime needed to refute each of the 501 non-solutions is shown in Figure 4.2c.

As such, our evaluation has validated our plan verifier and has shown that it can handle verification instances it was tasked within a reasonable time-frame. At this point, we want to note that since the publication and development of this plan verifier a second HTN plan verifier has been published by Barták et al. [R48]. It views the HTN planning problem as a grammar and if parsing of the plan is possible with that grammar, it is a solution to the planning problem. This approach showed – compared to the technique presented here – a significant improvement in performance.

[3] We considered a timeout of one hour. All other computations finished within a few seconds.

4.3. A Tree-style SAT Formula for Totally Ordered Planning

The encoding for plan verification describes the process of decomposing the initial abstract task into a given task network. If we were to remove the clauses asserting that the result of that decomposition is the given plan π and instead check that the obtained plan is executable, we would – at first glance – have constructed a formula that is satisfiable if a solution to the planning problem exists. Unfortunately, this approach has two significant drawbacks. First, HTN planning is undecidable [R143], i.e., it cannot be possible to create a single propositional formula that is satisfiable if and only if a given planning problem has a solution. The formula we described in the previous section cannot capture any decomposition leading to a solution, but only those that have a decomposition of height less than K. The parameter K, in turn, depends on the (maximum) length n of plans considered, as we compute K from it. Since HTN planning is undecidable, no upper bound to n or K can exist. To obtain a complete planner, we would have to increase n until a solution has been found.

Second, our initial experiments showed that the plan verification formula is unusable for planning due to its size. It will contain $\Theta(n^4)$ clauses, if we consider plans of length n. For a plan of 100 actions, this means 10^8 clauses, which is at the upper limit of solvability for current SAT solvers. For even longer plans, we cannot expect a SAT solver to find a satisfying valuation in any reasonable time frame. Further, the formula contains a high degree of non-determinism, making it hard for SAT solvers.

Thus a new encoding was developed, specifically designed for the purpose of planning. A significant number of clauses in the plan verification formula, in fact those responsible for $\Theta(n^4)$ clauses, encodes the partial order of tasks in task networks. Since handling partial order is (seemingly) the most difficult element of an HTN planning problem, we first developed a specialised encoding for totally-ordered HTN planning problems. Based on this encoding we were able to add support for partial order (see Sec. 4.4).

4.3.1. Path Decomposition Trees

The central deficiency of the encoding for plan verification is that it does not exploit the specific decompositional structure of the planning problem at all – apart from using it to compute more succinct height bounds. An encoding exploiting the specific structure of decompositions contained in the problem should expose more of the problem's structure in the propositional formula and thus should make the formula easier to solve. As the first step in exploiting the problem-specific structure of decompositions, we need a representation of that structure. Unfortunately, considering all individual decompositions of the initial abstract task into plans is impractical or impossible (depending on the problem), because there are either too many or even infinitely many such decompositions.

Thus, we need a compact representation of decompositions of the initial abstract task a_I. Assume that this representation would encode *all* such decompositions. Any propositional encoding based on this representation could only encode a subset of these decompositions, as we would else created a terminating decision procedure for an undecidable problem. Since representing more information than can be utilised by any

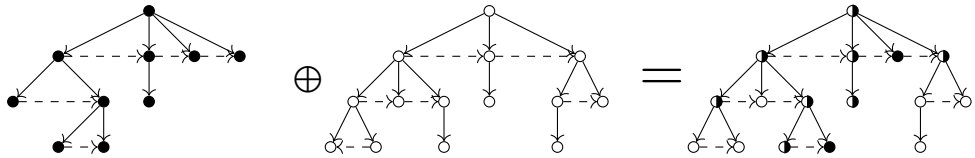


Figure 4.3.: Two Decomposition Trees, one marked with black nodes and one with white nodes. On the right, an ordered Path Decomposition Tree is shown that has both Decomposition Trees as sub-trees. The morphism is shown via the colour of the nodes.

encoding is futile, we will also restrict this compact representation to a subset of all possible decompositions of the initial abstract task. The design choice at this point is to determine which subset to choose. Under the assumption that the specialised encoding will – as the encoding for plan verification – bound the maximum depth of decomposition in some way, our compact representation will consider all decompositions up to a given depth limit K. Note that via iterating over the depth bound K, we still retain completeness.

As a compact representation of decompositions, we introduced the concept of Path Decomposition Trees (PDTs). Intuitively speaking, a PDT is a tree P such that all possible Decomposition Trees T (Def. 6) up to a given depth bound K are rooted subtrees of P. This allows us to use the same nodes of the PDT to represent multiple Decomposition Trees, which essentially "fuses" multiple Decomposition Trees and significantly compacts their representation. To distinguish the Path Decomposition Trees for totally-ordered HTN planning problems from those for general, partially-ordered problems, we introduce them formally as Ordered Path Decomposition Trees, but use the term Path Decomposition Tree throughout this thesis.

Definition 10 (Ordered Path Decomposition Tree [C5])
Let $\mathcal{P}$ be a planning problem and K a depth bound.
Then $P_K = (V, E, \alpha)$ is an Ordered Path Decomposition Tree iff

- *(V, E) is a directed and ordered tree rooted at r_I,*
- *$\alpha : V \to 2^{A \cup O}$ labels each node with a set of (primitive and abstract) tasks, and*
- *for every Decomposition Tree $T = (V^T, E^T, \prec^T, \alpha^T, \beta^T)$, (V^T, E^T) is a subtree of (V, E) under the morphism $\phi : V^T \to V$ such that for the root r of (V^T, E^T) $\phi(r) = r_I$ holds , for every node $v \in V^T$ it holds that $\alpha^T(v) \in \alpha(\phi(v))$, and the order $\prec^T$ is the same as the order of nodes in (V, E) under ϕ.*

We show in Figure 4.3 two Decomposition Trees and a possible Path Decomposition Tree that contains both of them as sub-trees.

To see that a PDT can be a *compact* representation of decompositions, consider all Decomposition Trees up to depth K for a planning problem where the maximum number of subtasks in a method is Δ. The last depth layer of a Decomposition Tree can contain up to Δ^K nodes. The number of all inner nodes in the tree is less or equal to $\Delta^K - 1$ and equal only if $\Delta = 2$. In the edge case $\Delta = 1$ the number of inner nodes is $K - 1$.

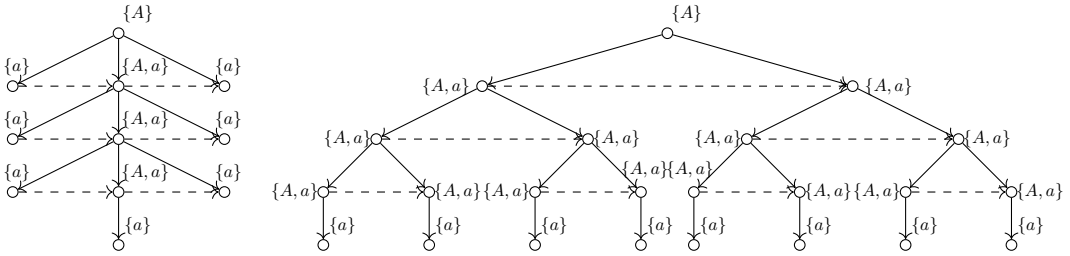


Figure 4.4.: Two PDTs for a domain with the three decomposition methods $A \to aA$, $A \to Aa$, and $A \to a$.

Any of these nodes can be labelled with any (primitive) task or not be present in the Decomposition Tree at all. Thus there are $\mathcal{O}((|O|+1)^{\Delta^K})$ different Decomposition Trees of depth K. However, there is always a path Decomposition Tree of size $\mathcal{O}(\Delta^K)$ that has all Decomposition Trees as sub-trees – namely one whose inner nodes have always Δ children and that labels all nodes v with $\alpha(v) = A \cup O$. This shows that the PDT representation can be exponentially more compact than enumerating all Decomposition Trees.

Naturally, we are interested in a "minimal" PDT. Minimal is put into quotation marks, as it is *a priori* not clear what minimality means here, as there are several potentially plausible metrics we might want to minimise. At first glance, it seems useful to minimise the number of nodes in the tree or the size of the label sets. Both optimisations might lead to smaller and more informative encodings. There is, however, no obvious means to weigh these two criteria against each other, i.e. it is not clear whether it is better to have fewer nodes in P_K at the cost of larger label sets $\alpha(\cdot)$.

Even if we want to optimise the number of vertices of the PDT, the problem might be computationally difficult. Even if computing the optimal size of the PDT is still polynomial,[4] the minimal PDT can have exponentially many nodes. As we have to compute the PDT explicitly for encoding, a complexity analysis at this point is futile. Note that locally optimising the number of children for every subtask of the tree does not lead to a globally optimal number of nodes for the PDT. To see this, consider a planning problem with three decomposition methods $A \to aA$, $A \to Aa$, and $A \to a$, where A is an abstract task, a a primitive action, and A the initial abstract task. The PDT with the minimal number of nodes for $K = 4$ is depicted in Figure 4.4. We can see that only a single task in every layer is labelled with the abstract task A. This ensures that only one node per layer has children in the next layer. If a Decomposition Tree contains the first decomposition method, we can use the first two children to represent its subtasks. If it contains the second method, we can use the latter two. This construction results in a size of the PDT that is linear in the depth K. Note that there are inner nodes with three children, but the maximum size of each method is 2. If we optimise the number of children for each node locally, as done in the second PDT depicted in Figure 4.4, each inner node has only two successors at the cost of the overall size of the tree now growing exponentially in K. This is due to the fact that now each inner node has two children

[4] Which is as of yet unknown.

labelled with the abstract task A, both of which must be expanded further.

Due to these difficulties in finding an "optimal" PDT, we opted to construct the PDT in a greedy fashion. For a given planning problem $\mathcal{P}$, we start with a PDT containing a single node r labelled with $\alpha(r) = \{a_I\}$. We repeatedly expand leafs of the current tree until all leafs are either at depth K or their label sets contain only primitive actions. Consider a leaf l with the label set $\alpha(l)$. We gather all methods applicable to the abstract tasks in $\alpha(l)$. Since the planning problem is totally-ordered, these methods' tasks networks will be sequences of tasks. We combine these sequences into a sequence of sets of tasks in a greedy fashion while trying to minimise the size of these sets. E.g. we reduce the sequences aAa, Aa, and aB to $\{a\}, \{A, B\}$, and $\{a\}$. For each set s in this sequence, we generate a new child for l and label it with s. The constructed tree is a valid PDT [C5].

4.3.2. Tree-style SAT Formulae

In the previous section, we introduced PDTs which are compact representations of all possible Decomposition Trees of an HTN planning problem up to a given depth bound K. For every solution π to an HTN planning problem, there exists a Decomposition Tree that has π as its executable yield (see Theorem 1). Conversely, a solution can be extracted from every Decomposition Tree with an executable yield. Thus instead of modelling solutions in a propositional formula directly (which is difficult), the formula can encode a Decomposition Tree via its decision variables. A valuation of the formula will represent such a Decomposition Tree and the formula will assert that the tree is consistent and that its yield is executable.

Let $P_K = (V, E, \alpha)$ be a totally-ordered Path Decomposition Tree. When constructing the formula, we introduce two types of decision variables.

- a^v – for every $v \in V$ and $a \in A \cup O$
- m^v – for every $v \in V$ and $m \in M$

Using these decision variables, we will select a Decomposition Tree $T = (V^T, E^T, \prec^T, \alpha^T, \beta^T)$ that is a sub-tree of P_K. Setting a^v to true means that v is part of V^T and that $\alpha^t(v) = a$. Similarly setting m^v to true represents $\beta^T(v) = m$.

The propositional formula itself ensures that the represented Decomposition Tree is valid. Following the definition of Decomposition Trees (Definition 6), we have to ensure that

1. for every node $v \in V$ at most one a^v and m^v are true, i.e. v is labelled with at most one task and at most one method is applied.
2. if a^v is true for an abstract task a, an appropriate method m^v is chosen, else no method is chosen.
3. if m^v is true, the appropriate tasks will be assigned to the children of v and all other children have no tasks assigned to them.
4. if a^v is true, its parent also has a task assigned to it (except if a is the root).

These constraints can be transformed into a propositional formula $\mathcal{F}_{decomp}$ [C5]. To ease the further construction of the formula, we also added clauses that ensure that if a primitive action is assigned to an inner node v, it is also assigned to exactly one of the children of v. As a result, every valuation assigns the action at the leafs of the represented Decomposition Tree to the leafs of P_K.

In addition to being a decomposition of the initial abstract task a_I, the yield of the represented Decomposition Tree T must also be executable. Executability means (see Theorem 5) that there is a linearisation of the yield which is executable in the initial state. Using the above-mentioned encoding trick, the yield of T is assigned to the leafs of P_K. Since the HTN planning problem is assumed to be totally-ordered and the PDT P_K complies with that ordering, the natural order of the leafs of P_K will be the order of the tasks in the represented yield of T. As such, the tasks assigned to the leafs of P_K from a sequence of actions for which any propositional encoding for classical planning can be used to ensure executability. We chose the original encoding by Kautz and Selman [R145] since subsequent encodings mostly allow for actions parallelism, which does not occur in totally-ordered planning problems. We call this formula $\mathcal{F}_{exe}$. The connection between the two formulae is made via re-using the a^v atoms for leafs v as the action atoms in $\mathcal{F}_{exe}$.

We showed that $\mathcal{F}_{decomp} \wedge \mathcal{F}_{exe}$ is satisfiable if and only if the HTN planning problem $\mathcal{P}$ for which the formula was created has a solution of depth at most K [C5]. Thus a planner based on it using an appropriate iteration over K will be sound and complete.

To solve any totally-ordered planning problem $\mathcal{P}$, we iterate over the depth bound K starting with the minimally necessary value to obtain a primitive decomposition of a_I. For every depth bound, we construct the PDT P_K and construct the formulae $\mathcal{F}_{decomp}$ and $\mathcal{F}_{exe}$ for it. We then pass $\mathcal{F}_{decomp} \wedge \mathcal{F}_{exe}$ on to a SAT solver. If it returns a satisfying valuation, we can extract the plan and return. If not we increase K. Note that we can technically end this iteration after $K = |A|(2^{|L|})^2$, as if no plan exists at this point for a totally-ordered HTN planning problem, none can exist for higher K [C5].

Note that this (original) procedure has still room for further technical improvements. For example, we could re-use the PDT constructed for $K-1$ to compute the one for K as they are sub-trees of each other. Similarly, the propositional formula for depth K will contain (a large part of) the formula for depth $K-1$, which can be exploited by using an incremental SAT solver [R46]. Further, one can imagine interleaving the individual calls to the SAT solver for different depth bounds, as is done for length bounds in classical planning [R78].

We implemented the described encoding technique within the hierarchical planning framework PANDA.[5] In addition to the code for computing the PDT and transforming it into a propositional formula, the author of this thesis has also implemented significant parts of the preprocessor infrastructure of the planner.

[5] PANDA – currently in its third re-implementation – encompasses several software components written by Daniel Höller, Gregor Behnke, and various students. It contains parsers, a "grounder" [F12], several further preprocessing components, a plan-space-search-based planner [F29, R75], a progression-based planner [F20], and the SAT-based planner described in this thesis.

4.3.3. Empirical Evaluation

To show that the presented planning approach performs well in practice, we conducted an empirical evaluation of its performance.

Planners. We compared our approach against other state-of-the-art HTN planning systems. Each planner was given 10 minutes runtime and 4 GB of RAM per instance on an Intel Xeon E5-2660. Since there are both HTN planners specifically designed to solve totally-ordered HTN planning problems and those that can solve arbitrary problems, we compared our technique against both types of planners. We considered the two following dedicated total-order HTN planners:

- SHOP, a progression-based planner that performs blind depth first search [R135] and
- HTN2ASP, a translation of totally-ordered HTN planning problems into answer set programming [R114]. Since the code of this planner is not available, a Masters student has re-implemented the translation according to the authors' descriptions.

From the group of general HTN planners, we considered a variety of planners employing different solving strategies:

- PANDApro, a progression-based algorithm that uses an encoding of the current state and remaining task network into classical planning for heuristic guidance [F20]. We used it with the greedy A^* search algorithm.
- HTN2STRIPS, a translation of HTN planning problems into a sequence of classical planning problems [F34] which are then solved by a state-of-the-art classical planner, in this case jasper [R79].[6]
- PANDA, a plan-space-search-based planner with heuristics based on the Task Decomposition Graph (TDG). We used both the heuristic that estimates action costs (TDG-c) as well as the one estimating remaining modification effort (TDG-m), both in conjunction with greedy A^* [F29, R75].
- FAPE, a plan-space-search-based planner that is capable of handling domains with temporal constraints [R76]. FAPE uses blind search, but employs an effective pruning technique for search nodes. It is based on a translation of the current search node into a flat temporal model on which feasibility is determined [R61]. The ordering in the domain's methods is translated into temporal constraints. As such, FAPE's pruning can take some, but not all ordering constraints into account in its pruning, which is currently not possible in both PANDApro's and PANDA's heuristics.

  Note that FAPE cannot handle recursion in the planning problem. As such, we ran it only on instances which do not contain recursion.
- UMCP is a plan-space-search-based planning algorithm [R148]. We implemented this algorithm within PANDA. We tested it in its breath-first search, depth-first search, and heuristic configurations.

[6] For a comparison including classical planners form the most recent IPC, see Section 4.4.5.

We tested the presented propositional encoding in conjunction with the best performing SAT solver of the SAT Competition 2016: cryptominisat5 [R65], MapleCOMSPS [R63], and Riss6 [R64]. The resulting planner is named totSATfootnoteThe name totSAT is a joking reference to the emblem of the PANDA planner within which totSAT was implemented. The panda seems (at least to me) to raise his hand to pad his full belly – he is totally full or "total satt" in German.

Benchmark Instances. There is currently no established set of benchmarking instances for HTN planning, let alone for totally-ordered HTN planning. As such, we used instances from domains which have been used in other empirical evaluations of HTN planners [F20, F29, R75, R82]. These are the domains Entertainment, UM-Translog, Satellite, Woodworking, SmartPhone, Rover, and Transport containing in total 127 planning problems. Of those instances only 36 are totally-ordered (20 of UM-Translog, 12 of Entertainment, and 4 of Satellite). We manually added ordering constraints to all other instances, making them totally-ordered. The ordering was added so that at least one solution was preserved in every instance. Further note that only the domains Satellite, Woodworking, and Rover contain no recursion in the lifted model, i.e. those are the only ones on which FAPE was run.

	#instances	totSAT cryptominisat5	totSAT MapleCOMSPS	totSAT Riss6	**PANDApro greedy A^* FF**	HTN2ASP	SHOP	**HTN2STRIPS**	**TDG-c greedy A^***	**TDG-m greedy A^***	**FAPE**	**UMCP-BF**	**UMCP-DF**	**UMCP-H**
Entertainment	12	**12**	**12**	**12**	9	**12**	5	5	9	9	-	5	5	6
UM-Translog	22	**22**	**22**	**22**	**22**	21	**22**	18	**22**	**22**	-	**22**	**22**	**22**
Satellite	25	**25**	**25**	**25**	**25**	**25**	**25**	23	**25**	**25**	**25**	24	18	20
Woodworking	11	**11**	**11**	**11**	**11**	**11**	10	5	9	9	0	6	6	6
SmartPhone	7	**7**	**7**	**7**	**7**	6	4	6	5	6	-	4	4	4
Rover	20	**20**	**20**	**20**	17	8	16	5	2	2	15	0	0	0
Transport	30	**28**	27	26	20	14	0	20	2	1	-	1	0	1
total	127	**125**	124	123	111	97	82	82	74	74	40 / 56	62	55	59

Table 4.2.: Number of solved instances per planner per domain. Maxima are indicated in bold. Planners shown in bold are general HTN planners, i.e. they can also handle partially-ordered planning problems. Copied from [C5]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

Results. Table 4.2 shows the coverage, i.e. the number of instances solved within the time and memory limits, for all planners. Figure 4.5 shows the number of solved instances in relation to the time it took to solve the problems. The new planner totSAT solves between 123 to 125 (depending on the solver) out of the 127 instances. Note that totSAT solves all instances except for the two largest instances in the Transport domain. We tested these two with a higher time limit and totSAT solves both within 30

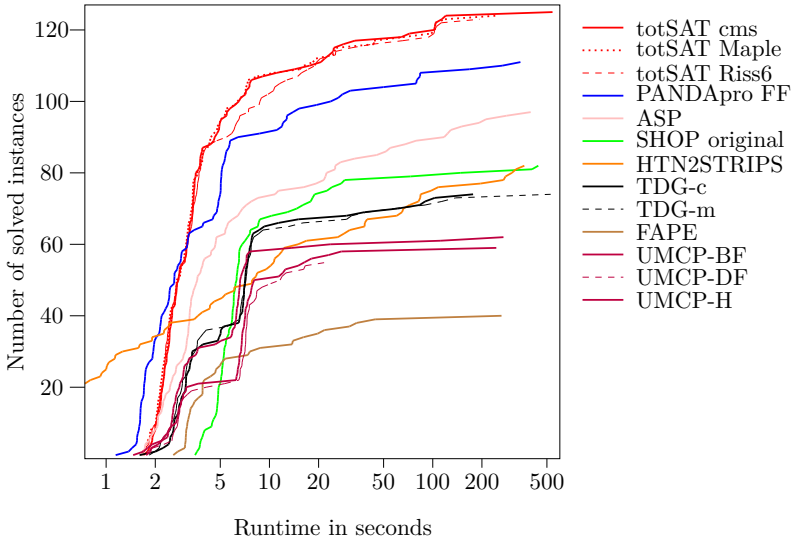


Figure 4.5.: Runtime vs number of solved instances per planner. Copied from [C5]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

minutes.

The two other planners specifically developed for totally-ordered problems (HTN2ASP and SHOP) have a higher coverage than all other general purpose HTN planners, except for PANDApro. Such a behaviour is expected as these two planners can exploit the fact that the domain is totally ordered – as does our encoding. It is remarkable that this advantage even allows the blind-search-based planner SHOP to beat both HTN2STRIPS and the TDG-based planners which use informed search.

4.4. Handling Partially-Ordered Problems

In the previous section, we presented an encoding of totally-ordered planning problems into propositional logic based on a compact representation of all possible decompositions up to a given depth bound. We further showed that the resulting planner outperforms other state-of-the-art HTN planners, some of them significantly. In practical applications, however, planning problems are seldom totally-ordered, as witnessed by the fact that only 36 out of the 127 original benchmark instances were totally-ordered. As an example, consider the DIY domain and its abstract task `setup_drill` setting up a drill for operation. This includes several steps, but at least inserting the battery, inserting the drill bit, setting the gear switch, and setting the direction switch appropriately. Each of these steps is represented by a separate sub-task of the method decomposing `setup_drill`. As these steps can be executed independently from each other there won't be any ordering between them. Further these tasks themselves might contain more than one operation as e.g. inserting the drill bit, which might first insert a bit holder, then fasten it, and then insert the drill bit into the bit holder. Partial order in the method for

`setup_drill` is required to allow the – physically possible and plausible – interleaving of these steps, e.g. with inserting the battery into the drill. Note that – in an assistance scenario – the user might actually request to change the plan to use any of these particular orders (or parts of it). Thus, the planning model has to be able to represent them all and must thus allow for partial order in methods. Further note that there are structures in plans – especially the interleaving of primitive actions stemming from different abstract tasks, which is necessary in the DIY domain – that cannot be represented with totally-ordered domains [F36, F42].

As partial order is an important and, especially in assistance domains, practically occurring feature of HTN planning problems, the planner totSAT presented in the previous section will not suffice as a planner for these settings. In this section we show how the encoding of totSAT can be extended to also cover partially-ordered planning problems. We first present a naive extension of the totSAT encoding that tracks the partial order explicitly in decision variables [C6]. Next, we will show that the ordering constraints contained in a planning problem can be extracted before the problem is encoded and can thus be handled in the formula more efficiently [C2]. We introduce the notion of Solution Order Graphs – a data structure that represents the order of all primitive decompositions of the initial abstract task up to a given depth limit. Lastly, we present an empirical evaluation that compares the performances of the encodings to each other as well as to current state-of-the-art HTN planning systems.

4.4.1. Naive Encoding

The presented encoding of totally-ordered HTN planning problems into propositional logic is based on the concept of Path Decomposition Trees (PDTs). The definition of PDTs in Section 4.3.1 relied on the fact that the subtasks of each decomposition method form a sequence and thus their ordering can be represented in the PDT by an ordered set of children for any inner task. As such the order of the yield of a decomposition tree – the plan it represents – was given by the order induced on the leafs of the PDT. The propositional encoding exploited this fact to encode state transitions directly using the decision variables a^v representing the actions assigned to the leafs of the PDT.

This technique is not directly applicable if the domain contains partially-ordered methods as we cannot choose a sequential order of the children for each inner node. The first solution to this problem we present is to separate the – so far – joint handling of subtasks and the order between them. In an unordered PDT, the order between vertices is not taken into account any more, neither is the order in Decomposition Tree T. It solely represents the tasks contained in any Decomposition Tree T.

Definition 11 (Unorderd Path Decomposition Tree [C6])
Let $\mathcal{P}$ be a planning problem and K a depth bound.
Then $P_K = (V, E, \alpha)$ is a Path Decomposition Tree iff

- *(V, E) is a directed tree rooted at r_I,*
- *$\alpha : V \to 2^{A \cup O}$ labels each node with a set of (primitive and abstract) tasks, and*
- *for every Decomposition Tree $T = (V^T, E^T, \prec^T, \alpha^T, \beta^T)$, (V^T, E^T) is a subtree*

of (V, E) *under the morphism* $\phi : V^T \to V$ *such that for the root* r *of* (V^T, E^T) $\phi(r) = r_I$ *holds and for every node* $v \in V^T$ *it holds that* $\alpha^T(v) \in \alpha(\phi(v))$.

Unordered PDTs can be computed in the same way as ordered PDTs are. We can even use the same procedural mechanics, if we use any topological order[7] of each method instead of the method's actual order [C6].

The propositional encoding of an ordered PDT P_K does not take the ordering of its nodes into account – it comes only into play when executability is encoded. As such we can use the same propositional formula $\mathcal{F}_{decomp}$ for an unordered PDT as for the ordered case. Consequently, its correctness and completeness are retained, i.e. any satisfying valuation of its decision variables a^v and m^v is a valid decomposition tree of depth $\leq K$ and all such decomposition trees correspond to a satisfying valuation [C5, C6].

What remains to encode is executability of the yield of the represented Decomposition Tree T. A witness for executability is a linearisation π of the yield, which is compatible with the ordering constraints imposed by the applied methods. An ordering constraint can only originate in a method imposing such a constraint to its subtasks [C6]. We track these constraints via additional decision variables $\mathtt{b}_w^v$ for each two vertices v and w of the PDT that are direct children of the same node. $\mathtt{b}_w^v$ being true represents that the method applied to their common parent node has introduced the ordering $v \prec w$ on them. The correct truth values of these variables can be maintained using implications from the m^u variables for the parent u of v and w [C6].

Consider any two leafs l_1 and l_2 of a DT T as a sub-tree of the PDT P_K. We can w.l.o.g. assume that they are also leafs of P_K (see Section 4.3.2). The order between l_1 and l_2 in the DT T can only be induced by the method applied to their last common ancestor $a = lca(l_1, l_2)$ in T respectively P_K [C6]. This order will have been imposed on the children c_1 and c_2 of a that are ancestors of l_1 and l_2, respectively. This is the ordering encoded by variable $\mathtt{b}_{c_2}^{c_1}$.

To find an executable linearisation of the yield of T, the encoding proceeds in three steps. First, it encodes a linearisation of the leafs of P_K, which will also include a linearisation of the leafs of T as a subsequence. Second, it ensures that this linearisation does not violate the ordering constraints imposed by the chosen Decomposition Tree via the $\mathtt{b}_w^v$ atoms. Third, it asserts that tasks of the chosen linearisations are executable in the initial state.

To choose a linearisation of the leafs of P_K, we encode a matching of the leafs of P_K to a sequence of time-steps [C6]. We chose the number of time-steps as the number of leafs of P_K. To check the third condition, we can employ any propositional encoding for classical planning. We again use the one by Kautz and Selman [R145]. Lastly, we have to ensure that the chosen linearisation is compatible with the ordering induced by the Decomposition Tree T on the leafs. This is not the case if there are two leafs l_1 and l_2 for which the DT T enforces the order $l_1 \prec l_2$, but l_1 is mapped to a time-step i and l_2 to a time-step $j < i$. This situation can be forbidden by clauses using the $\mathtt{b}_{c_2}^{c_1}$ atoms [C6].

[7] Technically any order suffices, but empirically it is better to use a topological order.

The most significant drawback of the presented encoding is its size. Checking that the linearisation does not violate the ordering constraints induced by the applied decomposition methods still requires $\Theta(n^4)$ clauses. Note that the rest of the formula contains only $\Theta(n^2)$ clauses. Due to the disparity in number between the order-related clauses and those pertaining to decomposition and executability, they will dominate the encoding causing the formula (potentially) to be harder to solve for SAT solvers.

4.4.2. SOGs – Solution Order Graphs

Checking whether the chosen linearisation of the leafs of T/P_K complies with the order imposed by the Decomposition Tree is the largest and thus most problematic part of the formula. Thus, we developed a technique to reduce its size. The naive encoding in the previous section exploits only the structure of decomposition in the planning problem, but not the structure of ordering constraints. We introduce a compact representation of the ordering of all potential yields of Decomposition Trees T of depth less than K. This ordering is a super-graph of all possible task networks that can be obtained via K-depth-limited decomposition from the initial task. We call it the Solution Order Graph (SOG) [C2].

Formally a Solution Order Graph S is a transitively closed DAG[8] whose vertices are the leafs of P_K. Let l_1 and l_2 be two leafs of P_K, $a = lca(l_1, l_2)$ their last common ancestor, and c_1 and c_2 the children of a that are ancestors of l_1 and l_2, respectively. We call the SOG S consistent with respect to P_K if for every pair of vertices l_1 and l_2 the presence of an edge between them is consistent with the applicable methods. If all methods applicable to a either impose the ordering $c_1 \prec c_2$ or do not assign tasks to either c_1 or c_2, then the presence of the edge (l_1, l_2) is consistent. If all methods applicable to a do not impose the order $c_1 \prec c_2$ or $c_2 \prec c_1$ or do not assign tasks to either c_1 or c_2, then the absence of the edge (l_1, l_2) is consistent. Any other situation is inconsistent. A consistent SOG thus requires that all methods applicable to each inner node treat their subtasks "uniformly" when it comes to the order imposed to them. This uniformity is preserved by a method not assigning a task to one of the children at all. Given a SOG S that is consistent with a PDT P_K, we know that if tasks are assigned to two leafs l_1 and l_1 of P_K their order in the yield of the represented Decomposition Tree will be the same as in S – due to the fact that the sole method that determines the order between them has forced the ordering contained in S [C2]. Note that S does not contain orderings that are not implied by the applied methods, i.e. it represents exactly the imposed constraints. The important observation is that this is independent of the actually applied methods. Consequently, the ordering of l_1 and l_2 is not dependent on any decision variable in the encoded formula any more. We will exploit this property to simplify the validity check for the linearisation.

At this point, the question is how an SOG S that is consistent with a PDT P_K can be computed. Unfortunately, not every PDT has a consistent SOG S. However, we can derive a necessary condition for a PDT P_K to have a consistent SOG S. Consider an inner node a with two children c_1 and c_2 and two methods that would impose different

[8] A DAG is a Directed Acyclic Graph, i.e. a directed graph that does not contain a directed cycle.

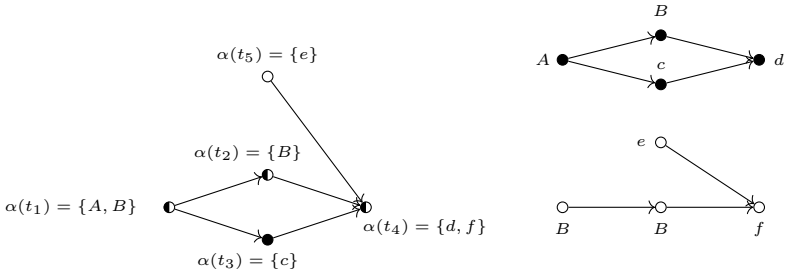


Figure 4.6.: On the right, we depict two task networks. On the left, we show a possible common supergraph G such that both task networks are induced subgraphs of G. Copied from [C2]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

orderings between c_1 and c_2. If such vertices existed, any two leafs l_1 and l_2 below c_1 and c_2 would violate the consistency of S. We can represent this local consistency of the order imposed to the children of an inner node a as a graph problem. The task network of each method m applicable to a task in $\alpha(v)$ can be equivalently viewed as a vertex-labelled DAG D_m. The children of a also form a DAG G whose nodes are labelled with sets of tasks – the $\alpha(\cdot)$ sets. If the PDT has a consistent SOG S then all D_m are induced subgraphs of G. We showed that this condition is also sufficient, i.e. if for every inner node there is a DAG G over its children for which all D_m are induced subgraphs, we can compute an SOG S for the respective TDG P_K [C2, Theorem 2]. Two task network graphs D_m and a possible DAG G for the children are depicted in Figure 4.6.

We can construct the SOG together with the PDT. When constructing the PDT, we keep a current SOG S for the current leafs of the PDT. Whenever we expand a leaf l of P_K, we construct its children in conjunction with a DAG G such that for all applicable methods m, D_m is an induced subgraph of G. We add the children to P_K and replace the node l in S with G. Once the construction of P_K is completed, S will be a consistent SOG for P_K [C2].

For this construction procedure, we have to compute a set of appropriate children C and DAG G on them, given a set of applicable methods M. Obtaining any graph G with the required property is easy, as the disjoint union of all D_m for $m \in M$ will suffice. Such a choice would lead to enumerating all Decomposition Trees explicitly and is thus not feasible. Instead, we are however interested in a small DAG G that "fuses" as many tasks in the applicable methods into one. As for the case of total order, locally minimising the number of children does not lead to a globally minimal number of leafs. Further, we showed that even locally minimising the number of children is already $\mathbb{NP}$-complete and therefore globally optimising them must be at least $\mathbb{NP}$-hard.[9]

Definition 12 (Minimal Induced Supergraph)
The decision problem MinimalInducedSupergraph *is given a family of transitively closed DAGs D_m and a number k to decide whether there exists a transitively closed DAG G with at most k vertices such that every D_m is an induced subgraph of G.*

[9] Consider a PDT for $K = 1$.

Theorem 5 (Minimal Induced Supergraph [C2])
MinimalInducedSupergraph *is* $\mathbb{NP}$*-complete.*

Currently, the DAG G is constructed in a greedy fashion [C2]. We start with G to be the first D_m and iteratively merge the next method's DAG into the current G. Each D_m is merged greedily vertex by vertex updating the set of edges and anti-edges, i.e. pairs of vertices between which no edge can be inserted into G in order to keep the induced subgraph property for every previously processed D_m. Whenever merging the next node of D_m with any of the nodes of G is not possible, we create a new vertex in G.

Note the requirement that a DAG G over the children of each inner node must exists that is consistent with the order of all applicable methods may increase the number of children as compared to the construction method used for the naive encoding, which ignored order. Thus, using the SOG-based construction, the number of leafs of the PDT might increase. Our evaluation shows that this is indeed the case, but the benefits of having extracted a SOG outweighs the increase of the number leafs.

4.4.3. From n^4 to mostly n^2 Ordering Clauses

The SOG S constructed in conjunction with the PDT P_K provides additional structural information on the planning problem that can be exploited to improve the propositional encoding. The clauses tracking order in the naive encoding, i.e. those defining $\mathtt{b}_w^v$ atoms, can be removed, as the order of the leafs is now uniquely determined. Thus reasoning on this order has not to be performed by the SAT solver. Our main objective was to replace the clauses ensuring that the chosen linearisation of the leafs of P_K adheres to the ordering imposed by the selected Decomposition Tree. Due to the construction of the SOG, this ordering is now independent of the actually chosen Decomposition Tree. As such, we may choose any linearisation of S.

As described in Section 4.4.1, the linearisation is chosen by matching each leaf of P_K to a timestep. To encode a linearisation that respects the leaf's ordering, we utilise the following observation. If we match a leaf l to a timestep t, all successors of l cannot be matched to a time before t. Requiring this condition for all leafs and timesteps is also sufficient. We add for every leaf l and timestep t an atom $\mathbf{f}_t^l$ representing that it is **f**orbidden to match leaf l to time t. Note that $\mathtt{f}_t^l$ can also be interpreted as: l must be matched to a timestep after t. We can encode the following rules that ensure a correct linearisation has been chosen, based on the above observation [C2].

1. If leaf l is matched to time t, all direct successors of l in S cannot be matched to t.
2. If leaf l cannot be matched to time t, l cannot be matched to $t-1$.
3. If leaf l cannot be matched to time t, the direct successors of l in S cannot be matched to t as well.

Encoding these three constraints requires $\mathcal{O}(L^2 \cdot \Delta^+(S))$ clauses where L is the number of leafs and $\Delta^+(S)$ is the maximum number of direct successors of any node in S. In the worst case, $\Delta^+(S)$ will be $L-1$, but in most planning problems, $\Delta^+(S)$ is typically very small, i.e. at most 3. As such, the encoding requires $\mathcal{O}(L^3)$ clauses in theory, but often

degenerates and requires only $\mathcal{O}(L^2)$ clauses in practice. This constitutes a significant improvement over the previous naive encoding in terms of formula size.

4.4.4. Attaching Modern Propositional Encodings of Classical Planning

Up to this point, we have always used the encoding of Kautz and Selman [R145] to express the executability of a linearisation of primitive actions. Since the development of this formula in 1996, there have been significant improvements to SAT planning for classical planning problems. The most significant improvements were the addition of state invariants [R139] and the ability to encoding parallel action execution [R104]. Invariant clauses are an extension to the Kautz and Selman encoding and do not require any modification to the encoding of the HTN planning problems. Further, empirically they always improved the performance of the planner. We thus used it in all configurations in the following evaluation.

The $\forall$-step[10] and $\exists$-step encodings for classical planning alter the way actions are executed in the encoding [R104]. While originally only one action could be executed per timestep, both encodings allow the for execution of multiple actions at the same time. Naturally, this is not possible for each subset Π of the set of primitive actions A. Consequently both encodings pose additional restrictions to the set of actions to be executed in parallel. While the $\forall$-step encoding requires that all linearisations are executable, the $\exists$-step encoding ensures only that at least one such linearisation of the actions in Π is actually executable. However both require that all executable linearisations result in the same state, i.e. the state after their execution is the same.

Technically, the classical encoding by Kautz and Selman uses the same variables to reference to the actions executed at each timestep as the $\exists$-step encoding. For each timestep t and action $a \in A$ the decision variable $a@t$ denotes that a is executed at time t. While the $\exists$-step encoding allows for multiple $a@t$ atoms to be true for one timestep t, the formula we presented for matching the leafs of the PDT to timesteps does not [C6]. We can relax this formula to allow for matching multiple leafs of the PDT to the same timestep [C2]. It however does not suffice to simply remove constraints from the matching, i.e. the one restricting the number of matched edges per timesteps, as matching two leafs that are labelled with the same task to the same timestep is illegal. If so, we would only execute the respective action once, while the HTN problem forced us to execute it twice, which might not be possible. We showed how this situation can be forbidden [C2]. Thus, our encoding profits from the newer and more efficient encodings of classical planning problems. Note that this modification applies both to the naive encoding, as well as to the SOG-based encoding.

4.4.5. Empirical Evaluation

We showed the encoding for totally-ordered HTN planning can be modified to also handle partial order in HTN planning problems. It is however unclear which of these encodings is suited best for solving HTN planning problems. As such, we compared them

[10] A variant of the $\forall$-step encoding is already described Kautz and Selman [R145].

empirically against each other. We considered the naive tree-style encoding SAT-tree (Section 4.4.1) and the SOG-based encoding SAT-F (Section 4.4.3). Both encodings were tested with the Kautz and Selman (the "bare" SAT-tree and SAT-F encodings) and the ∃-step encoding for primitive executability (SAT-tree ∃ and SAT-F ∃).

Benchmark Instances. For the evaluation, we use the domain presented in Section 4.3.3. Note that (some of) these domains are naturally partially-ordered. For the previous evaluation in Section 4.3.3 we have manually modified the partially ordered domains so that they are totally ordered. In this evaluation, we use the original, partially ordered ones.

Further, we added a new domain, PCP, which exploits the fact that general, partially-ordered HTN planning is undecidable [R143]. Each PCP instance encodes an instance of undecidable Post's Correspondence Problem [R160]. We chose this problem as a benchmark, as it represents one of the computationally most difficult problems HTN planners can handle.

Planners. Each planner was given 10 minutes runtime and 4 GB RAM per instance on an Intel Xeon E5-2660. We compared our encodings against the same planners[11] as in Section 4.3.3, omitting those that are specifically developed for totally-ordered instances (HTN2ASP and SHOP). We added the planner SHOP2 [R118], which is an extension of SHOP that can handle partially-ordered planning problems. While in the previous evaluation, we used the HTN2STRIPS planner only in conjunction the classical planner jasper, we here also tested it in conjunction with the best-performing planners of the 2018 International Planning Competition. These are Fast Downward Stone Soup [R56], saarplan [R51], and LAPKT-BFWS-Preference [R52]. Further, we also included the SAT-based planner MpC [R78], which uses the ∃-step encoding.

We tested our encodings in conjunction with the best-performing SAT-solvers of the SAT Competition 2018. The SAT solvers expMC [R50], cryptominisat5.5 [R57], CaDiCaL [R49], and MapleLCMDistChronoBT [R55] achieved the best performance when used together with our encoding.

Results. We show in Table 4.3 for each planner and domain the number of instances that the planners solved in the respective domain, as well as totals over all domains. We can clearly see that all the propositional encodings presented in this chapter outperform other state-of-the-art HTN planning systems and solve at least 9 instances more than the next competitor (PANDApro with the lm-cut heuristic vs. SAT-tree using CaDiCaL).

Interestingly, the SAT-tree and SAT-F encodings do not differ much in their coverage, which depending on the SAT solver either rises by 1-2 instances or falls by 2 instances. If we consider the versions of these encodings that use the ∃-step encoding for primitive executability, we can see a significant improvement. The SAT-tree encoding solves 1 to 8 more instances, while for the SAT-F encoding coverage rises by 7 to 13 instances. In total, the combination of the SAT-F and ∃-step encoding showed the best coverage. The lack of improvement from the SAT-tree to the SAT-F encoding hints at a synergy effect between the SAT-F and the ∃-step encoding, which provides an interesting avenue for

[11] Note that the original paper [C2] did not include UMCP as a planner in the evaluation. We have included it here for the sake of completeness.

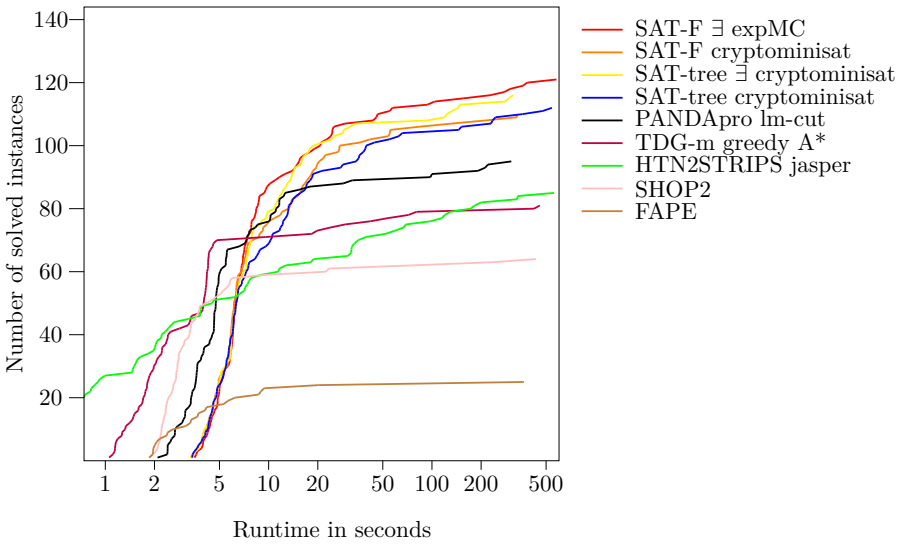


Figure 4.7.: Runtime vs number of solved instances per planner. Copied from [C2]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

future research.

In Figure 4.5 we additionally the number of solved instances per planner as a function of time. We show only the best performing configurations for each planner. Here, we observe that the planners based on our propositional encoding trail in performance for the first 5-10 seconds, as they have an initial effort in constructing the PDT, writing the formula and starting the SAT solver. However, after 10 seconds of runtime they clearly outperform all other planning systems. Further, we can see the runtime dominance of the SAT-F ∃ encoding over the other encodings.

We also checked whether the differences in runtime were actually statistically significant. Using a Kolmogorov-Smirnov Test, we found that the differences between the several SAT encodings are not statistically significant (i.e. $p < 0.05$). However, if we compare the SAT-base planners with all other planners in the comparison, the results are statistically significant. The highest p-value for comparing any planner with any of the SAT-based planners is 0.000139, when comparing SAT-F Kautz&Selman CaDiCaL and PANDApro lm-cut. If we consider only the best configuration of the SAT-based planner – SAT-F ∃ expMC – the highest p-value for any other planner is 0.0000823, again for PANDApro lm-cut.

In our evaluation, we also made interesting observations regarding the use of classical planners in conjunction with the HTN2STRIPS encoding [F34]. All currently best-performing classical planners (Fast Downward Stone Soup, SaarPlan and LAPKT-BFWS) performed significantly worse than the older japser planner. Further, MpC was the worst performing planner in our evaluation solving as many instances as FAPE which was only run on a third of all instances. The poor performance of MpC is caused by an adverse interaction between the disabling graph used in the ∃-step encoding and the HTN2STRIPS encoding. Essentially, the Disabling Graph for the HTN2STRIPS encod-

total	TRANSPORT	ROVER	ENTERTAINMENT	PCP	SMARTPHONE	WOODWORKING	SATELLITE	UM-TRANSLOG	
144	30	20	12	17	7	11	25	22	#instances
121	**22**	10	**12**	**12**	**7**	**11**	**25**	**22**	SAT-F ∃ expMC
120	20	**11**	**12**	**12**	**7**	**11**	**25**	**22**	SAT-F ∃ MapleLCM
118	20	9	**12**	**12**	**7**	**11**	**25**	**22**	SAT-F ∃ CaDiCaL
117	20	8	**12**	**12**	**7**	**11**	**25**	**22**	SAT-F ∃ cryptominisat
108	15	5	**12**	**12**	**7**	**11**	24	**22**	SAT-F expMC
108	14	6	**12**	**12**	**7**	**11**	24	**22**	SAT-F MapleLCM
107	15	4	**12**	**12**	6	**11**	**25**	**22**	SAT-F CaDiCaL
110	17	4	**12**	**12**	**7**	**11**	**25**	**22**	SAT-F cryptominisat
114	**22**	4	**12**	**12**	6	**11**	**25**	**22**	SAT-tree ∃ expMC
112	20	4	**12**	**12**	6	**11**	**25**	**22**	SAT-tree ∃ MapleLCM
111	19	4	**12**	**12**	**7**	**11**	**25**	**22**	SAT-tree ∃ CaDiCaL
116	21	6	**12**	**12**	**7**	**11**	**25**	**22**	SAT-tree ∃ cryptominisat
106	15	4	**12**	**12**	6	**11**	**25**	**22**	SAT-tree expMC
107	15	4	**12**	**12**	6	**11**	**25**	**22**	SAT-tree MapleLCM
106	15	4	**12**	**12**	6	**11**	**25**	**22**	SAT-tree CaDiCaL
112	18	5	**12**	**12**	**7**	**11**	**25**	**22**	SAT-tree cryptominisat
95	9	4	11	10	5	9	**25**	**22**	PANDApro lm-cut
95	11	3	11	9	5	9	24	**22**	PANDApro FF
93	7	4	**12**	11	5	9	23	**22**	PANDApro ADD
81	1	2	9	9	5	8	**25**	**22**	TDG-m greedy A*
78	1	2	9	8	5	10	21	**22**	TDG-c greedy A*
61	0	0	6	0	4	6	23	**22**	UMCP H
56	1	0	5	0	4	6	18	**22**	UMCP BF
57	0	0	5	0	4	6	20	**22**	UMCP DF
85	19	5	5	3	6	5	23	19	HTN2STRIPS jasper
77	17	5	5	3	6	5	19	17	HTN2STRIPS FD-SS 2018
66	13	4	5	3	5	5	14	17	HTN2STRIPS SaarPlan
63	13	4	4	3	5	5	12	17	HTN2STRIPS LAPKT-BFWS
25	3	4	4	0	4	4	0	6	HTN2STRIPS MpC
64	0	5	4	0	4	8	22	**22**	SHOP2
25/56	-	-	-	-	-	0	22	-	FAPE

Table 4.3.: Number of solved instances per planner per domain. Maxima are indicated in bold. Copied and modified (added UMCP) from [C2]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

ing will be near to a complete graph: for most pairs of two actions in the encoding, the effect of one will the other's preconditions false and vice versa. Thus the ∃-step encoding cannot execute many actions in parallel, which is one of the cornerstones of its efficiency.

To conclude our evaluation, we investigated the impact of our more compact encoding

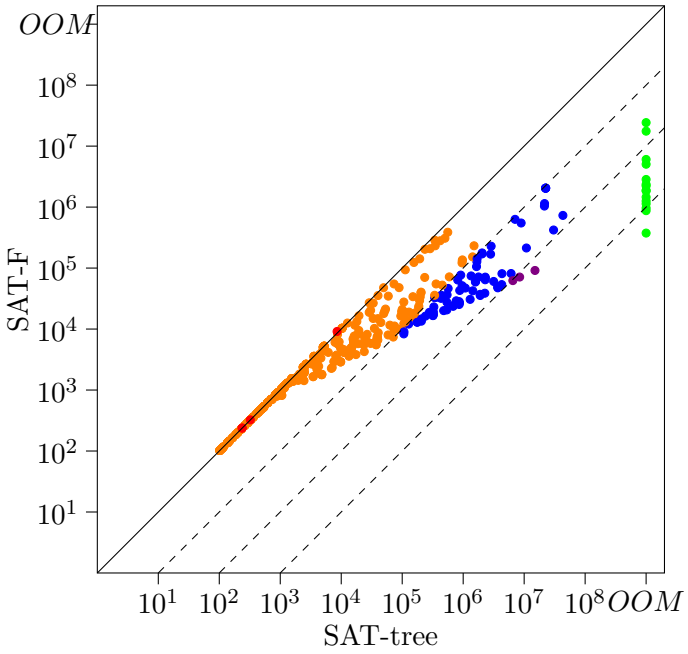


Figure 4.8.: Number of clauses in instances compared per encoding, axes are scaled logarithmically. Colours indicate orders of magnitude. OOM = Out-Of-Memory. Copied from [C2]. Reprinted with kind permission of AAAI press. AAAI does not endorse any of Ulm University's products or services.

of order (Section 4.4.3) on the size of the encoding. Although the number of clauses decreases from $\mathcal{O}(L^4)$ to $\mathcal{O}(L^3)$ in theory, an empirical evaluation is still necessary. Note that the number of clauses depends on L – the number of leafs of the PDT. Since we require an additional property for the PDTs used in the SAT-F encoding – namely that the children of each task are order-consistent – the constructed PDTs may differ between the SAT-tree and SAT-F encodings. The increase in size of the PDT necessary for the SAT-F encoding might off-set the more compact encoding. We analysed all 269 PDTs which were constructed by both encodings for the problems in the benchmark set. Out of these PDTs 68 differed in their number of leafs between the two encodings. The highest increase was 9.5%, the highest decrease 14.2%, which is in absolute numbers and increase of 16 leafs and a decrease of 5 leafs. Note that a decrease is possible, as we only greedily optimise the PDT.

Figure 4.8 contains a scatter plot indicating for each PDT the number of clauses of the formula generated based on it. To keep the sizes comparable, we only considered the encodings without the $\exists$-step encoding. We can observe that the size of the formula almost never increases – it does so in only four instances (the red dots in the plot), and in all of them only slightly. There are several instances in which the size decreases by more than a magnitude and for some even by two. Lastly, there are several instances (green dots) for which the construction of the SAT-tree formula leads to an out-of-memory error, while the SAT-F formula was still constructable. Especially the bad performance of SAT-tree in the rover domain is caused by this problem. Here 14 out of 20 instances

led to an out-of-memory error.

4.5. Finding Provably Optimal Plans

The encodings in Sections 4.3 and 4.4 were solely designed for *satisficing* planning. In satisficing planning, we only ask for *some* solution that solves the given planning problem. Consequently, this solution can – and will frequently – contain more actions than strictly necessary. For example, the planner might repeatedly insert and remove the same battery from a device – just because it is possible. We observed that especially our SAT-based planner tends to include more actions than necessary into plans.

Such plans containing redundant actions or action sequences can cause severe issues when presented to a user. As in the case of attaching and removing the battery, a human user can easily recognise the redundancy in the plan. He or she might then perceive an assistant as highly incompetent, might abort interaction, and never trust or even use the assistant again. To avert this issue, the planner should only produce plans that do not contain such redundancies. In other words, the planner should produce optimal plans, i.e. plans with the minimum number of actions required.

The SAT-based planning technique we described so far cannot directly be used to find optimal plans. The planner bounds the depth of decomposition – and not the number of actions in the plan. A depth bound implicitly also contains a bound on the number of actions, e.g. the number of leafs of the PDT. This bound only pertains to the maximum number of actions in the plan, i.e. an encoding for depth K cannot consider plans longer than the bound, but may not consider shorter plans. In general, if the shortest plan for a given depth K has length ℓ, a plan of depth $K+1$ with length $\ell-1$ can exists, i.e. an increase in the considered decomposition depth may allow for shorter plans. As an example consider a planning problem with two abstract tasks A and B and three primitive actions a, b, and c. Assume further that the problem contains three methods: $A \to abc$, $A \to aB$, and $B \to b$. If we consider the depth limit $K = 1$, the shortest possible plan contains 3 actions. For the larger depth bound $K = 2$ it is possible to find the shorter plan ab.

Instead of bounding the depth of the decomposition, we have to directly bound the number of actions. As a first step, we have to be able to directly restrict the number of actions in the propositional encoding to be smaller than a given number ℓ. This can be done by enforcing that at most ℓ of the $a@t$ atoms of the formula for primitive executability are true[12]. Since this type of constraint is fairly common in propositional modelling, compact encodings are readily available. We use the sequential encoding [R110].

Adding this constraint enables us to construct propositional formulae that represent plans up to a given length bound ℓ. To be able to ensure optimality, we however need a formula for a given ℓ that is satisfiable if and only if there exists a plan of length $\leq \ell$. To this end, we have to ensure that we choose the depth bound $K(\ell)$ for the construction of the formula appropriately, i.e. in a way so that all plans of length $\leq \ell$ can be derived

[12]Note that we are technically not restricted to simply optimising the length of the plan. We could also optimise any additive metric that assigns (rational) costs to individual actions.

with at most depth K. We will describe how this bound $K(\ell)$ is chosen in Section 4.5.1.

Lastly, we have to determine how the planner determines the optimal length of a solution. A simple option would be to construct the formula for increasing length bounds ℓ until a solution has been found. Note that we can – in general – not terminate this procedure as HTN planning is undecidable. We denote this strategy with INC(rement).

Although being simple, this strategy has a drawback, as it may require unnecessarily many calls to the SAT-solver. Technically, if the optimal plan has length ℓ^*, only the formulae for ℓ^* and $\ell^* - 1$ must be considered, where the latter proves that no better solution exists. The issue is determining ℓ^* as quickly as possible. Streeter and Smith [R97] presented several algorithms for optimising the plan length using SAT-based planners for classical planning domains. Optimising the plan length in HTN planning bears one additional difficulty, compared to classical planning: no a priori upper bound on the length of the optimal plan is known[13]. Thus, before optimising the plan length, we start by running the planner in satisficing mode, i.e. we try to obtain any solution to the planning problem. Note that this procedure is usually faster than running INC, as the plan length is (up to) exponential in the depth of decomposition. Using the length of this initially found solution as an upper bound, we run the optimisation algorithm. We tested an adaptation of Streeter and Smith's [R97] strategy S2, which is effectively binary search over the possible plan length. We denote this strategy with BIN(ary). In addition, we also tested a strategy that decreases the length bound starting with the initial upper bound until no plan is found any more, denoted DEC(rement).

4.5.1. Succinct Bounds on the Decomposition Depth

As we stated above, our SAT-based optimal HTN planner must be able to compute – for a given length bound ℓ – a depth bound $K(\ell)$ so that if no plan of length $\leq \ell$ exists with depth $\leq K(\ell)$ none will exist at all. We previously discussed this question in the context of plan verification (cf. Section 4.2.2), were we had to determine a depth bound $K(\ell)$ for a given plan of length ℓ such that if there is no decomposition of depth $\leq K$, there is none at all. For optimal planning, we need the same value $K(\ell)$, i.e. we could use the same three bounds K_{theo}, K_{unit}, and K_{TTG}. Unfortunately these bounding methods are impractically high for many HTN planning problems [C1], especially for more complex ones. The three bounding methods were only sufficient for the HTN planning problems considered at that time. Further, both K_{unit} and K_{TTG} rely on specific structures to be present in the planning problem, which is often not the case. If both methods are not applicable, we have to rely on K_{theo} which is often too high by several magnitudes.

We therefore developed a new, fourth method to compute an upper bound on the required decomposition depth $K_4(\ell)$ [C1]. This method is based on computing for every pair of plan length ℓ and task (primitive and abstract) t an individual bound $K_4(\ell, t)$. The value of $K_4(\ell, t)$ should – in contrast to $K_4(\ell)$ – be the minimum decomposition depth necessary so that if no decomposition of t into *exactly* ℓ actions with depth $\leq K_4(\ell, t)$ exists, there exists none at all. Consequently, we will select $K_4(\ell) = \max_{0 \leq m \leq \ell} K_4(m, a_I)$, where a_I is the initial abstract task.

[13] In classical planning it is $2^{|L|}$.

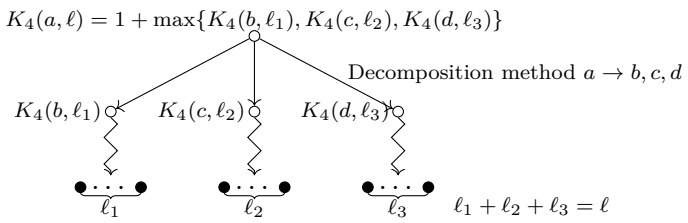


Figure 4.9.: Consideration of a new decomposition method, which increases the considered decomposition depth by one. Copied from [C1]. IJCAI. The authors have kindly been granted the right to reuse any material in other works of their own authorship. IJCAI does not endorse any of Ulm University's products or services.

The most significant obstacle the algorithm has to overcome are so-called ε-methods and unit-methods. The former are methods that contain no subtask and the latter are those that contain only a single subtask which is abstract. Both types of methods allow for arbitrarily deep decompositions for a single given plan. Our algorithm to compute $K_4(\ell, t)$ proceeds in two steps: we first consider the planning problem without such decomposition methods and compute $K_4(\ell, t)$ accordingly. Then, later on, we will account for ε- and unit-methods.

As an additional speed-up to the algorithm, we do not consider all abstract tasks simultaneously, but only those that form a Strongly Connected Component (SCC) within the Task Transition Graph (TTG). We process the SCCs S of the TTG in their reverse topological order – thus whenever computing the values $K_4(\ell, t)$ for all $t \in S$, the depth bounds for all tasks occurring in decomposition methods for tasks $t \in S$ have already been correctly computed – except for those that are themselves in S.

The algorithm for the first step iteratively updates the values of $K_4(\ell, t)$ based on previously computed values for other tasks. Each update, i.e. increase of $K_4(\ell, t)$ indicates that a new decomposition of t with length ℓ with higher depth has been found. As such, the update tries – when considering the value $K_4(\ell, t)$ – to construct a decomposition of t into ℓ actions with maximum depth. It does so by first considering all applicable decomposition methods $m_t = (t, tn_t)$. If t is to be decomposed into ℓ primitive actions starting with the method m_t, each task t' in tn_t will be decomposed into $\phi(t')$ primitive actions. To find a decomposition of t into ℓ actions, $\sum_{t' \in T(tn_t)} \phi(t') = \ell$ must hold. For such a distribution of primitive actions to subtasks ϕ, the maximum achievable depth of decomposition is $K^* = 1 + \max_{t' \in T(tn_t)} K_4(\phi(t'), \alpha(tn_t)(t'))$. This update is depicted in Figure 4.9. We can thus update $K_4(\ell, t)$ if it is smaller than K^*. To achieve completeness, we have to iterate over all such distributions ϕ of which there are exponentially many. We achieved a polynomial runtime in ℓ using dynamic programming [C1]. We further showed that if the update is performed ℓ times, the bound will be correct [C1].

The computation has so far excluded ε- and unit-methods. We include them in a second step of the algorithm, where we perform the same update mechanic as described in the previous paragraph – but only to those ε- and unit-methods. To achieve completeness, we perform the update $|S| - 1$ times for every abstract task within an SCC

S. Further, we iterate the steps one and two in total $|S| + 1$ times [C1].

4.5.2. Evaluation

To ascertain whether the proposed technique is actually efficient in practice, we compared it against other optimal HTN planners [C1]. There was however only one such planner published prior to our work: PANDA with the A* search algorithm and the TDG-c heuristic [F29].

We therefore modified both the progression search-based algorithm of PANDApro [F11, F20] and the translation into classical planning of HTN2STRIPS [F34] to allow for finding guaranteed optimal solutions. We refer to our paper for details [C1]. We else used the same setting as in the evaluation in Section 4.4.5.

The coverage results are depicted in Table 4.4 while the runtime behaviour is shown in Figure 4.10. From both we can see that the new SAT-based optimal HTN planner outperforms the other optimal planners significantly – with a margin of at least 19 instances. Between the three algorithms INC, DEC, and BIN, there is no significant difference. As expected BIN solves slightly more instances than INC and DEC. We however presume that it will get more pronounced with even better propositional encodings and SAT solvers.

Lastly, we investigated the empirical difference between the three previous methods to compute the depth bound and the new method K_4. Figure 4.11 shows for every call to the computation of the depth bound that was conducted during the evaluation both values as a scatter plot. At first, we note that K_4 never computed a worse bound than the other three methods. Even further, its bound is often far better – in some case up to three magnitudes. Lastly, there were 37 cases, shown as red dots, where the three previous methods returned a finite bound, but K_4 returned $-\infty$. Thus K_4 could prove that no plan of the given length existed, while the three older methods could not, showing the effectiveness of the algorithm in practice.

	#instances	SAT BIN			SAT DEC			SAT INC			PANDApro LM-Cut	TDG-c A*	HTN2 STRIPS	
		cryptominisat	expMV	MapleLCM	cryptominisat	expMV	MapleLCM	cryptominisat	expMV	MapleLCM			bounded maxPB	maxPB
UM-Translog	22	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	16	12
Satellite	25	**25**	**25**	**25**	**25**	24	**25**	24	23	**25**	21	21	8	6
Woodworking	11	**11**	10	**11**	**11**	10	10	10	10	10	10	8	5	5
SmartPhone	7	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	5	5	5	4
PCP	17	12	12	12	12	12	12	12	12	11	**13**	5	1	0
entertainment	12	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	5	5	4	0
rover	20	7	7	7	5	6	4	**8**	**8**	7	1	4	1	0
transport	30	**14**	11	**14**	**14**	11	**14**	13	9	12	3	2	1	0
total	144	**106**	102	**106**	104	100	102	104	99	102	80	72	41	27

Table 4.4.: Coverage of all evaluated planners on the benchmark set. Copied from [C1]. IJCAI. The authors have kindly been granted the right to reuse any material in other works of their own authorship. IJCAI does not endorse any of Ulm University's products or services.

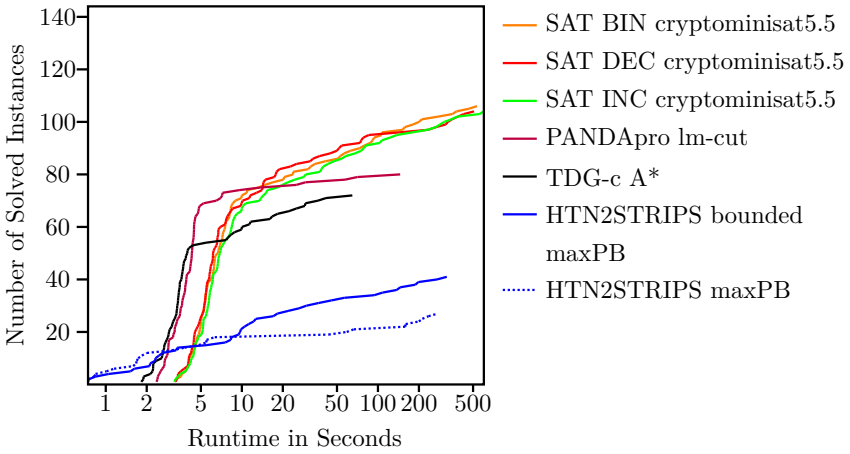


Figure 4.10.: Runtime vs Solved instances for selected planners. Copied from [C1]. IJCAI. The authors have kindly been granted the right to reuse any material in other works of their own authorship. IJCAI does not endorse any of Ulm University's products or services.

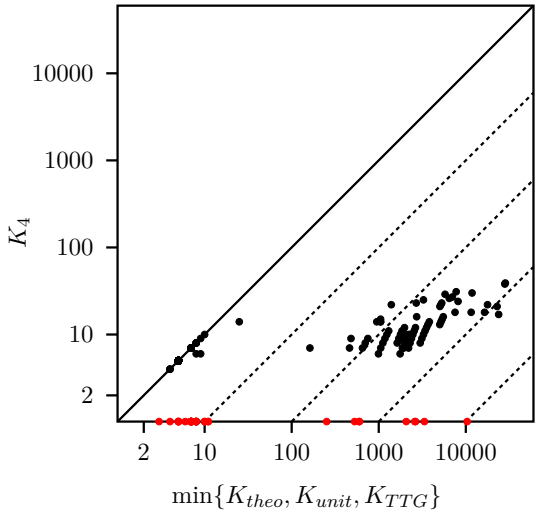


Figure 4.11.: We show for every computed depth bound the bounds K_{theo}, K_{unit}, and K_{TTG} versus K_4. Red dots indicate that the depth bound K_4 returned $-\infty$. Copied from [C1]. IJCAI. The authors have kindly been granted the right to reuse any material in other works of their own authorship. IJCAI does not endorse any of Ulm University's products or services.

4.6. Discussion and Future Work

In this chapter, we presented both the very first verifier for HTN plans, as well as an highly efficient planning technique for HTN planning. With respect to Mixed-Initiative planning, the new planner is an important step towards fulfilling the objective of providing a *fast planner* (see Section 3.3). Its performance is higher than all the currently available HTN planners on the tested benchmarks, which should also translate to real-world application domains. Specifically, we will elaborate on the performance of our planner in a DIY assistance setting in Section 5.3.

In addition to the high performance of the planner, the declarative representation of the problem for planning also allows for easily extending the presented technique. For search-based planners, such extensions usually require a modification of the search routine and several consequent changes to its heuristics. For planning approaches based on propositional logic, these extensions are relatively simple, as the additional constraints posed by these extensions have only to be added to the encoding of the problem. We will show – as one example – how constraints in Linear Temporal Logic can be integrated into the planning process in the following Section 5.2. As a further possibility, one can consider to add support for non-propositional state fluents. For example, it could be useful to model screws in a DIY domain as a numerical resource (i.e. by modelling the number of still available screws), or to model angles, thickness of a material, diameters of a hole as numerical properties, instead of as discrete properties. Such state variables can be easily handled by using SAT Modulo Theory (SMT) [R93] instead of a standard SAT solver. Such an integration was already proposed for classical planning with Planning Modulo Theories [R83] and could be adapted here. SMT allows for

formulating individual propositions in terms of expressions over a mathematical theory. For the examples in the DIY domain, a linear algebra over rational numbers suffices. By integrating SMT solving capabilities into the planner it can (presumably) cope e.g. with complex geometrical reasoning, which is, as of now, hard for a purely propositional planner.

As for the presented propositional encodings itself, there are several avenues of possible future work. For example, the PDT/SOG is currently computed in a greedy fashion. An optimisation of the PDT might further improve the performance of the planner. Similarly, the PDT is constructed uniformly up to the given depth limit, i.e. all branches of the tree are expanded up to that depth. It might be worthwhile to study show an asymmetric expansion can be utilised to focus on the difficult parts of the planning problem. Lastly, the encoding of the linearisation, which is currently the most difficult part of the encoding might be improved using either SMT modelling or GraphSAT [R77].

5. From Changing Plans to Assisting Users

5.1. Summary

In the previous chapter, we presented the first plan verifier for HTN planning. We subsequently used the basic ideas and principles of the technique to develop a new and highly efficient HTN planner. A planner used in a mixed-initiative planning-based assistance system is not only required to quickly respond, i.e. to plan quickly, but also has to be able to change plans according to the user's instructions. As we argued in Section 3.4.2, these requests to perform changes to a plan can be viewed suitably as a request to find a plan complying with a Linear Temporal Logic (LTL) formula ϕ. As such, the planner should be able to handle planning problems in conjunction with an LTL constraint ϕ resulting from a user request. The planner must thus be able to find a plan that is both a solution to the LTL constraint and to the HTN planning problem – so that it still solves the original planning problem.

As argued in Section 4.6, a propositional encoding of planning forms a good basis for further extensions of the planning process. The declarative nature of the encoding – it "simply" models what a solution is – allows for adding further constraints to the solution that have the same general structure. For example, temporal control constraints – such as LTL formulae – can be supported with relative ease by adding clauses to the formula. These constraints pertain only to the primitive actions in a solution, and thus techniques for them are applicable both to hierarchical and to classical planning. For LTL, a propositional encoding of planning with a constraint ϕ that is based on the $\exists$-step encoding for classical planning already exists [R96]. We can thus integrate it into the planner presented in the previous chapter without modification. We start by briefly describing the encoding and related approaches. Unfortunately, the encoding does not support the full set of syntactically valid LTL formulae, but only those formulae not containing the temporal next operator X. This operator is known to be difficult to handle since its presence prohibits the exploitation of stutter equivalence [R153]. It is however necessary in practice, as it is the only means to specify that a specific task has to be executed has the immediately next action – e.g. in a user request "I want to saw next." We show how the X-operator can be nevertheless integrated into the encoding [C4]. The necessary changes inspired us to modify the encoding further, which ultimately made it more efficient.

Lastly, we turn towards the practical application of planning-based assistance systems. For it, we focussed on an assistance scenario in the Do-It-Yourself (DIY) home

improvement setting. The resulting assistant, ROBERT, was developed in cooperation with Robert Bosch GmbH [C3, F18]. The objective of ROBERT is to support novices when performing small household DIY projects with the help of electronic tools, e.g. drills, saws, and sanders. ROBERT provides these novices with suitable and situation-adapted instructions on how to complete their project. Using planning techniques, we can choose the instructions presented to the user in a way that is appropriate for the tools and materials that the user has available. We will describe the application setting of ROBERT and show how its components – planner, reasoner, and dialogue manager – operate together. For a seamless interaction between these three components, their knowledge models have to be tightly intertwined. We present two means to couple ontological and planning-related knowledge and exemplify their benefits in planning-based assistance systems [C3, F33].

Core publications described in this chapter

[C3] **Gregor Behnke**, Marvin Schiller, Matthias Kraus, Pascal Bercher, Mario Schmautz, Michael Dorna, Michael Dambier, Wolfgang Minker, Birte Glimm, and Susanne Biundo. "Alice in DIY-Wonderland or: Instructing novice users on how to use tools in DIY projects". *AI Communications*, 32(1), 2019, pp. 31–57. DOI: 10.3233/AIC-180604

[C4] **Gregor Behnke** and Susanne Biundo. "X and more Parallelism: Integrating LTL-Next into SAT-based Planning with Trajectory Constraints While Allowing for Even More Parallelism". *Inteligencia Artificial*, 21(62), 2018, pp. 75–90. DOI: 10.4114/intartif.vol21iss62pp75-90

[C10] **Gregor Behnke**, Denis Ponomaryov, Marvin Schiller, Pascal Bercher, Florian Nothdurft, Birte Glimm, and Susanne Biundo. "Coherence Across Components in Cognitive Systems – One Ontology to Rule Them All". *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 2015, pp. 1442–1449

5.2. Handling User Requests via LTL

As argued in Section 3.4, users of a mixed-initiative planning system might request alterations of a currently considered plan. These changes may also require further consequential changes to the plan. Determining these changes is a computationally hard problem (see Section 3.5.2). Since this problem is as hard as planning itself, it makes sense to use a planner for it. Further, the immediate objective should not be to develop techniques to answer every possible type of request via a specialised solver, as this would incur unnecessary redundancy and additional development effort. Instead, requests should be transformed into a uniform description language and handled accordingly. We argued in Section 3.4.2 that a user's requests can be interpreted as constraint in Linear Temporal

Logic (LTL, [R155]). When integrating LTL constraints into search-based HTN planners, a significant amount of research to handle these constraints efficiently would be required. For example new heuristics would have to be developed that take these constraints accurately into account. SAT-based planners on the other hand are well suited for handling them as the formulae can be added "on-top" of the existing encoding.

5.2.1. Planning with LTL constraints

At this point, we assume that the planner is given the user's requests for alterations are already in form of an LTL formula ϕ. In a planning-based assistant, the user will input his request in natural language. The first step is thus a transformation into a formula in LTL, which we can then handle algorithmically (see Section 5.3.2).

To suitably answer requests by the user to change a plan, we have to enable the planner to find a plan that satisfies the given formula ϕ. In the past, three general types of approaches for this problem were developed, but all of them, except for the work by Sohrabi et al. [R95], were done in the context of classical planning. Since an LTL formula usually only refers to primitive actions and predicates, this does – in general – not make a difference, as the techniques for classical planning will also be applicable to HTN planning.

The first technique to handle LTL constraints is to track the validity of the LTL formula during a search procedure. In classical state-based forward search, this means to progress the formula ϕ through the applied states until it evaluates to $\top$ or $\bot$. In the former case a solution has been found, in the latter the current search node can be pruned. A planner that uses this technique is the TALPLANNER [R125]. Sohrabi et al. [R95] applied a similar technique when handling PDDL3's preferences [R109] in HTN planning. Similar techniques could be applied to progression-based HTN planners. The main issue with this technique is that existing heuristics – which are necessary for sufficiently fast planning – have to be altered in order to take the LTL formula ϕ into account.

A second option for handling these constraints is to alter the preconditions and effects of the primitive actions and the goal state, such that only plans that satisfy the formula ϕ are executable and reach the goal state. These encodings are usually based on an automata-theoretic interpretation of the formula. Most approaches transform the LTL formula ϕ into an equivalent Büchi automation B. For LTL over finite traces, i.e. finite plans [R80] the Büchi automaton B accepts the plan if it halts in an accepting state. Planning with LTL usually considers only finite traces i.e. finite plans, although work on infinite traces in planning exists [R96]. The first encoding that used these automata was proposed by Edelkamp [R115], while similar compilations were e.g. proposed by Baier and McIlraith [R98]. The drawback of all approaches that rely on Büchi automata is that the size of the constructed automaton can be exponential in the size of the formula and thus the encoding can be as well. Torres and Baier [R72] proposed to use the equivalent representation of an LTL formula ϕ as an Alternating Automaton A for encoding. Alternating Automata are a common intermediary step when constructing the Büchi automata of an LTL formula ϕ, but are linear in size with respect to the formula

ϕ [R126]. Based on them Torres and Baier [R72] presented a polynomial-size encoding.

The third option for handling LTL constraints is to encode the formula ϕ as a set of additional clauses for a SAT-based planner. To the best of our knowledge, only one such approach exists for planning [R96]. It uses a propositional encoding for LTL formulae that was developed in the context of bounded model checking and which is linear in size [R99]. A naive addition of this encoding to current propositional encodings of classical planning, i.e. the $\exists$-step encoding, leads to incorrect results, as the encoding cannot handle multiple actions occurring that the same time correctly. Instead, one would have to use the non-parallel encoding by Kautz and Selman [R145] which decreases the efficiency of the resulting planner significantly. To allow $\exists$-step parallelism in conjunction with the LTL encoding, a modification to the $\exists$-step encoding was presented that preserved correctness and completeness [R96].

Both a polynomial transformation and an encoding of LTL formulae into propositional logic seem to be suitable for a mixed-initiative planner, as both do not require (significant) alterations to the rest of the planner. To decide between these two approaches, we investigated them further. During these investigations, we showed that from a theoretical point-of-view both encodings are identical [C4]. We showed that the encoding by Mattmüller and Rintanen [R96] was – unknown to its authors – an encoding of an Alternating Automaton into propositional logic. As such, it uses the same information and exposes the same structure as the encoding by Torres and Baier [R72]. The latter however adds additional actions into the plan, which are needed to synchronise the execution of the automaton with the plan – which is not needed in a propositional encoding. This makes the former – the encoding into propositional logic – far more compact and thus our encoding of choice.

5.2.2. Improving the Propositional Encoding for LTL

The propositional encoding for LTL formulae by Mattmüller and Rintanen [R96] has a practical disadvantage: its integration with the $\exists$-step encoding is only correct if the formula does not contain the LTL next operator X. Supporting the operator X is however necessary in practice, as it is the only means to specify that a specific task has to be executed has the immediately next action – e.g. in a user request "I want to saw next."

The missing support for the operator X is caused by the way the encoding allows for parallelism. Parallelism is based on the stutter-equivalence of LTL_{-X} [R153], i.e. the fact that these formulae cannot distinguish between traces (or plans) in which an action is repeated only once or twice or more often. For example, any LTL_{-X} formula ϕ will have the same truth value on the traces *abc* and *aaabbbcccc*. If we consider any trace of actions, we can try to modify it without changing the truth of any (possible) LTL_{-X} formula. We can rename all actions in the trace that do not influence any of the atomic propositions in the formula to the first action before them that changes any of these propositions. Based on stutter equivalence, we can now allow parallel execution of these actions. As a consequence, the encoding restricts the parallel execution of actions to one that alters any proposition of ϕ followed by an arbitrary number of actions that don't.

Technically, Mattmüller and Rintanen [R96] showed that adding edges to the Disabling Graph (DG, see Section 2.5), a data structure used in the construction of the ∃-step encoding, and adding a set of clauses for every proposition symbol in ϕ suffices to achieve correctness and completeness for LTL_{-X}. For notational purposes we will denote the additional clauses for a proposition p with $\mathcal{A}(p)$.

In the presence of the X operator, LTL formulae are not stutter equivalent, thus breaking the encoding by Mattmüller and Rintanen [R96]. We proposed an alteration of their encoding that allows for handling these constraints without disallowing all parallelism. The encoding of the LTL formula ϕ into propositional logic contains a decision variable for every timestep t and subformulae ψ of ϕ that is true if ψ is satisfied by the trace starting at time t. As such, we know for every timestep whether the X operator must be evaluated at this timestep. If so, we can forbid any parallelism via an guarded at-most-one constraint [C4]. Otherwise, we can allow for the parallelism allowed by the encoding by Mattmüller and Rintanen [R96]. Using the idea of only restricting parallelism at timesteps when doing so is necessary, we have also introduced a second improvement to the encoding that enforces the $\mathcal{A}(p)$ clauses only if necessary, i.e. if the proposition p is evaluated in the LTL formula at the current timestep [C4]. This results in a less severe restriction to parallelism, which allows for plans to be found with fewer parallel timesteps, which in turn allows for a practically more efficient planner.

To show that the new encodings are correct, we could not rely on prior proofs by Mattmüller and Rintanen [R96] as they relied on stutter-equivalence. We developed a new theoretical framework, called Partial Evaluation Traces allowing us to prove the correctness of our encoding [C4]. To show that our alterations are not only correct, but also improves on the prior encoding, we conduced an empirical evaluation [C4], which confirmed our assumption.

5.3. Practical Application

To conclude this thesis, we present how planning-based assistance can be suitably applied in a real-world setting. As this setting, we chose supporting novice users in Do-It-Yourself (DIY) handiwork tasks. The assistant for this scenario, called ROBERT [C3, F18], was developed in a technology transfer project jointly with the industry partner Robert Bosch GmbH, one of the world's largest manufacturers of electronic tools. In this section, we will describe the application scenario of ROBERT and then proceed to give an overview of ROBERT's architecture and its components. Next, we elaborate on the knowledge engineering aspects of ROBERT and lastly on a subject group study with ROBERT.

5.3.1. Scenario

Users who have never or only seldom used electronic tools in the past, can be frightful of using such tools for small household tasks and Do-It-Yourself (DIY) home-improvement projects. This can, e.g. be caused by unfamiliarity with the tool or the lack of knowledge on how to handle it properly, which in turn can, e.g. cause fear of using an electronic saw. ROBERT aims at assisting such novice users by providing them suitable step-

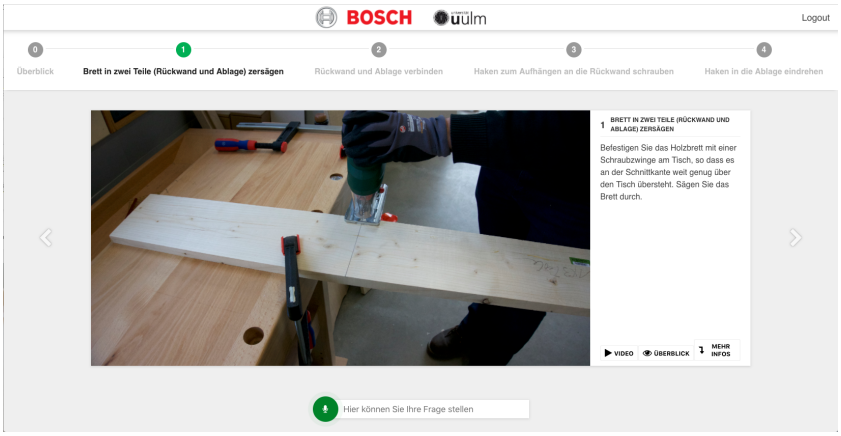


Figure 5.1.: An instruction presented by Robert to the user. Robert provides its users with a textual description of what to do, an image of the task, as well as a video showing how to perform the step.
The instructions are provided in German. A translation into English would read: "1 Cutting the plank into two pieced (back and tray). Fix the plan with a screw clamp to the table such that the cut mark is sufficiently separated from the edge of the table. Cut the plank.".
Copied from [C3]. The final publication is available at IOS Press through http://dx.doi.org/10.3233/AIC-180604.

by-step instructions on how to use electronic tools in order to achieve their objective. Using the planner presented in Chapter 4, Robert determines which actions have to be performed in order to complete the user's project. When doing so, the tools and materials available to the user are taken into account by means of representing them in the planning problem's initial state. The plan found by the planner is then presented to the user as a step-by-step instruction taking the form of a slideshow (see Figure 5.1). Each step is augmented with a suitable description in text-form, as well as with a picture and a video of the action to perform. Robert allows its users to ask questions about the presented instructions, e.g. to ask what a metal drill is and how it can be recognised (see Figure 5.2). Further, using the HTN planning domain that was used to generate the presented plan, Robert can provide the user with both a more abstract description of what to do, as well as with semantic information on where in the completion of the project the user is currently located (see bar at the top of Figure 5.1). This information also enables Robert to provide motivating feedback to the user (e.g. "You have completed step three, now the wooden frame is finished."). Lastly, Robert can react upon requests by the user to complete the project in another way than currently presented, i.e. requests to change the plan.

5.3.2. System Architecture

Robert consists of three components that interact with each other in order to provide suitable assistance to Robert's users [C3, F18]: a planner, a component for knowledge management, and a dialogue manager. The general architecture of Robert is derived

Figure 5.2.: ROBERT explains individual concepts to its user. This can both be a textual description or, e.g. in the case of explaining what a "Phillips Screw" is, an image.
In English the text in the white box would read: "2.5 Insert the drill bit. Insert the 3 millimetre metal drill bit into the electric drill. In order to do so, the rotation direction switch of the electric drill into the middle position ...".
The text in the green box reads: "A metal drill bit with a diameter of 3 millimetre can be used for drilling into metal and wood. A metal drill bit with a diameter of 3 millimetre has a conical shape (i.e. not a wedge-shaped one)". Copied from [C3]. The final publication is available at IOS Press through http://dx.doi.org/10.3233/AIC-180604.

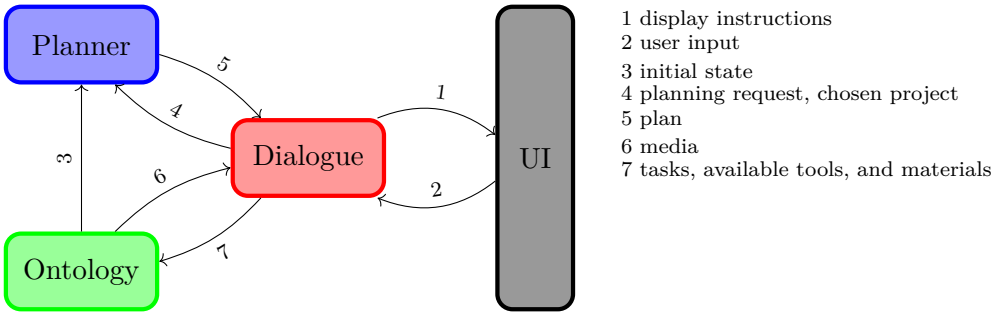


Figure 5.3.: Robert's system architecture. Each arrow describes data or instructions passed between the individual components. Copied from [C3]. The final publication is available at IOS Press through http://dx.doi.org/10.3233/AIC-180604.

from an earlier assistant developed at Ulm University [F43, F31, F40, R74]. It was designed to assist the user in setting up an ensemble of home entertainment devices. The instructions were – similar to Robert– determined by a planner and presented to the user via a dialogue management system. The previous assistant however lacked the ability to change plans based on the user's instructions (it was only able to change plans in the event of an execution error). It also lacked a dedicated knowledge base for general domain knowledge and was thus not able to explain factual background knowledge to the user.

In Robert, each component is responsible for an aspect of the assistance provided by Robert and functions are handled by the component best suited for them. The same holds for the information stored by Robert: while procedural knowledge is formulated in the planner's HTN domain model, the knowledge management component stores general domain knowledge. The general architecture and the information exchanged between the components is depicted in Figure 5.3.

At its core, Robert uses the HTN planner presented in this thesis (Chapter 4). It reasons and provides plans based on an HTN planning problem which models a wide variety of tasks in the DIY setting. We tested the domain in conjunction with several state-of-the-art HTN planning systems and found that only the SAT-based planner presented in this thesis is able to find plans within a reasonable timeframe. Once a user starts interaction with Robert, the dialogue component inquires for the user's objective and the available tools and materials. The objective is transferred to the planner, while the information on tools and materials is communicated to the knowledge manager. Based on the objective of the user, i.e. the project the user wants to complete, the planner starts its planning process, where the user's project is the goal, i.e. the initial abstract task. Since static information about the domain is necessary during the planning process, but is stored in the ontology, the planner requests such static information from the knowledge manager prior to planning.

Once a plan has been found, it is transferred to the dialogue manager, which is responsible for instructing the user based on that plan. The plan is transformed into step-by-step instructions in the form of a slideshow, where each action in the plan corre-

sponds to one slide, and presented to the user. Each action is displayed in terms of text, image, and video material. Suitable media content for each instruction is retrieved from ROBERT's knowledge-base via ontology reasoning [C3, F18]. The dialogue manager is thereafter responsible for mediating the interaction between ROBERT and its user, which is primarily performed via natural language interaction [F23]. Natural language interaction is an important feature in the DIY setting, as it enables hands-free interaction in a situation where the user will usually not be able to use his or her hands for interaction, as they are, e.g. dirty, in gloves, or he is holding a tool. The dialogue manager accepts the user's inquiries and determines, based on a machine learning model, which action to take [F23]. This includes simple interactional requests, requests for factual knowledge, as well as requests to change the current plan. Instructional requests like "Show the next step" can be handled by the dialogue manager directly. Requests for factual knowledge, e.g. "How does a Phillips Screw look like", are handled by the knowledge management component. Lastly, requests to change the current plan, e.g. "I don't want to use an electric saw", are handled by the change-request mechanism of the planner. The respective answers are subsequently computed by the responsible component and transferred to the dialogue manager for communication to the user. Note that in the case of requests to change the plan, the user's request is transformed into a formula in LTL before it is handled by the planner. This component is based on a machine learning technique and was implemented by a student under the supervision of the author of this thesis.

5.3.3. Coherent Models for Multi-Component Planning-Based Assistance Systems

ROBERT – as do almost all planning-based assistance systems (see Section 3.2.2) – consists of several components besides its planner. Each of these components might require its own model of the application domain in order to perform its functions. These models will, at least to some degree, be concerned with the same entities and facts. This necessarily creates redundancies between the models, which over time might lead to inconsistencies between them as they are modified and extended. These inconsistencies will inevitably lead to problems in the assistant. Thus, we ensure consistency of the different models by design.

Our approach to achieve model consistency is to generate or extract the models required for each component based on a common knowledge base [F37, C10]. Since no single modelling framework can provide a suitable uniform representation of all models in a planning-based assistant, the knowledge base is still divided into several separate models. We reduce redundancy as far as possible by storing only partial models in the knowledge base, only containing those information the respective component and modelling language is best suited for. Whenever necessary, we generate elements needed to complete one model automatically based on the appropriate information stored in the other models.

From the planner's point of view, the most important connection its model has, is with the domain's factual knowledge. For example the planner's model in ROBERT in only connected to the domain's factual knowledge. The planner's interaction with

ROBERT's dialogue component's is conducted via the shared terms contained in the factual knowledge base. In ROBERT, the domain's general background knowledge is stored and handled in form of an ontology based on standard first order semantics. In the following section, we will discuss two methods for integrating the knowledge stored in such an ontology with a planning model. In ROBERT, we have only used the second means for integration, as the first was not suitable for the application scenario. It has however proved useful in other application scenarios such as fitness training [F27].

5.3.3.1. Knowledge Management and Decomposition Methods

One notable feature of ontologies is that they describe a hierarchical structure of concepts. Similarly, HTN planning problems describe a hierarchy of tasks. Due to this similarity, a representation of the HTN's decomposition hierarchy in description logics seems to be a suitable candidate for knowledge integration. If successful, we could then extract the abstract tasks and decomposition methods for the HTN planning model from the ontology, eliminating the duplicated reference to them in both the planning model and the factual knowledge model.

The intuitive semantics we propose for the connection between ontologies and planning is that a concept A shall represent all possibilities to refine the represented abstract task A into primitive actions [C10]. Consequently $A \sqsubseteq B$ would hold for two concepts, if all primitive decompositions for A would also be primitive decompositions for B. This notion coincides with a decomposition method applicable to the abstract task B resulting in a task network containing only the task A – all primitive decompositions for A are also possible primitive decompositions of B. Unfortunately, this intuition fails for decomposition methods containing more than one task. Individuals contained in the ontological concepts under our intuition describe sequences or sets of objects[1] instead of atomic objects – which is a base assumption of description logics. To allow for a representation of decomposition methods with more than one task, we use a role `includes` that represents the union of individuals in a concept. For example, we denote a decomposition method for A into B and C with the axiom $\exists\texttt{includes}.B \sqcap \exists\texttt{includes}.C \sqcap \forall\texttt{includes}.(B \sqcup C) \sqsubseteq A$ [C10]. The modelling pattern used here is called onlysome and can express the combination of a set of concepts $\{C_1, \ldots, C_n\}$, but only these concepts in description logics [R102]. Formally we introduce the axiom $\mathit{onlysome}(C_1, \ldots, C_n) \sqsubseteq A$ for every method that decomposes A into a task network containing $C_1, \ldots, C_n$. We showed that modelling methods in this way correctly interacts with subsumption in description logics, i.e. that two such expression subsume each other if and only if the subconcepts can be matched to each other. Note that this modelling of decomposition methods completely ignores ordering constraints contained in the domain. This is due to the fact that DAG-style orderings cannot be represented in description logics due to the tree-model property [R127].

The main advantage of a connection between decomposition methods and abstract tasks on the one hand and concepts in description logics on the other is that inference mechanisms in description logics can be used to infer new decomposition methods. If

[1] Note that a plan is essentially a set of tasks.

a new subsumption $A \sqsubseteq B$ is derivable in the ontology, we can add a decomposition method for B into A into the planning model [C10]. We showed that we can complete a partially modelled planning domain with this mechanics. As a further benefit, we can use this connection between the planning domain and the ontology to enhance plan explanations [R84] by integrating ontology explanations into them [C10].

5.3.3.2. Knowledge Management and State Information

As a second means to separate knowledge between the planning model and the ontology, we studied how state-related information can be separated from procedural information. In contrast to the previously described mechanism, this separation is more suited to the means by which ontologies and planning describe their models. While we describe static and relational information in the ontology, we describe dynamic and procedural information in the planning model. Static knowledge relates e.g. to the types and objects occurring in the application domain – for ROBERT e.g. screws, drills, drill bits, etc. – but also to the current, i.e. the initial, state of the world. This information is stored in the ontology and transformed into information for the planner whenever a planning processes is initiated. Individuals in the ontology correspond to objects in the planning problem, concepts correspond to types, roles correspond to predicates, and axioms and assertions involving roles are interpreted as facts. Since we cannot interpret arbitrary axioms as facts for the planner's initial state, we only consider axioms of the form $\exists r.B \sqsubseteq A$ which correspond to facts $(r\ A\ B)$ in the planner's initial state [C3, F33].

We go even further with our separation of factual and procedural knowledge. Especially in the DIY domain, there is a lot of factual knowledge which is tightly coupled with procedural knowledge for handling electronic tools. This, for example, pertains to the possible configurations of electronic drills. Its configuration options – e.g. the inserted bit, drilling speed, and torque – must be adequate for the material that will be drilled into. The allowed configurations are static in the sense that they do not change over time. These configurations are naturally referenced in the preconditions of the drilling action in ROBERT's planning model. By using our approach of separating factual and procedural knowledge, we can store the allowed configurations as axioms in the ontology and translate them into facts of the planning problem's initial state. The drill action can then reference these facts and check whenever the action is to be applied, whether the facts hold [C3]. The drill action does not have to take individual configurations into account, but only a general mechanism to handle them. This way, only a single drill action is required in the planning model – as opposed to one per type of drill and type of material. This makes the planning problems significantly shorter and easier to comprehend and modify.

5.3.4. Evaluation

To show that the assistance provided by ROBERT is useful, we evaluated it with a subject group [C3, F25]. The group consistent of 18 persons, 10 females and 8 males,

with a mean age of 33.4 years[2]. Each participant constructed a wooden key rack from a provided wooden plank. Although this task seems trivial at first glance, it encompasses several steps that are difficult for novice users. They were given an electric jigsaw and an electric drill. One part of the group used the full assistance capabilities of ROBERT (13 participants), while a control group of 5 participants was provided with a basic instruction. To keep both groups comparable, the basic instructions were delivered using ROBERT's user interface in order to exclude preferences solely based on graphical presentation. The basic instructions did not instruct the participants to configure their devices, just to use them. The user interface did not respond to voice commands, showed no videos, and did not answer any questions.

In a subjective post-questionnaire, participants using ROBERT's full assistance generally perceived it as useful (3.59 on a 5-point Likert scale) and had a positive overall evaluation of the assistant (3.85 on a 5-point Likert scale). In addition to subjective measures, we have also objectively compared the participants. We considered the time needed by the participants to fully configure each electronic device they used. This measure is an indicator for the quality of the instructions provided by ROBERT as less time is presumed to be needed with better assistance. In Figure 5.4 we show the time needed by every participant to set up the electric drill and the electric jigsaw. It shows a trend for the participants using ROBERT being faster than those with only basic instructions. Note that this difference is already significant for the jigsaw (a one-way analysis of variance, ANOVA, yields $p = 0.002$).

[2]Note that the group is relatively small. The study was conducted as part of an industry transfer project with Robert Bosch GmbH. Its main focus was the investigation of engineering questions related to ROBERT. Thus, in agreement with Robert Bosch GmbH, the size of the cohort was chosen to be relatively small.

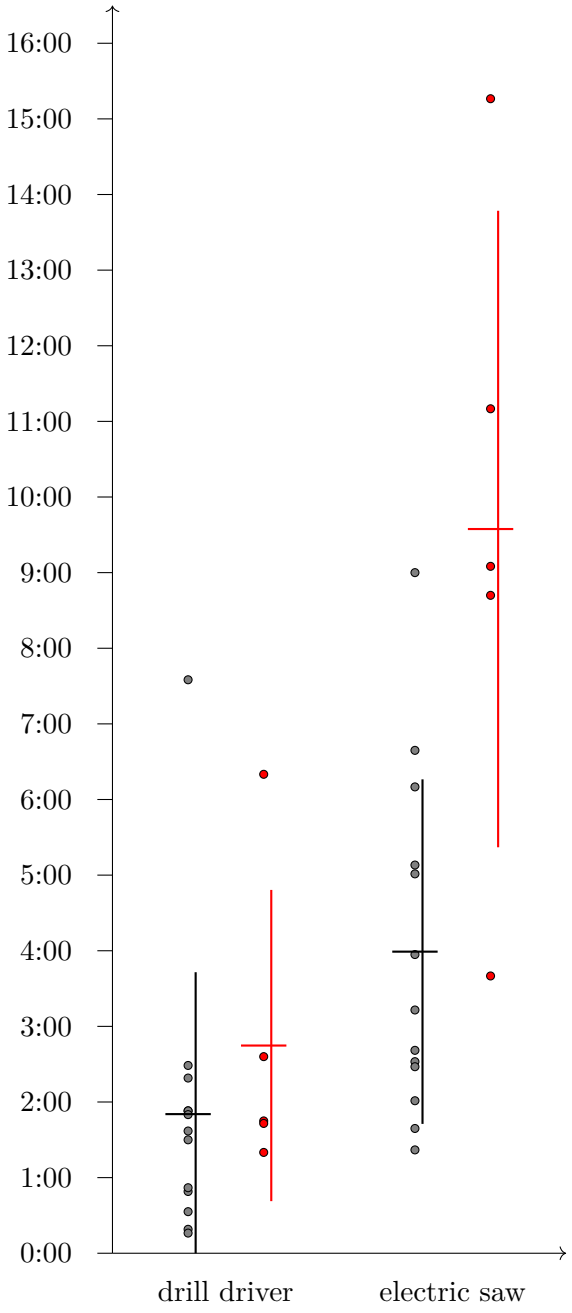


Figure 5.4.: Scatterplot of time to operation (in minutes) for electric drill driver and jigsaw; left sub-columns: full assistance, right sub-columns: baseline, horizontal bars: mean times, vertical bars: standard deviation. Copied and adapted from [C3]. The final publication is available at IOS Press through http://dx.doi.org/10.3233/AIC-180604.

6. Conclusion

In order for planners to provide the basis for useful and individualised assistance, they have to be both *fast* and *able to change plans* according to the wishes of their users. In this thesis, we have shown how both of these challenges can be addressed within hierarchical planning – a formalism that resembles domain structures and planning strategies used by humans.

We started out by studying the computational complexity of changing and verifying plans [C8, C9]. The core result of this investigation – that HTN plan verification is $\mathbb{NP}$-complete – led to the development of the first plan verifier for HTN plans [C7]. It uses a translation of the problem into propositional logic. Based on the experience with the encoding for verification, we developed an encoding for totally-ordered HTN planning problems [C5]. It is based on the new theoretical concept of Path Decomposition Trees which are a means to compactly specify the possible, K-depth-limited decompositions of a given hierarchical planning problem. The resulting planner is highly efficient and outperformed all other state-of-the-art planners on totally-ordered planning problems [C5].

Next, we extended the concept of Path Decomposition Trees such that it is also applicable to general, partially-ordered HTN planning problems [C6]. Solution Order Graphs – which can be extracted from appropriately constructed Path Decomposition Trees – allow for a compact representation of the order of primitive actions in solutions [C2]. We showed that Solution Order Graphs enable a concise encoding of partial order in HTN planning problems into propositional logic [C2]. In practical applications, it not only important to find some plan that solves the problem at hand, but also the shortest possible plan. To allow an SAT-based HTN planner to find such optimal solutions, we presented a method to compute succinct depth bounds given a length bound [C1]. Using them, a once found solution can be optimised until no shorter plan exists.

Our propositional encodings showed a significant empirical improvement over the state of the art, and – as did the planner for totally-ordered HTN planning problems – outperformed all competitors [C2]. This new planner thus constitutes an important step towards providing *fast* planners for planning-based assistance systems. The declarative nature of the planning approach we have taken constitutes a suitable foundation for future developments in HTN planning, e.g. for the integration of numeric state fluents via SAT Modulo Theories.

In order to allow our planner to handle a user's request to *change* a given plan, we first propose to view these requests uniformly as formulae in Linear Temporal Logic. Using existing encoding techniques, this allows our planner to answer change requests by the user. The existing encoding, however, does not support the full set of temporal

operators. We showed how the encoding can be modified to do so and simultaneously provided an improvement to the encoding which allowed to find plans more quickly under these constraints [C4]. Both the interpretation of requests to change a plan and our improvement to the LTL encoding are applicable both to hierarchical and to classical planning.

Lastly, we considered the application of the presented techniques in a real-world assistance scenario. We have developed the assistant ROBERT within an knowledge transfer project conducted jointly with Robert Bosch GmbH. It supports novice users in Do-It-Yourself (DIY) home improvement projects by suitably instructing them on the steps to take in order to complete their project [C3]. This, e.g. involves instructions on how to configure and use electronic tools appropriately. ROBERT uses the hierarchy of its HTN planning model to communicate instructions of different levels of abstraction to the user, which allows for a smoother interaction. We further developed methods for knowledge integration between a planning model and a logic-based ontology [C10], facilitating a coherent behaviour of all of ROBERT's components. It also enables new capabilities and eases domain modelling. In an empirical evaluation, users perceived ROBERT's assistance as helpful [C3]. They further objectively benefited from the provided assistance as they were able to perform tasks faster than a control group without ROBERT's assistance. In order to provide its assistance, ROBERT requires a highly complex planning domain, which models several aspects of real-world tasks in the DIY setting. Without the scientific advances presented in this thesis, ROBERT would not have been able to find and change plans for its users within a reasonable timeframe and would thus not have been able to assist its users.

A. Bibliography

References to bibliographic items are prefixed with a letter indicating their type. References prefixed with C are peer-reviewed core publications of this dissertation and reprinted in full in Appendix B. References prefixed with F are further publications of which I am a co-author. All F publications except for [F43] are peer-reviewed. References prefixed with R denote related work of with I am not a(n) (co-)author.

Core Publications

[C1] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "Finding Optimal Solutions in HTN Planning – A SAT-based Approach". *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*. IJCAI, 2019, pp. 5500–5508. DOI: 10.24963/ijcai.2019/764.

[C2] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "Bringing Order to Chaos – A Compact Representation of Partial Order in SAT-based HTN Planning". *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*. AAAI Press, 2019, pp. 7520–7529. DOI: 10.1609/aaai.v33i01.33017520.

[C3] **Gregor Behnke**, Marvin Schiller, Matthias Kraus, Pascal Bercher, Mario Schmautz, Michael Dorna, Michael Dambier, Wolfgang Minker, Birte Glimm, and Susanne Biundo. "Alice in DIY-Wonderland or: Instructing novice users on how to use tools in DIY projects". *AI Communications*, 32(1), 2019, pp. 31–57. DOI: 10.3233/AIC-180604.

[C4] **Gregor Behnke** and Susanne Biundo. "X and more Parallelism: Integrating LTL-Next into SAT-based Planning with Trajectory Constraints While Allowing for Even More Parallelism". *Inteligencia Artificial*, 21(62), 2018, pp. 75–90. DOI: 10.4114/intartif.vol21iss62pp75-90.

[C5] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "totSAT – Totally-Ordered Hierarchical Planning through SAT". *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*. AAAI Press, 2018, pp. 6110–6118.

[C6] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "Tracking Branches in Trees – A Propositional Encoding for solving Partially-Ordered HTN Planning Problems". *Proceedings of the 30th International Conference on Tools with Ar-*

tificial Intelligence (ICTAI 2018). IEEE Computer Society, 2018, pp. 73–80. DOI: 10.1109/ICTAI.2018.00022.

[C7] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "This is a solution! (... but is it though?) – Verifying solutions of hierarchical planning problems". *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press, 2017, pp. 20–28.

[C8] **Gregor Behnke**, Daniel Höller, Pascal Bercher, and Susanne Biundo. "Change the Plan – How hard can that be?" *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016)*. AAAI Press, 2016, pp. 38–46.

[C9] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition". *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*. AAAI Press, 2015, pp. 25–33.

[C10] **Gregor Behnke**, Denis Ponomaryov, Marvin Schiller, Pascal Bercher, Florian Nothdurft, Birte Glimm, and Susanne Biundo. "Coherence Across Components in Cognitive Systems – One Ontology to Rule Them All". *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 2015, pp. 1442–1449.

Further Co-Authored Publications

[F11] Daniel Höller, Pascal Bercher, **Gregor Behnke**, and Susanne Biundo. "On Guiding Search in HTN Planning with Classical Planning Heuristics". *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*. IJCAI, 2019.

[F12] **Gregor Behnke**, Daniel Höller, Pascal Bercher, and Susanne Biundo. "More Succinct Grounding of HTN Planning Problems – Preliminary Results". *Proceedings of the Second ICAPS Workshop on Hierarchical Planning*. 2019, pp. 40–48.

[F13] **Gregor Behnke**, Daniel Höller, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. "Hierarchical Planning in the IPC". *Proceedings of 2019 Workshop on the International Planning Competition (WIPC 2019)*. 2019.

[F14] Daniel Höller, **Gregor Behnke**, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. "HDDL – A Language to Describe Hierarchical Planning Problems". *Proceedings of the Second ICAPS Workshop on Hierarchical Planning*. 2019, pp. 6–17.

[F15] Matthias Kraus, Marvin Schiller, **Gregor Behnke**, Pascal Bercher, Susanne Biundo, Birte Glimm, and Wolfgang Minker. "A Multimodal Dialogue Framework for Cloud-Based Companion Systems". *Proceedings of the 9th International Workshop on Spoken Dialog Systems Technology (IWSDS 2018)*. Ed. by Rafael Banchs, Luis Fernando D'Haro, and Haizhou Li. Lecture Notes in Electrical Engineering. Springer, 2019.

[F16] **Gregor Behnke** and Susanne Biundo. “X and more Parallelism – Integrating LTL-Next into SAT-based Planning with Trajectory Constraints while Allowing for even more Parallelism”. *Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2018)*. 2018, pp. 1–10.

[F17] **Gregor Behnke**, Daniel Höller, and Susanne Biundo. “Tracking Branches in Trees – A Propositional Encoding for solving Partially-Ordered HTN Planning Problems”. *Proceedings of the First ICAPS Workshop on Hierarchical Planning*. 2018, pp. 40–47.

[F18] **Gregor Behnke**, Marvin Schiller, Matthias Kraus, Pascal Bercher, Mario Schmautz, Michael Dorna, Wolfgang Minker, Birte Glimm, and Susanne Biundo. “Instructing Novice Users on How to Use Tools in DIY Projects”. *Proceedings of the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence (IJCAI-ECAI 2018)*. IJCAI, 2018, pp. 5805–5807.

[F19] Daniel Höller, **Gregor Behnke**, Pascal Bercher, and Susanne Biundo. “Plan and Goal Recognition as HTN Planning”. *Proceedings of the 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*. IEEE Computer Society, 2018, pp. 466–473. *ICTAI 2018 CV Ramamoorthy Best Paper Award*.

[F20] Daniel Höller, Pascal Bercher, **Gregor Behnke**, and Biundo Biundo. “A Generic Method to Guide HTN Progression Search with Classical Heuristics”. *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press, 2018, pp. 114–122. *ICAPS 2018 Best Student Paper Award*.

[F21] Daniel Höller, Pascal Bercher, **Gregor Behnke**, and Susanne Biundo. “HTN Plan Repair Using Unmodified Planning Systems”. *Proceedings of the First ICAPS Workshop on Hierarchical Planning*. 2018, pp. 26–30.

[F22] Daniel Höller, Pascal Bercher, **Gregor Behnke**, and Susanne Biundo. “Plan and Goal Recognition as HTN Planning”. *Proceedings of the AAAI 2018 Workshop on Plan, Activity, and Intent Recognition (PAIR 2018)*. 2018, pp. 607–613.

[F23] Matthias Kraus, **Gregor Behnke**, Pascal Bercher, Marvin Schiller, Susanne Biundo, Birte Glimm, and Wolfgang Minker. “A Multimodal Dialogue Framework for Cloud-Based Companion Systems”. *Proceedings of the 10th International Workshop on Spoken Dialog Systems Technology (IWSDS 2018)*. 2018.

[F24] Benedikt Leichtmann, Pascal Bercher, Daniel Höller, **Gregor Behnke**, Susanne Biundo, Verena Nitsch, and Martin Baumann. “Towards a Companion System Incorporating Human Planning Behavior – A Qualitative Analysis of Human Strategies”. *Proceedings of the 3rd Transdisciplinary Conference on Support Technologies (TCST 2018)*. 2018, pp. 89–98. *TCST 2018 Best Paper Award*.

[F25] Marvin Schiller, **Gregor Behnke**, Pascal Bercher, Matthias Kraus, Michael Dorna, Felix Richter, Susanne Biundo, Birte Glimm, and Wolfgang Minker. “Evaluating Knowledge-Based Assistance for DIY”. *Proceedings of MCI Workshop “Digital Companion”*. 2018, pp. 925–930.

[F26] **Gregor Behnke**, Benedikt Leichtmann, Pascal Bercher, Daniel Höller, Verena Nitsch, Martin Baumann, and Susanne Biundo. "Help me make a dinner! Challenges when assisting humans in action planning". *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017)*. IEEE, 2017.

[F27] **Gregor Behnke**, Florian Nielsen, Marvin Schiller, Pascal Bercher, Matthias Kraus, Birte Glimm, Wolfgang Minker, and Susanne Biundo. "SLOTH – the Interactive Workout Planner". *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017)*. IEEE, 2017.

[F28] **Gregor Behnke**, Florian Nielsen, Marvin Schiller, Denis Ponomaryov, Pascal Bercher, Birte Glimm, Wolfgang Minker, and Susanne Biundo. "To Plan for the User Is to Plan With the User – Integrating User Interaction Into the Planning Process". *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by Susanne Biundo and Andreas Wendemuth. Cognitive Technologies. Springer, 2017. Chap. 7, pp. 123–144.

[F29] Pascal Bercher, **Gregor Behnke**, Daniel Höller, and Susanne Biundo. "An Admissible HTN Planning Heuristic". *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*. IJCAI, 2017, pp. 480–488.

[F30] Pascal Bercher, Daniel Höller, **Gregor Behnke**, and Susanne Biundo. "User-Centered Planning". *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by Susanne Biundo and Andreas Wendemuth. Cognitive Technologies. Springer, 2017. Chap. 5, pp. 79–100.

[F31] Pascal Bercher, Felix Richter, Thilo Hörnle, Thomas Geier, Daniel Höller, **Gregor Behnke**, Florian Nielsen, Frank Honold, Felix Schüssel, Stephan Reuter, Wolfgang Minker, Michael Weber, Klaus Dietmayer, and Susanne Biundo. "Advanced User Assistance for Setting Up a Home Theater". *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by Susanne Biundo and Andreas Wendemuth. Cognitive Technologies. Springer, 2017. Chap. 24, pp. 485–491.

[F32] Florian Nothdurft, Pascal Bercher, **Gregor Behnke**, and Wolfgang Minker. "User Involvement in Collaborative Decision-Making Dialog Systems". *Dialogues with Social Robots: Enablements, Analyses, and Evaluation*. Ed. by Kristiina Jokinen and Graham Wilcock. This book chapter was accepted at the 7th International Workshop On Spoken Dialogue Systems (IWSDS 2016). Springer, 2017, pp. 129–141.

[F33] Marvin Schiller, **Gregor Behnke**, Mario Schmautz, Pascal Bercher, Matthias Kraus, Michael Dorna, Wolfgang Minker, Birte Glimm, and Susanne Biundo. "A Paradigm for Coupling Procedural and Conceptual Knowledge in Companion Systems". *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017)*. IEEE, 2017.

[F34] Ron Alford, **Gregor Behnke**, Daniel Höller, Pascal Bercher, Susanne Biundo, and David W. Aha. "Bound to Plan: Exploiting Classical Heuristics via Automatic Translations of Tail-Recursive HTN Problems". *Proceedings of the 26th International Conference on Automated Planning and Scheduling, (ICAPS 2016)*. AAAI Press, 2016, pp. 20–28.

[F35] Pascal Bercher, Daniel Höller, **Gregor Behnke**, and Susanne Biundo. "More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks". *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 225–233.

[F36] Daniel Höller, **Gregor Behnke**, Pascal Bercher, and Susanne Biundo. "Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages". *Proceedings of the 26th International Conference on Automated Planning and Scheduling, (ICAPS 2016)*. AAAI Press, 2016, pp. 158–165.

[F37] **Gregor Behnke**, Pascal Bercher, Susanne Biundo, Birte Glimm, Denis Ponomaryov, and Marvin Schiller. "Integrating Ontologies and Planning for Cognitive Systems". *Proceedings of the 28th International Workshop on Description Logics (DL 2015)*. CEUR Workshop Proceedings, 2015, pp. 338–360.

[F38] **Gregor Behnke**, Marvin Schiller, Denis Ponomaryov, Florian Nothdurft, Pascal Bercher, Wolfgang Minker, Birte Glimm, and Susanne Biundo. "A Unified Knowledge Base for Companion-Systems – A Case Study in Mixed-Initiative Planning". *Proceedings of the First International Symposium on Companion Technology (ISCT 2015)*. 2015, pp. 43–48.

[F39] Pascal Bercher, Daniel Höller, **Gregor Behnke**, and Susanne Biundo. "User-Centered Planning – A Discussion on Planning in the Presence of Human Users". *Proceedings of the First International Symposium on Companion Technology (ISCT 2015)*. 2015, pp. 79–82.

[F40] Pascal Bercher, Felix Richter, Thilo Hörnle, Thomas Geier, Daniel Höller, **Gregor Behnke**, Florian Nothdurft, Frank Honold, Wolfgang Minker, Michael Weber, and Susanne Biundo. "A Planning-based Assistance System for Setting Up a Home Theater". *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, 2015, pp. 4264–4265.

[F41] Florian Nothdurft, **Gregor Behnke**, Pascal Bercher, Susanne Biundo, and Wolfgang Minker. "The Interplay of User-Centered Dialog Systems and AI Planning". *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL 2015)*. Association for Computational Linguistics, 2015, pp. 344–353.

[F42] Daniel Höller, **Gregor Behnke**, Pascal Bercher, and Susanne Biundo. "Language Classification of Hierarchical Planning Problems". *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. Vol. 263. IOS Press, 2014, pp. 447–452.

Further Co-Authored Publications (non peer-reviewed)

[F43] Pascal Bercher, Felix Richter, Frank Honold, Florian Nielsen, Felix Schüssel, Thomas Geier, Thilo Hörnle, Stephan Reuter, Daniel Höller, **Gregor Behnke**, Klaus Dietmayer, Wolfgang Minker, Michael Weber, and Susanne Biundo. *A Companion-System Architecture for Realizing Individualized and Situation-*

Adaptive User Assistance. technical report. Ulm University, 2018. DOI: 10.18725/OPARU-11023.

Related Work

[R44] Pascal Bercher, Ron Alford, and Daniel Höller. "A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations". *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*. IJCAI, 2019.

[R45] Sunandita Patra, Malik Ghallab, Dana Nau, and Paolo Traverso. "Acting and planning using operational models". *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*. AAAI Press, 2019.

[R46] Dominik Schreiber, Tomáš Balyo, Damien Pellier, and Humbert Fiorino. "Tree-REX: SAT-based Tree Exploration for Efficient and High-Quality HTN Planning". *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*. AAAI Press, 2019, pp. 382–390.

[R47] Mohammad Abdulaziz and Peter Lammich. "A Formally Verified Validator for Classical Planning Problems and Solutions". *Proceedings of the 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*. IEEE Computer Society, 2018, pp. 474–479.

[R48] Roman Barták, Adrien Maillard, and Rafael C. Cardoso. "Validation of Hierarchical Plans via Parsing of Attribute Grammars". *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press, 2018.

[R49] Armin Biere. "CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018". *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. University of Helsinki, 2018, pp. 13–15.

[R50] Md Solimul Chowdhury, Martin Müller, and Jia-Huai You. "Description of expSAT Solvers". *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. University of Helsinki, 2018, pp. 22–23.

[R51] Maximilian Fickert, Daniel Gnad, Patrick Speicher, and Jörg Hoffmann. "SaarPlan: Combining Saarland's Greatest Planning Techniques". *IPC2018 – Classical Tracks: Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*. 2018, pp. 10–15.

[R52] Guillem Frances, Hector Geffner, Nir Lipovetzky, and Miquel Ramirez. "Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants". *IPC2018 – Classical Tracks: Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*. 2018, pp. 22–26.

[R53] Daniel Kinzler. *Extraction of Linear Temporal Logic Formulae from Natural Language*. Project Report at Ulm University. 2018.

[R54] Ugur Kuter, Robert P. Goldman, Daniel Bryce, Jacob Beal, Matthew Dehaven, Christopher S. Geib, Alexander F. Plotnick, Tramy Nguyen, and Nicholas Roehner. "XPLAN: Experiment Planning for Synthetic Biology". *Proceedings of the First ICAPS Workshop on Hierarchical Planning*. 2018, pp. 48–52.

[R55] Vadim Ryvchin and Alexander Nadel. "Maple_LCM_Dist_ChronoBT: Featuring Chronological Backtracking". *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. University of Helsinki, 2018, p. 29.

[R56] Jendrik Seipp and Gabriele Röger. "Fast Downward Stone Soup 2018". *IPC2018 – Classical Tracks: Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*. 2018, pp. 72–74.

[R57] Mate Soos. "The CryptoMiniSat 5.5 set of solvers at the SAT Competition 2018". *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. University of Helsinki, 2018, pp. 17–18.

[R58] Stephan Gocht and Tomáš Balyo. "Accelerating SAT Based Planning with Incremental SAT Solving". *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press, 2017, pp. 135–139.

[R59] Abdeldjalil Ramoul, Damien Pellier, Humbert Fiorino, and Sylvie Pesty. "Grounding of HTN Planning Domain". *International Journal on Artificial Intelligence Tools*, 26(5), 2017, pp. 1–24.

[R60] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. "Hierarchical Planning: Relating Task and Goal Decomposition with Task Sharing". *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*. AAAI Press, 2016.

[R61] Arthur Bit-Monnot, David Smith, and Minh Do. "Delete-Free Reachability Analysis for Temporal and Hierarchical Planning". *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 1698–1699.

[R62] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. "A Compilation of the Full PDDL+ Language into SMT". *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016)*. AAAI Press, 2016, pp. 79–87.

[R63] Jia Hui Liang, Chanseok Oh, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. "MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB". *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. University of Helsinki, 2016, pp. 52–53.

[R64] Norbert Manthey, Aaron Stephan, and Elias Werner. "Riss 6 Solver and Derivatives". *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. University of Helsinki, 2016, pp. 56–57.

[R65] Mate Soos. "The CryptoMiniSat 5 set of solvers at SAT Competition 2016". *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. University of Helsinki, 2016, p. 28.

[R66] Álvaro Torralba, Carlos Linares López, and Daniel Borrajo. "Abstraction Heuristics for Symbolic Bidirectional Search". *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*. AAAI Press, 2016, pp. 3272–3278.

[R67] David R. Winer and R. Michael Young. "Discourse-Driven Narrative Generation with Bipartite Planning". *Proceedings of the 9th International Natural Language Generation Conf. (INLG 2016)*. The Association for Computer Linguistics, 2016, pp. 11–20.

[R68] Ron Alford, Pascal Bercher, and David W. Aha. "Tight Bounds for HTN Planning". *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*. AAAI Press, 2015, pp. 7–15.

[R69] Ron Alford, Pascal Bercher, and David W. Aha. "Tight Bounds for HTN Planning with Task Insertion". *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. 2015, pp. 1502–1508.

[R70] Steve Chien, Gregg Rabideau, Daniel Tran, Martina Troesch, Joshua Doubleday, Federico Nespoli, Miguel Perez Ayucar, Marc Costa Sitja, Claire Vallat, Bernhard Geiger, Nico Altobelli, Manuel Fernandez, Fran Vallejo, Rafael Andres, and Michael Kueppers. "Activity-based Scheduling of Science Campaigns for the Rosetta Orbiter". *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 2015, pp. 4416–4422.

[R71] Arman Masoumi, Megan Antoniazzi, and Mikhail Soutchanski. "Modeling Organic Chemistry and Planning Organic Synthesis". *Proceedings of the First Global Conference on Artificial Intelligence (GCAI 2015)*. EasyChair, 2015, pp. 176–194.

[R72] Jorge Torres and Jorge A. Baier. "Polynomial-time Reformulations of LTL Temporally Extended Goals into Final-state Goals". *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 2015, pp. 1696–1703.

[R73] Patrick Bechon, Magali Barbier, Guillaume Infantes, Lesire. Charles, and Vincent Vidal. "HiPOP: Hierarchical Partial-Order Planning". *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS 2014)*. IOS Press, 2014, pp. 51–60.

[R74] Pascal Bercher, Susanne Biundo, Thomas Geier, Thilo Hörnle, Florian Nothdurft, Felix Richter, and Bernd Schattenberg. "Plan, Repair, Execute, Explain – How Planning Helps to Assemble your Home Theater". *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press, 2014, pp. 386–394.

[R75] Pascal Bercher, Shawn Keen, and Susanne Biundo. "Hybrid planning heuristics based on task decomposition graphs". *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014)*. AAAI Press, 2014, pp. 35–43.

[R76] Filip Dvorak, Arthur Bit-Monnot, Félix Ingrand, and Malik Ghallab. "A flexible ANML actor and planner in robotics". *Proceedings of the 4th Workshop on Planning and Robotics (PlanRob)*. 2014, pp. 12–19.

[R77] Martin Gebser, Tomi Janhunen, and Jussi Rintanen. “SAT Modulo Graphs: Acyclicity”. *Logics in Artificial Intelligence, 14th European Conference JELIA 2014*. Springer, 2014, pp. 137–151.

[R78] Jussi Rintanen. “Madagascar: Scalable Planning with SAT”. *The 2014 International Planning Competition: Description of Participating Planners, Deterministic Track*. 2014, pp. 66–70.

[R79] Fan Xie, Martin Müller, and Robert Holte. “Jasper: The art of Exploration in Greedy Best First Search”. *The 2014 International Planning Competition: Description of Participating Planners, Deterministic Track*. 2014, pp. 39–42.

[R80] Giuseppe De Giacomo and Moshe Y. Vardi. “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. AAAI Press, 2013, pp. 854–860.

[R81] Remco Straatman, Tim Verweij, Alex Champandard, Robert Morcus, and Hylke Kleve. “Game AI Pro: Collected Wisdom of Game AI Professional”. Ed. by Steven Rabin. CRC Press, 2013. Chap. Hierarchical AI for Multiplayer Bots in Killzone 3.

[R82] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo. “Improving Hierarchical Planning Performance by the Use of Landmarks”. *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012, pp. 1763–1769.

[R83] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. “Planning Modulo Theories: Extending the Planning Paradigm”. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press, 2012, pp. 65–73.

[R84] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. “Making Hybrid Plans More Clear to Human Users – A Formal Approach for Generating Sound Explanations”. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press, 2012, pp. 225–233.

[R85] David E. Smith. “Planning As an Iterative Process”. *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012, pp. 2180–2185.

[R86] Tiago Stegun Vaquero, Gustavo Costa, Flavio Tonidandel, Haroldo Igreja, Jose Reinaldo Silva, and J. Christopher Beck. “Planning and Scheduling Ship Operations on Petroleum Ports and Platforms”. *Proceedings of the 2012 Scheduling and Planning Applications woRKshop (SPARK 2012)*. 2012, pp. 8–16.

[R87] Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller, and Bernd Schattenberg. “Advanced user assistance based on AI planning”. *Cognitive Systems Research*, 12(3), 2011, pp. 219–236.

[R88] Thomas Geier and Pascal Bercher. “On the Decidability of HTN Planning with Task Insertion”. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. AAAI Press, 2011, pp. 1955–1961.

[R89] Mohamed Elkawkagy, Bernd Schattenberg, and Susanne Biundo. "Landmarks in Hierarchical Planning". *Proceedings of the 20nd European Conference on Artificial Intelligence (ECAI 2010)*. IOS Press, 2010, pp. 229–234.

[R90] Malte Helmert and Hauke Lasinger. "The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem". *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*. AAAI Press, 2010, pp. 234–237.

[R91] Scott Sanner. "Relational Dynamic Influence Diagram Language (RDDL): Language Description". Accessed March 22, 2019, http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf. 2010.

[R92] Ronald Alford, Ugur Kuter, and Dana Nau. "Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way". *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*. AAAI Press, 2009, pp. 1629–1634.

[R93] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories". *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. IOS Press, 2009. Chap. 26, pp. 825–885.

[R94] Allen Nikora and Galen Balcom. "Automated Identification of LTL Patterns in Natural Language Requirements". *Proceedings of the 20th IEEE International Conference on Software Reliability Engineering (ISSRE 2009)*. IEEE Press, 2009, pp. 185–194.

[R95] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. "HTN Planning with Preferences". *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*. AAAI Press, 2009, pp. 1790–1797.

[R96] Robert Mattmüller and Jussi Rintanen. "Planning for Temporally Extended Goals as Propositional Satisfiability". *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*. AAAI Press, 2007, pp. 1966–1971.

[R97] Matthew Streeter and Stephen Smith. "Using Decision Procedures Efficiently for Optimization". *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*. AAAI Press, 2007, pp. 312–319.

[R98] Jorge Baier and Sheila McIlraith. "Planning with First-Order Temporally Extended Goals Using Heuristic Search". *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*. AAAI Press, 2006, pp. 788–795.

[R99] Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. "Linear Encodings of Bounded LTL Model Checking". *Logical Methods in Computer Science*, 2(5), 2006, pp. 1–64.

[R100] John Bresina and Paul Morris. "Mission Operations Planning: Beyond MAPGEN". *Proceedings of the 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT 2006)*. IEEE, 2006, pp. 151–156.

[R101] Malte Helmert. “The Fast Downward Planning System”. *Journal of Artificial Intelligence Research (JAIR)*, 26, 2006, pp. 191–246.

[R102] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai Wang. “The Manchester OWL Syntax”. *Proceedings of the Workshop on OWL Experiences and Directions (OWLED 2006)*. 2006.

[R103] Maja Pesic and Wil van der Aalst. “A Declarative Approach for Flexible Business Processes Management”. *Proceedings of the 2006 International Conference on Business Process Management (BPM 2006)*. Springer, 2006, pp. 169–180.

[R104] Jussi Rintanen, Kejio Heljanko, and Ilkka Niemelä. “Planning as satisfiability: parallel plans and algorithms for plan search”. *Artificial Intelligence*, 170(12-13), 2006, pp. 1031–1080.

[R105] James Thomas and Robert Michael Young. “Author in the Loop: Using Mixed-Initiative Planning to Improve Interactive Narrative”. *Proceedings of the Workshop on AI Planning for Computer Games and Synthetic Characters*. 2006, pp. 21–30.

[R106] Nate Blaylock and James Allen. “Generating Artificial Corpora for Plan Recognition”. *Proceedings of the 10th International Conference on User Modeling (UM 2005)*. Springer, 2005, pp. 179–188.

[R107] Simon Davies. “Planning and problem solving in well-defined domains”. *The Cognitive Psychology of Planning*. Ed. by Robin Morris and Geoff Ward. Psychology Press, 2005. Chap. 2, pp. 35–51.

[R108] Maria Fox, Richard Howey, and Derek Long. “Validating Plans in the Context of Processes and Exogenous Events”. *Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference (AAAI 2005)*. 2005, pp. 1151–1156.

[R109] Alfonso Gerevini and Derek Long. *Plan Constraints and Preferences in PDDL3*. Tech. rep. Department of Electronics for Automation, University of Brescia, 2005.

[R110] Carsten Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005)*. Vol. 3709. Springer, 2005, pp. 827–831.

[R111] Geoff Ward and Robin Morris. “Introduction to the psychology of planning”. *The Cognitive Psychology of Planning*. Ed. by Robin Morris and Geoff Ward. Psychology Press, 2005. Chap. 1, pp. 1–34.

[R112] Mitchell Ai-Chang, John Bresina, Len Charest, Adam Chase, Jennifer Hsu, Ari Jonsson, Bob Kanefsky, Paul Morris, Kanna Rajan, Jeffrey Yglesias, Brian Chafin, Willian Dias, and Pierre Maldague. “MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission”. *Intelligent Systems, IEEE*, 19(1), 2004, pp. 8–12.

[R113] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Ed. by Denise E. M. Penrose. Morgan Kaufmann, 2004.

[R114] Jürgen Dix, Ugur Kuter, and Dana Nau. "Planning in answer set programming using ordered task decomposition". *Proceedings of the 26th Annual German Conference on Artificial Intelligence (KI 2003)*. Springer, 2003, pp. 490–504.

[R115] Stefan Edelkamp. "On the Compilation of Plan Constraints and Preferences". *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*. AAAI Press, 2003, pp. 374–377.

[R116] Maria Fox and Derek Long. "PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains". *Journal of Artificial Intelligence Research (JAIR)*, 20, 2003, pp. 61–124.

[R117] Karen Myers, Peter Jarvis, Mabry Tyson, and Michael Wolverton. "A Mixed-initiative Framework for Robust Plan Sketching". *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*. AAAI Press, 2003, pp. 256–265.

[R118] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. Murdock, Dan Wu, and Fusun Yaman. "SHOP2: An HTN Planning System". *Journal of Artificial Intelligence Research (JAIR)*, 20, 2003, pp. 379–404.

[R119] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*. Springer, 2002.

[R120] Maria Fox and Derek Long. "PDDL+: Modelling continuous time-dependent effects". *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space (IWPSS 2002)*. 2002.

[R121] Alfonso Gerevini and Ivan Serina. "LPG: A Planner Based on Local Search for Planning Graphs with Action Costs". *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*. AAAI Press, 2002, pp. 12–22.

[R122] Karen L. Myers, W. Mabry Tyson, Michael J. Wolverton, Peter A. Jarvis, Thomas J. Lee, and Marie desJardins. "PASSAT: A User-centric Planning Framework". *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space (IWPSS 2002)*. 2002.

[R123] Stanley H. Ambrose. "Paleolithic Technology and Human Evolution". *Science*, 291(5509), 2001, pp. 1748–1753.

[R124] Susanne Biundo and Bernd Schattenberg. "From Abstract Crisis to Concrete Relief – A Preliminary Report on Combining State Abstraction and HTN Planning". *Proceedings of the 6th European Conference on Planning (ECP 2001)*. AAAI Press, 2001, pp. 157–168.

[R125] Patrick Doherty and Jonas Kvarnström. "TALPLANNER – A temporal logic-based planner". *The AI Magazine*, 22(3), 2001, pp. 95–102.

[R126] Paul Gastin and Denis Oddoux. "Fast LTL to Büchi Automata Translation". *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*. Springer-Verlag, 2001, pp. 53–65.

[R127] Erich Grädel. "Why Are Modal Logics So Robustly Decidable?" *Current Trends in Theoretical Computer Science*. Ed. by B. Păun, G. Rozenberg, and A. Salomaa. World Scientific Publishing Co., Inc., 2001, pp. 393–408.

[R128] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". *Journal of Artificial Intelligence Research (JAIR)*, 14, 2001, pp. 2531–302.

[R129] Héctor Muñoz-Avila, David Aha, Dana Nau, Rosina Weber, Len Breslow, and Fusun Yaman. "SiN: Integrating Case-based Reasoning with Task Decomposition". *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*. AAAI Press, 2001, pp. 999–1004.

[R130] Drew McDermott. "The 1998 AI Planning Systems Competition". *AI Magazine*, 21(2), 2000, pp. 35–55.

[R131] Dana Nau, Michael Ball, John Baras, Abdur Chowdhury, Edward Lin, Jeff Meyer, Ravi Rajamani, John Splain, and Vinai Trichur. "Generating and evaluating designs and plans for microwave modules". *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 14(4), 2000, pp. 289–304. DOI: 10.1017/S0890060400144026.

[R132] Bernhard Nebel. "On the Compilability and Expressive Power of Propositional Planning Formalisms". *Journal of Artificial Intelligence Research (JAIR)*, 12, 2000, pp. 271–315.

[R133] David Wilkins. *Using the SIPE-2 Planning System – A Manual for SIPE-2, Version 6.1*. Tech. rep. 2000.

[R134] Amnon Lotem, Dana Nau, and James Hendler. "Using Planning Graphs for Solving HTN Planning Problems". *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999)*. AAAI Press, 1999, pp. 534–540.

[R135] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. "SHOP: Simple hierarchical ordered planner". *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*. 1999, pp. 968–973.

[R136] George Ferguson and James Allen. "TRIPS: An Integrated Intelligent Problem-Solving Assistant". *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*. AAAI Press, 1998, pp. 567–572.

[R137] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. "Hybrid Planning for Partially Hierarchical Domains". *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*. AAAI Press, 1998, pp. 882–888.

[R138] Amol Mali and Subbarao Kambhampati. "Encoding HTN Planning in Propositional Logic". *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS 1998)*. AAAI, 1998, pp. 190–198.

[R139] Jussi Rintanen. "A planning algorithm not based on directional search". *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR 1998)*. Morgan Kaufmann Publishers, 1998, pp. 617–624.

[R140] Avrim Blum and Merrick Furst. "Fast Planning Through Planning Graph Analysis". *Artificial Intelligence*, 90(1-2), 1997, pp. 281–300.

[R141] Leonard Breslow and Aha David. *NaCoDAE: Navy Conversational Decision Aids Environment (Technical Report AIC-97-018)*. NCARAI, Washington, DC, 1997.

[R142] Manuela Veloso, Alice Mulvehill, and Michael Cox. "Rationale-Supported Mixed-Initiative Case-Based Planning". *Proceedings of the 9th Conference on Innovative Applications of Artificial Intelligence (IAAI 1997)*. AAAI Press, 1997, pp. 1072–1077.

[R143] Kutluhan Erol, James Hendler, and Dana Nau. "Complexity results for HTN planning". *Annals of Mathematics and AI*, 18(1), 1996, pp. 69–93.

[R144] George Ferguson, James Allen, and Bradford Miller. "TRAINS-95: Towards a Mixed-Initiative Planning Assistant". *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS 1995)*. 1996, pp. 70–77.

[R145] Henry Kautz and Bart Selman. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search". *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996)*. 1996, pp. 1194–1201.

[R146] Christer Bäckström and Bernhard Nebel. "Complexity Results for SAS+ Planning". *Computational Intelligence*, 11(4), 1995, pp. 625–656.

[R147] Tom Bylander. "The Computational Complexity of Propositional STRIPS Planning". *Artificial Intelligence*, 69(1-2), 1994, pp. 165–204.

[R148] Kutluhan Erol, James Hendler, and Dana Nau. "UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning". *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS)*. AAAI Press, 1994, pp. 249–254.

[R149] Michael Young, Martha Pollack, and Johanna Moore. "Decomposition and Causality in Partial-Order Planning". *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS)*. AAAI Press, 1994, pp. 188–193.

[R150] J Scott Penberthy, Daniel S Weld, et al. "UCPOP: A Sound, Complete, Partial Order Planner for ADL". *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR)*. Morgan Kaufmann, 1992, pp. 103–114.

[R151] Edwin Pednault. "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus". *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR 1989)*. Morgan Kaufmann Publishers Inc., 1989, pp. 324–332.

[R152] G. Edward Barton. "On the Complexity of ID/LP Parsing". *Computational Linguistics*, 11(4), 1985, pp. 205–218.

[R153] Leslie Lamport. "What Good is Temporal Logic?" *Proceedings of the 9th IFIP World Computer Congress*. Elsevier, 1983, pp. 657–668.

[R154] Richard Byrne. "Planning meals: Problem solving on a real data-base". *Cognition*, 5, 1977, pp. 287–332.

[R155] Amir Pnueli. “The Temporal Logic of Programs”. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*. IEEE, 1977, pp. 46–57.

[R156] Austin Tate. “Generating Project Networks”. *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI 1977)*. Morgan Kaufmann, 1977, pp. 888–893.

[R157] Stephen Cook. “The Complexity of Theorem-proving Procedures”. *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC 1971)*. ACM, 1971, pp. 151–158.

[R158] Richard Fikes and Nils Nilsson. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. *Artificial Intelligence*, 2(3–4), 1971, pp. 189–208.

[R159] Allen Newell, John Shaw, and Herbert Simon. “Report on a general problem-solving program”. *Proceedings of the International Conference on Information Processing*. Edmund C. Berkeley and Associates, 1959, pp. 256–264.

[R160] Emil Post. “A Variant of a Recursively Unsolvable Problem”. *Bulletin of the American Mathematical Society*, 52 4 1946, pp. 264–268.

[R161] Euclid. *The Elements*. Ca. 300 BCE.

[R162] Sun Tzu. *The Art of War*. Ca. 500 BCE.