On Partial Satisfaction Planning with Total-Order HTNs

Gregor Behnke[1], **David Speck**[2], **Michael Katz**[3], **Shirin Sohrabi**[3]

[1]University of Amsterdam, Amsterdam, The Netherlands
[2]Linköping University, Linköping, Sweden
[3]IBM T.J. Watson Research Center, Yorktown Heights, USA
g.behnke@uva.nl, david.speck@liu.se, michael.katz1@ibm.com, ssohrab@us.ibm.com

Abstract

Since its introduction, partial satisfaction planning (PSP), including both oversubscription (OSP) and net-benefit, has received significant attention in the classical planning community. However, hierarchical aspects have been mostly ignored in this context, although several problem domains that form the main motivation for PSP, such as the rover domain, have an inherent hierarchical structure. In this paper, we are taking the necessary steps for facilitating this research direction. First, we formally define hierarchical partial satisfaction planning problems and discuss the usefulness and necessity of this formalism. Second, we present a carefully structured set of benchmarks consisting of OSP and net-benefit problems with hierarchical structure. We describe and analyze the different domains of the benchmark set and the desiderata that are met to provide an interesting and challenging starting point for upcoming research. Third, we introduce various planning techniques that can solve hierarchical OSP problems and investigate their empirical behaviour on our proposed benchmark.

Introduction

The ability to produce solutions in constrained settings when the entire goal might not be achievable is central to automated planning applications. In classical planning, it is tackled under the general umbrella of so-called partial satisfaction planning (PSP). Two most explored problems in PSP are the net-benefit (NB) and the oversubscription planning (OSP) problems. The former assumes that the plan cost is comparable to its utility and attempts at maximizing the difference (van den Briel et al. 2004) and can be efficiently compiled into classical planning (Keyder and Geffner 2009). The latter was originally introduced by Smith (2004), on top of the classical planning setting, and multiple practical methods for solving OSP were introduced since then (Mirkis and Domshlak 2013, 2014; Muller and Karpas 2018; Katz et al. 2019a; Katz and Keyder 2022; Speck and Katz 2021; García-Olaya, de la Rosa, and Borrajo 2021). The original motivation for OSP was the rover domain, where a variety of science targets of interest exist, but the rover can only visit a few of them in any given command cycle. Note however that the domain has a natural hierarchical structure that can be exploited for efficient planning for large tasks, as the case is for many other real life planning applications. Nonetheless, the research on these two problems in partial satisfaction planning has focused on the classical setting.

In this paper, we aim at filling this gap. We extend Hierarchical Task Network planning (HTN) (e.g., (Erol, Hendler, and Nau 1994)) with partial satisfaction planning. In particular, we extend the HTN planning formalism with utilities and define computational problems for both OSP and NB over HTNs. The OSP problem is similar to preference-based planning (Baier and McIlraith 2008), where the planning problem is augmented with specification of properties that constitute a high-quality plan. The specification of high quality plan is facilitated using the PDDL3.0 (Gerevini and Long 2005; Gerevini et al. 2009) language, or its extensions (e.g., PDDL3.0 extensions for HTN Planning (Sohrabi, Baier, and McIlraith 2009)). While the general preference-based planning can deal with complex preferences (i.e., temporal properties of state trajectory), in OSP, similar to net-benefit, utilities are over final state goals and sufficient to be expressed as simple preferences (or soft goals). The cost bound in OSP is not generally within the definition of a preference-based planning problem. We note that PDDL3.0 allows expressing a net-benefit problem, as well as the oversubscription problem; we are able to specify both the cost bound constraint as well as the soft preferences in PDDL3.0.

OSP benchmarks are a rare commodity. The benchmarks used in early work are not publicly available and ready to be used. The first usable set of oversubscription planning instances slightly modifying PDDL syntax was provided by Katz et al. (2019b), and henceforth was used for the evaluation of OSP planners. Later, García-Olaya, de la Rosa, and Borrajo (2021) proposed a benchmark set, created in a similar manner, but instead of modifying the PDDL syntax, exploiting PDDL3.0 language (Gerevini and Long 2005). Following their example, we present a carefully crafted benchmark set of adapted planning domains. We describe the method we used to generate the set of instances for these domains and discuss the desired properties of such instances. We focus on OSP, where utilities and costs are incomparable, i.e., cannot be directed related or converted within the model. We further focus only on totally-ordered HTN planning problems, as for them a sufficiently large set of non-partial-satisfaction domains and problems is available due to the IPC 2020. To extend an HTN problem to an oversub-

scription HTN problem, we need to define two additional components, namely preferences with their utility values and the cost bound. While the former are defined manually in this work, coming up with a good cost bound is a difficult task, often hard to perform manually. We propose an automated, domain independent way to choose a cost bound for given preferences and utilities.

Finally, we deal with the question of solving the new problem. As there are no existing tools that can solve it, we propose two new techniques for it. Focusing on oversubscription planning and restricting our attention to totally ordered HTNs, we introduce an approach based on symbolic search, extending two previous works on oversubscription planning (Speck and Katz 2021) and on TOHTN planning (Behnke and Speck 2021). Further, we introduce an explicit heuristic search approach that can use admissible heuristics developed for ordinary HTN planning (Höller et al. 2018).

Background

We consider the HTN with preferences formalism (Sohrabi, Baier, and McIlraith 2009) and adapt it to our notion to our framework. An HTN planning problem with user preferences is described as $\Pi = (\mathcal{V}, \mathcal{P}, \mathcal{A}, \mathcal{M}, s_0, \mathcal{I}, \preceq)$ where: $\mathcal{V}$ is the set of Boolean state variables[1] (sometimes also called propositions), $\mathcal{P}$ is the set of primitive actions described by the action's name, precondition, and effect, $\mathcal{A}$ is the set of abstract tasks, $\mathcal{M}$ is the set of decomposition methods described by the method's name, abstract task, and method's task network, s_0 is the initial state, $\mathcal{I}$ is the initial task network (or initial plan), $\preceq$ is a preorder between plans. $\preceq$ is assumed to be a reflexive and transitive relation between plans. If π_1 and π_2 are plans for Π and $\pi_1 \preceq \pi_2$ then it is said that π_1 is *at least as preferred as* π_2 and $\pi_1 \prec \pi_2$ is abbreviation for $\pi_1 \preceq \pi_2$ and $\pi_2 \not\preceq \pi_1$. While the $\preceq$ relation can be defined in many ways, Sohrabi, Baier, and McIlraith (2009) proposed to described it through means of a PDDL3.0 metric function – the metric function allows to specify the utility of a given plan $u(\pi)$ and then define $\pi_1 \preceq \pi_2 \leftrightarrow u(p_1) \geq u(p_2)$. A task network is a pair (T, C) where T is a set of task nodes and C is a set of constraints. A method is *totally ordered* if its task network is *totally ordered*. A domain is a total-order domain if every $m \in \mathcal{M}$ is *totally ordered*. A plan π is a solution to Π if and only if: π is a plan for the HTN planning problem $\Pi' = (\mathcal{V}, \mathcal{P}, \mathcal{A}, \mathcal{M}, s_0, \mathcal{I})$ and there does not exist a plan π' for Π' such that $\pi' \prec \pi$. π is a plan for the HTN planning problem Π' if it only contains primitive actions, is executable in s_0, and can be derived from the initial plan $\mathcal{I}$ by repeatedly applying decomposition methods $m \in \mathcal{M}$ to the abstract tasks contained in it. We denote the state reached after executing π in s_0 with $\gamma(s_0, \pi)$.

Problem Definition

In order to extend the notion of HTN planning to the partial satisfaction setting, we need to be able to encode the utility of a plan, and for the oversubscription setting, the bound on plan cost.

[1] $\mathcal{V}$ and $\mathcal{A}$ are not part of the original definition.

Partial Satisfaction Planning with TOHTNs

We start with the net-benefit setting for fact utilities.

Definition 1 (NBHTN Planning Task) *A net-benefit Hierarchical Task Network* (NBHTN) planning task $\Pi = \langle \mathcal{V}, \mathcal{P}, \mathcal{A}, \mathcal{M}, s_0, \mathcal{I}, c, u \rangle$ *consisting of eight components – $\mathcal{V}$: state variables, $\mathcal{P}$: primitive actions, $\mathcal{A}$: abstract tasks, $\mathcal{M}$: decomposition methods, s_0: initial state, $\mathcal{I}$: initial task network, $c : \mathcal{P} \to \mathbb{N}_0$ primitive action cost function, and $u : \mathcal{S} \to \mathbb{R}$ utility function. With $\mathcal{S}$ we refer to the set of all possible states defined over $\mathcal{V}$, i.e., $\mathcal{S} = 2^{\mathcal{V}}$.*

The cost function is naturally extended onto plans via $c(a_1; a_2; \ldots; a_n) = \sum_{i=1}^{n} c(a_i)$.

The aim of net-benefit Hierarchical Task Network planning is to find a sequence of actions π that is a solution to HTN, maximizing its net-benefit $u(\gamma(s_0, \pi)) - c(\pi)$. Syntactically, we use HDDL (Höller et al. 2019), the base description language for HTN planning problems, to define the HTN part of such planning tasks. We use the PDDL3.0 syntax for preferences on top of HDDL to denote states s for which $u(s) \neq 0$ and the syntax for metrics to specify the value of that utility. Throughout the paper we will use the term preference to denote any state with $u(s) \neq 0$ and call $u(s)$ its utility. For practical reasons, we have limited ourselves to the case where preferences are described by single facts, i.e., we defined a preference as a state variable v and assigned a utility value $u(v)$ to it. The utility of a state s is then the sum of all utilities for facts that are contained in s.

We now move to the oversubscription setting.

Definition 2 (OSHTN Planning Task) *An oversubscription Hierarchical Task Network* (OSHTN) planning task $\Pi = \langle \mathcal{V}, \mathcal{P}, \mathcal{A}, \mathcal{M}, s_0, \mathcal{I}, c, u, b \rangle$ *consisting of nine components. The first eight components are as in NBHTN planning, and the last component is a cost bound $b \in \mathbb{N}$.*

The aim of a OSHTN planning task is to find a sequence of actions π, such that $c(\pi)$ is below the cost bound b and among these, the one with maximal utility $u(\gamma(s_0, \pi))$. In contrast to net-benefit planning – where it is possible to weigh costs and utilities against each other – oversubscription planning does not allow for any comparisons between costs and utilities. In (hierarchical) OSP planning, the cost bound b is a hard limit on the maximally allowed cost for a plan. The objective is then to maximise the utility of a plan – irrespective of its cost, provided it is below the cost bound. That is, the plan with the higher utility is always preferred, as opposed to the net-benefit planning, where the relative difference must be considered.

Task and State Utilities

We first discuss an alternative way to introduce utilities for HTN planning by assigning utilities to tasks and motivating the decision to assign utilities to states rather than tasks.

As in previous works that considered utilities for classical planning (Smith 2004; Mirkis and Domshlak 2013, 2014; Katz and Mirkis 2016; Muller and Karpas 2018; Katz et al. 2019a; Katz and Keyder 2022; Speck and Katz 2021; García-Olaya, de la Rosa, and Borrajo 2021), we just defined the utility of a plan to be the utility of its end state. For

HTNs, however, it also seems natural to define utilities on tasks or actions. This modelling seems to be, at first glance, more in line with the spirit of HTN planning – as objectives in HTN planning are encoded as tasks to perform.

However, both types of models are compilable into the other. Utility over tasks can be represented by utility over states by adding to the state representation the information about how often each task was executed, which is tracked by additional actions in the model. This is possible, if all tasks with non-zero utility have a non-zero implied minimal cost – otherwise an infinite utility might be reachable at zero cost. The utility function is then based on this state information. Utility over states can be represented by utility over tasks by adding tasks in the form of primitive actions whose preconditions assert that a state yielding utility has been reached and that can only be executed at the end of a plan, i.e., after all other actions. These actions then carry the utility of their associated states. The latter is similar to the compilations of Keyder and Geffner (2009) and Katz et al. (2019a). Clearly, such a compilation may lead to an increase in the size of the model. We leave a detailed theoretical analysis to future work.

We use the PDDL3.0 syntax for specifying utilities. It syntactically allows for specifying both state and task utilities (Gerevini and Long 2005; Gerevini et al. 2009). We found the formalisation using state utilities more natural and easier to handle within planners and thus used it. However, some of the benchmark domains we present (transport, barman, and robot) are at their core modelling task utilities.

Complexity Theoretic Results

Next, we prove that both net-benefit and oversubscription planning in the totally-ordered HTN setting (Definitions 1 and 2) are EXP-time complete and thus no more difficult than determining plan existence on its own. Note that for this, we need to consider the decision problem variant, i.e., we ask whether the maximum reachable utility is higher or equal to some bound U_B.

Theorem 1 *Both Net-benefit and Oversubscription Hierarchical Task Network planning are EXP-time complete, if the underlying HTN planning problems are totally-ordered.*

Proof: Hardness stems from total-order HTN planning being a special case of NBHTN planning and of OSHTN planning and from total-order HTN planning being EXP-time complete (Erol, Hendler, and Nau 1994). In what follows, we show for OSPHTN and the proof for NBHTN is analogous. Consider any total-order HTN planning problem, one can then create an OSHTN planning problem by assigning each action cost 0 and each state variable a utility of 1. If this problem has a maximum achievable utility of at least 1, then the original total-order HTN planning problem is solvable.

For membership, we can adapt the proof of Erol, Hendler, and Nau (1996). Given an OSHTN planning task, we first need to 2-regularise the problem (Höller et al. 2014; Behnke and Speck 2021). This means, that we replace all methods with more than two subtasks by a set of new methods that contain at most two subtasks. This process also introduces new abstract tasks – but only linearly many. As a result, we have obtained an equivalent, but at most linearly larger planning problem $\Pi = \langle \mathcal{V}, \mathcal{P}, \mathcal{A}, \mathcal{M}, s_0, \mathcal{I}, c, u, b \rangle$. Any algorithm that is exponential in $|\Pi|$ is also only exponential in the size of the original problem.

The decision procedure we propose will incrementally construct a set R of 4-tuples $\langle s, t, s', c \rangle$. If $\langle s, t, s', c \rangle \in R$, then either t is a primitive action of cost c that is applied to state s yields state s', or t is a abstract task that can be decomposed into a plan of cost c that if executed in state s will yield the state s'. I.e., R contains all the reachability information of the problem. After we have computed this set, we will iterate over all tuples of the form $\langle s_0, \mathcal{I}, s', c \rangle$ where c is less than the cost bound b. We then evaluate the utilities u in s' and return true, if a utility of at most U_B is reachable. Doing so is linear in the size of $|R|$.

Next, we determine the maximum size of R. Both s and s' are states of which there are $2^{|\mathcal{V}|}$ many. The task t can either be a primitive action or an abstract task, of which there are $|\mathcal{P} \cup \mathcal{A}|$. Lastly, c can be integer between 0 and b. Of these there are $b+1$ many. Since b will be encoded logarithmically in the input, there are up to $2^{|b|}$ up to many possible values for c, if $|b|$ is the length of the description of b in the input. Thus in total, R contains at most $(2^{|\mathcal{V}|})^2 \cdot |\mathcal{P} \cup \mathcal{A}| \cdot 2^{|b|} = |\mathcal{P} \cup \mathcal{A}| \cdot 2^{2|\mathcal{V}|+|b|}$ tuples.

What remains to show is that R can be computed in exponential time. Given the 2-regularised problem, we iterate over all states $s \in \mathcal{S}$ and actions $a \in \mathcal{P}$ that are applicable in s, compute the state s' resulting from applying a in s and add $\langle s, a, s', c(a) \rangle$ to R. From this on, the algorithm proceeds in rounds. In every round we iterate overall $\langle s, A, s', c \rangle \in R$. We then iterate over all elements $\langle s', B, s'', c' \rangle \in R$ such that a method decomposing a task C into A and B exists. We can then apply this method backwards and add $\langle s, C, s'', c + c' \rangle$ to R. We repeat this algorithm until a fixpoint has been reached. As we have to add at least one tuple each round, there are at most $|R|$ rounds. In each round we may iterate over all pairs of tuples in R. Thus, the overall runtime is bounded by $(|R|^3) = \mathcal{O}(|\mathcal{P} \cup \mathcal{A}|^3 2^{6|\mathcal{V}|+3|b|})$, which is only exponential. □

Benchmark Construction

Here, we describe how our benchmark set was constructed. We start with the existing benchmark set introduced for the International Planning Competition (IPC) 2020. We only considered domains from the total-order track, as more domains and planning algorithms are available for total-order.

Identifying Domains That Need Adaptation

In classical OSP, every cost-bounded applicable sequence of actions is a plan. As such, every reachable state, is a possible final state of a plan. Thus it is often the case that for each fact f there is both a plan for which f is true in the final state of the plan, and one for which f is false in the final state of the plan. Therefore, it is plausible to define utilities over all facts in the problem.

In contrast to the classical OSP, in OSHTN planning we still have to solve the underlying HTN planning problem, i.e., refine the initial plan. As such, not all states reachable

via repeated application of actions is also a possible final state for the problem. In fact, for HTN planning problems, the set of possible final states is often much smaller than the set of reachable states via action progression. As a result, it is often the case that there are facts f that are true in any final state. For example in the IPC domain transport, the facts describing the location of each package will always be set to the target location enforced by the initial plan. It is not sensible to define preferences over such facts, as making them true is unavoidable. Defining preferences, therefore, only makes sense for the facts that can be true in one final state and false in some other final state. We call these facts *flexible goal facts*. We thus need to identify domains and problem instances with a sufficient set of flexible goal facts. To determine this, we used an exhaustive search. Given the set of states $\mathcal{S}^*$ that can be reached as states after completing the problem's initial plan (i.e. as goal states in the HTN sense), we can compute $\left(\bigcup_{S\in\mathcal{S}^*} S\right) \setminus \left(\bigcap_{S\in\mathcal{S}^*} S\right)$.

To perform this exhaustive search, we used a BDD-based HTN planner (Behnke and Speck 2021), which allowed us to easily compute the required intersection. This procedure worked for some of the problems in the IPC benchmark set, while for a significant number of other problems from the IPC benchmark set, this computation failed due to time or memory exhaustion.

Based on the results of this computation, we have identified three domains that can be taken *as is* and extended with the additional components: Rover, Satellite, and Snake. For Rover, the flexible goal facts include the location of the rovers and which rover has taken which soil sample or image. For Satellite, these are the final direction to which the satellite is pointing and which instruments are currently on and calibrated. For Snake, these are the cells which are occupied by the snake's body.

For each of these domain and problem files, we do not remove or modify existing components, we only *add* preferences, utilities, and a cost bound. In particular, we keep the problem's initial task network, which enforces that only a subset of all sequences of actions are valid plans. In each of these domains, there are instances where our approach was not able to fully explore the search space. Luckily, in these domains, the variability in the final state follows a human understandable pattern, following which we were able to extrapolate what the set of final states would look like for these larger instances from the smaller ones. This allowed us to create preferences for these, more challenging instances, as well. For each flexible goal fact, we created a preference on it, and assigned that preference a utility, randomly chosen uniformly out of $[1, 20]$ ($[1, 100]$ for satellite only). In what follows, we describe the modifications made to the other three domains.

Modified Domains

The problems of the Barman, BlocksWorld, and Transport domains of the IPC 2020 admit exactly one final state – that is they fix both the goal and non-goal facts to one particular valuation. As such they cannot plausibly be used as an OSHTN planning domain without further modifications. Further the final state in the Robot domain only differs in the final location of the robot, which does not offer much variability for an OSHTN problem.

BlocksWorld The IPC 2020 domain BlocksWorld encodes in its initial plan which blocks have to be put onto which other blocks via abstract tasks. The model admits only one single state after executing the whole HTN. We introduced variability by allowing to forgo each of the block-putting tasks, by introducing a new method for them that decomposes them into the empty task sequence. For each such block-putting task, we added a corresponding preferences and selected its utility uniformly at random from $[1, 20]$.

Barman In the Barman domain, the initial plan in the IPC instances requires making of a fixed set of cocktails – we have made the making each of these cocktails optional. The preferences are then on creating a cocktail, per available cocktail. These preferences may reflect the net revenue from the cocktail. The utility for each preference has been selected uniformly at random from 10 to 10 times the number of cocktails. Further we randomly select 30% of the ingredients to be special and increase the utility of cocktails that contain special ingredients by 100.

Robot The state-based goal of the Robot domain requires certain objects to be placed in certain rooms. The HTN structure of the domain is a loop-structure using which execution can be ended at any time. We thus transferred the original state-based goals of the Robot domain into preferences, creating one preference per goal fact. The utilities were uniformly at random from $[1, 20]$.

Transport In the IPC version of Transport, the initial plan requires delivering a set of packages to a set of given locations. We modified the domain so that any package can be derived to *any* location. The preferences are then specified on package location. For each package we randomly select a set of possible target location and generate a preference for each package being at each of these locations. The utility of such a preference is the length of the shortest path from the initial package location to that location.

Over all seven domains for which we have generated problem instances with preferences and utilities, we have generated 220 instances in total.

Cost Bound

So far, we have only described how to generate the preferences and utilities for OSHTN planning problems. Both for OSHTN and for classical OSP, it is often unclear how to select the cost bound in a domain specific manner, once the preferences and their utilities are selected. We propose to compute a reasonable interval of cost bounds and then to select cost bounds for the actual benchmark instances.

Reasonable Interval of Cost Bounds We consider a cost bound to be reasonable for the benchmark if it potentially poses an interesting computational problem. We thus argue that any cost bound that is high enough to achieve the maximum overall achievable utility is not interesting – as an OSP planner in these cases can simply attempt to satisfy a set of preferences that achieve the maximum possible utility.

This would make an OSHTN/OSP problem with such a cost bound equivalent to a simple HTN/classical planning problem, which is not desired – even if an OSHTN/OSP planner would still need to prove this equivalence.

We thus propose to find the minimal cost needed for achieving the maximal achievable utility. We then set the upper bound of the interval of reasonable costs to this value minus one – as to exclude the edge case where the cost narrowly suffices to obtain the maximum utility. To compute the minimal cost c_u needed for achieving the maximal achievable utility u^*, we propose to transform the utilities to costs, using a transformation suggested by Katz et al. (2019a). In particular, we use their soft-goal compilation. This transformation adds actions pay_p that achieve a preference, while paying a penalty – the cost equal to the utility of that preference. While the transformation suggested by Katz et al. (2019a) has two cost functions, the original one c and the utility based one c', we combine them into a single cost function, multiplying the utility based cost c' by a large constant: $c + Mc'$. For a large enough M (we use $M = 10,000$), when losing any utility is more expensive than the cost of achieving preferences, the optimal cost of solving the transformed task will allow us to derive the desired value, simply by taking the cost modulo M. For each preference p, we thus add one abstract task $check_p$ to the model that has two methods: one that decomposes $check_p$ into an empty task network and one that decomposes it into the action pay_p. We then add all $check_p$ tasks to the initial plan $\mathcal{I}$ and add ordering constraints that they occur after all other original tasks in $\mathcal{I}$. Note that this compilation also bears similarities to the encoding for classical net-benefit planning into classical planning proposed by Keyder and Geffner (2009). Both it and the encoding by Katz et al. (2019a) are based on the idea of forgo (pay) and collect ($check$) actions. Adapting the encoding we just presented to the Net-Benefit setting only requires adapting the action costs to the ones proposed by Keyder and Geffner (2009). Such an encoding can be used as a baseline for future research on Net-benefit HTNs.

For the lower bound of the interval of reasonable costs, we argue that an instance in which it is impossible to achieve any non-zero utility, i.e., where it is impossible to satisfy any preference, is not interesting. Thus, we use the minimum cost $c_{>0}$ needed to satisfy any preference with non-zero utility as the lower bound. For the upper bound, we don't use the value itself, but the next larger value, as the bound for the interval of reasonable costs. We again compute the bound $c_{>0}$ using a model transformation. We add a primitive action $check_p$ for every preference p and set its precondition to p and its effects to $\emptyset$. We then add a new abstract task $Check$ that has one decomposition method per preference p, each generating the action $check_p$. Lastly, we add the $Check$ task to the initial plan and constraints that it follows the original tasks in that plan.

Computing the Bounds for the Benchmark

So far, we only described what the bounds should be and how to obtain them via model transformation. We still need to solve the resulting planning problems to obtain the bounds. Note that we need to solve these problems optimally in order to obtain the exact bounds. Only a few HTN planners support action costs and optimal planning.

We use the pandaPI progression planner (Höller et al. 2020) using A*. For optimal planning, we use the admissible LM-Cut heuristic (Helmert and Domshlak 2009). It his only applicable to classical planning, i.e., to states and not progression search states in HTN planning. The pandaPI planner uses the RC model transformation (Höller et al. 2020) to compute an approximation of the HTN progression search state, resulting in a classical planning problem and a classical state that the LM-Cut heuristic can be evaluated on. This approximate model contains actions that encode the HTN model's primitive actions as well as actions that encode the HTN model's decomposition. We have set the cost of these method actions in the RC model to zero s.t. the resulting estimate is admissible for HTNs and thus a cost-optimal planner. We further used the optimal symbolic planner (HTN-BDD) by Behnke and Speck (2021). We ran those planners for four hours per instances and a memory limit of 32GB.

Using these two planners, we could compute c_u for 58 out of the 220 generated instances. pandaPI A* LM-Cut solved 54 instances, while HTN-BDD solved 35. For $c_{>0}$, we succeeded in 179 instances, with pandaPI A* LM-Cut solving all of them and HTN-BDD solving 174. For the barman domain, we were further able to analytically determine the optimal cost to achieve the maximum preference. Of the 30 instances in this domain, only 7 were already optimally solved by the optimal HTN planners. The other 23 instances could not be solved optimally by any planner, but since we can analytically compute the optimal cost, we have obtained optimal bounds for 23 further instances.

Given the low coverage of optimal planners in these domains, we decided to forgo the requirement of exact bounds in instances where we can't compute the bounds exactly. Instead, we tried to estimate the values of c_u and $c_{>0}$ as best as possible – using satisficing planners. We employed pandaPI with greedy A* (weight 2) together with the FF (Hoffmann and Nebel 2001) and ADD heuristics (Bonet and Geffner 2001). We further used totSAT (Behnke, Höller, and Biundo 2018) and HTN2STRIPS (Alford et al. 2016). Note that there is a version of totSAT that can find guaranteed optimal plans (Behnke, Höller, and Biundo 2019). This version of the planner is not maintained anymore and does not correctly parse the input language we use for our transformation and was as thus discarded. It would, have also suffered from the issues we describe below and would thus have also required the same adaptations we applied to totSAT.

The totSAT and HTN2STRIPS planners were developed for agile planning, i.e., they attempt to find *any* solution as quickly as possible. Such a plan is usually not a good approximation on the true cost of an optimal plan.

HTN2STRIPS For HTN2STRIPS, this is caused by aborting the search once a solution was found for the smallest possible progression bound. It may however be possible to find shorter (and cheaper) plans with higher progression bound (Behnke, Höller, and Biundo 2019). We simply keep iterating over progression bounds until we reach the time limit and take the shortest plan found. HTN2STRIPS

uses a classical planner internally to search for an HTN plan. By default this is the planner jasper (Xie, Müller, and Holte 2014), which does not produce near optimal solutions. We ran HTN2STRIPS additionally with Fast Downward (Helmert 2006) using the A^* search with the LM-Cut heuristic and greedy best first search and the FF heuristic.

totSAT For totSAT, we modified the planner's overall algorithm. By default, totSAT iterates over the maximum decomposition depth K until a solution has been found and returns that solution – no matter its cost. Since the cost C of this plan is an upper bound to the true cost of an optimal plan, we modified the planner to continue, but with the restriction that the cost of the plan is limited to at most $C-1$. If we can find such a plan, we can set the upper bound to its cost and repeat the process. If there is no cheaper solution at depth K, there still might be a cheaper solution at depth $K+1$. We thus increase the depth to $K+1$ and again attempt to find a plan with cost at most $C-1$.

This technique requires us to be able to bound the maximum cost of a plan in totSAT's translation to SAT. For each action a at timestep t, we introduce decision variables $C^t_1,\dots,C^t_{c(a)}$ and add implications $a^t \rightarrow C^t_1,\dots,a^t \rightarrow C^t_{c(a)}$. I.e. choosing an action a to be in the plan at time t causes $c(a)$ atoms to be true. We then gather all such C^t_c atoms in a set $\mathfrak{C}$. To limit the overall cost of actions to some value L, we have to ensure that at most L of the atoms from $\mathfrak{C}$ are true. We use the sequential counter encoding (Sinz 2005) to encode this restriction.

This simple procedure has a problem: the size of the formula restricting the cost scales in $e \cdot C \cdot t$, where e is the maximum cost of an individual action, l is the cost limit, and t is the number of time steps. Given our compilations, e and C can both be significantly above 10^4, which makes the encoding impossible to use. Instead, we optimize the cost in a two step process. In the initial step, we set the cost of all actions to $\lfloor \frac{c(a)}{10^4} \rfloor$ and proceed as normal. If we have determined that the formula for depth K and cost limit C is not solvable, the cost of the best plan for depth K must be in $[(C+1)\cdot 10^4, (C+2)\cdot 10^4)$. From the previous call for cost $C+1$, we have a plan and can compute its cost C_o under the original model. We then add the cost limiting formula twice, once for the cost function $\lfloor \frac{c(a)}{10^4} \rfloor$ and once for the cost function $c(a) \bmod 10^4$. We limit the former to $C+1$, and iterate as described above over the latter starting with $C_o - 1 \bmod 10^4$. Once we have reached an unsolvable formula for this iteration, we proceed to the next depth $K+1$ and start iterating over the cost bound for the $\lfloor \frac{c(a)}{10^4} \rfloor$ cost model. This way, we try to approximate the cost of the cheapest plan.

Results As for the optimal planners, we ran the modified satisficing planners with a time limit of four hours per instances and a memory limit of 32GB.

For the translation to compute the upper bound c_u, the modified totSAT returned some plan in 187 out of the 220 instances while HTN2STRIPS with LM-Cut returned plans in 163 instances, HTN2STRIPS with FF in 165, and HTN2STRIPS with jasper in 172 instances. pandaPI with greedy A* with the FF heuristic solves 73 instances and with the ADD heuristic solves 88. Overall, the satisficing planners found some plan in 193 out of the 220 instances.

For the lower bound $c_{>0}$, totSAT produces plans for 202 instances, HTN2STRIPS with LM-Cut 177 and with FF 187 out of the 220 overall instances. The instances for which HTN2STRIPS finds a plan are a strict subset of the instances for which totSAT finds a plan, i.e., we have some plan for 202 instances in total. Further, pandaPI with greedy A* and the FF heuristic solves 174 instances while add solves 212. Overall, we can find plans and thus approximations for $c_{>0}$ for 218 of the 220 instances. We failed only in once instance of the domains Rover (p30) and Satellite (p17), each. Note that in these two domains, even the estimation of $c_{>0}$ still requires solving the underlying HTN which is too difficult for these two very large instances.

Generating the Bounds

We discarded the two instances for which we could not compute any estimate on the bound $c_{>0}$, as we have no way of setting the cost bound for them anyway. Additionally, we found that $c_u = c_{>0}$ or $c_u = c_{>0}+1$ for 11 instances, which were also discarded, since we cannot generate any plausible cost bound between c_u and $c_{>0}$. The remaining 207 instances were used to generate OSHTN planning instances. For 81 instances, we were able to optimally compute both c_u and $c_{>0}$. For a further 101 instances, we have approximations on both c_u and $c_{>0}$. Of these, for 63 the approximation to c_u is less than 10^4, i.e., we found a plan that does not contain any pay action. In these cases, the cost that we found is an upper bound to the true cost c_u. In the remaining 61 instances, the best plan that we found contained a pay action. For these plans their cost can be written as $b + p \cdot 10^4$. The cost b is however not an upper approximation to c_u – as it might be possible to use fewer pay actions (i.e., achieve a higher utility), which might require a more expensive plan. Lastly for 25 instances, we only have an approximation on $c_{>0}$, but not on c_u.

Since we don't have any informed means to investigate or to differentiate the different cases associated with the upper bound c_u, we treat all of them the same. For 182 instances, we thus have an upper bound c_u and a lower bound $c_{>0}$ for any sensible cost bound. We have created up to 10 OSHTN planning instances for each of the base problem instances, varying the cost bound. Since we expect the problems to become disproportionally harder with increasing cost bound, we decided to not select the cost bounds for these 10 instances linearly, but exponentially. We have thus generated the cost bounds $c_{>0} + 1 + \lfloor f \cdot (c_u - c_{>0} - 2) \rfloor$ for $f \in \{0, 0.08, 0.16, 0.25, 0.36, 0.46, 0.58, 0.71, 0.85, 1\}$. For some instances $|c_u - c_{>0} - 2|$ was not high enough to admit 10 different integer cost bounds. In these cases, we generated fewer than 10 instances. For the remaining 25 instances, for which we don't even have an estimate for u_c, we interpolated between $c_{>0}$ and $2 \cdot c_{>0}$, i.e., we used the above formula if one sets $c_u = 2 \cdot c_{>0}$.

Overall, this process has resulted in a benchmark set with 1.955 OSHTN planning instances. For a more detailed analysis of the benchmark set, we refer to Sec. Experimental Evaluation.

Solving OSHTN Planning Problem

There are multiple approaches to solving classical OSP, all of which are inspired by efficient and state-of-the-art search techniques for ordinary optimal classical planning. The most prominent techniques are explicit branch-and-bound search with heuristics that adapt the ideas of classical planning to OSP (Mirkis and Domshlak 2013; Domshlak and Mirkis 2015; Muller and Karpas 2018), reformulations of OSP as classical planning with two cost functions either for heuristic computation (Katz et al. 2019a) or to enable explicit A* search with bound-sensitive heuristics (Katz and Keyder 2022) and symbolic blind search (Eifler et al. 2020; Speck and Katz 2021). We propose a similar approach by considering and adapting two successful approaches to optimal HTN planning for OSHTN planning: explicit heuristic search and symbolic blind search. The benchmark set and source code of both planners is available online (Behnke et al. 2023).

Explicit Heuristic Search

First, we make a slight modification to the existing explicit heuristic search for optimal HTN planning (Behnke, Höller, and Biundo 2019; Höller et al. 2020). Instead of stopping the search when a goal node (i.e. we reached the empty task network) was expanded, we continue until the open list is empty, pruning successors with f value above the cost bound. This modification ensures that, assuming a given recursion depth bound (see Sec. Theoretical Properties on why such a bound is necessary), we will explore all goal states up to the specified cost bound. Given that, we can store during search the goal state with the maximal utility value observed so far. As a simple optimization, the search terminates if we encounter a goal node with the highest possible utility value.

Finally, we consider two different admissible heuristics, the blind heuristic which corresponds to an exhaustive search and the LM-cut heuristic for HTN (Helmert and Domshlak 2009; Höller et al. 2018).

Blind Symbolic Search

Behnke and Speck (2021) showed that symbolic search can be realized for HTN planning by incrementally constructing automata each representing all states and stacks, i.e., search nodes, reachable with certain costs. Speck and Katz (2021) showed that symbolic search can be used for classical OSP by representing the utility function and efficiently evaluating the reachable set of states, which are also represented as decision diagrams such as Binary Decision Diagrams (BDDs) (Bryant 1986). The same idea is possible for the Behnke and Speck (2021) approach. As Speck and Katz (2021), we use Algebraic Decision Diagrams (ADDs) to represent the utility function. When running the symbolic search algorithm, instead of terminating as soon as a search node with an empty task network is found, we evaluate all the search nodes we encounter during the search and determine the maximum utility of each of them using the underlying BDD representations. This evaluation takes is performed by intersecting all reachable final states will the utility function's ADD. The termination criterion is equivalent to the one we described for our explicit heuristic search approach, namely that the search terminates when the open list is empty (no new automaton can be generated), the cost bound is exceeded (all automata with costs up to the cost bound have been generated), or we encounter a state with the highest possible utility.

Theoretical Properties

We now consider the theoretical properties of the presented search approaches. Since they are all derived from optimal HTN planning with optimality guarantees, we only need to address the modifications we make for OSHTN planning.

It is important to point out that explicit search for optimal HTN planning is, in its base version, not a complete planning algorithm. This arises due to the fact that HTN problems can contain recursion in their methods, which can produce an infinitely large search space that can never be fully explored. This can be remedied by techniques like iterative deepening. For optimal planning, this simple solution is not applicable, as we need to guarantee that no cheaper solution than the one we found exists. The methods $A \mapsto Aa$ and $A \mapsto a$, where a is a primitive action with $c(a) = 0$ and A is an abstract task, will generate an infinite zero-cost loop within the search space, which would need to be fully explored to show that a given plan is an optimal solution. One method to avoid such infinite zero-cost loops is to restrict the maximum depth of decomposition allowed. Under this constraint, the search space is finite and exhaustive search will return an optimal solution (Sohrabi, Baier, and McIlraith 2009). Such a bound can be part of the problem, or it can be proven that the problem does not suffer such a problem, which is the case for a large set of planning problems, yielding optimality of exhaustive search for these domains (Nau et al. 2003). For the general case, we need to be able to infer such a bound, if it exists. For our purpose of OSHTN planning, we thus need to find a bound to the decomposition depth that if exhausted will guarantee that all possible goal states can be found within this bound. For total order HTN planning, the proof of Behnke, Höller, and Biundo (2018) is directly applicable and establishes an upper bound of $|\mathcal{A}| \cdot \left(2^{|\mathcal{V}|}\right)^2$. This bound is prohibitively large in practice, but applying it renders any explicit search for total order HTN planning to be complete. Note that for partial order HTN planning no such bound can exist, as deciding whether a solution exists is undecidable (Erol, Hendler, and Nau 1994).

To conclude optimality and completeness for the OSHTN explicit search, the main insight is that the underlying search algorithm eventually generates all search nodes with an empty task network with increasing cost. Further, any pruned task network can only lead to a plan that exceeds the cost limit as the heuristic we use for A* is admissible. Optimality stems from the fact that we maintain the state with the highest utility encountered during the search, so that eventually a solution is returned that leads to a state with the highest utility reachable while respecting the cost bound.

Proposition 1 *The presented explicit heuristic search is* optimal *and* complete *for OSHTN planning with bounded recursion depth. For total order OSHTN a depth limit exists that still guarantees completeness.*

Approach	Inter. Point	barman (290)	blocks (291)	robot (243)	rover (263)	satellite (159)	snake (109)	transport (600)	total coverage	norm. cov.
BDD	all	63	131	104	41	0	86	263	688	2.48
	0	10	20	26	8	0	15	60	139	4.17
	5	5	15	8	7	0	13	10	58	2.01
	9	4	7	7	6	0	12	3	39	1.57
LMC	all	53	156	103	43	16	90	270	731	2.68
	0	9	20	26	8	2	17	60	142	4.38
	5	4	18	7	7	1	14	13	64	2.15
	9	4	9	7	6	2	13	3	44	1.76
zero	all	36	153	99	41	0	81	0	410	1.96
	0	4	20	26	7	0	15	0	72	2.93
	5	4	17	7	7	0	12	0	47	1.75
	9	3	9	7	6	0	12	0	37	1.55

Table 1: Coverage of OSHTN planners on the seven benchmark domains. We show the overall coverage (row "all"), as well as the coverage for the interpolation points 0, 5, and 9, the overall coverage, and the coverage normalised to 1 per domain. For the three interpolation points coverage normalisation is w.r.t. the instances of that interpolation point.

In contrast to explicit search, the symbolic search technique for HTN planning (Behnke and Speck 2021) uses finite automata to represent sets of search nodes of explicit HTN search. The automata however only represents the task structure of explicit search nodes. To also represent state information, the transitions (s_1, a, s_2) in these automata are associated (or labelled) with BDDs that encode states associated with these transitions. Given that these automata can contain loops, they are able to represent both finite and infinitely large sets of search nodes reachable in explicit HTN search. Further, this representation is strong enough in the sense that it can represent any set of reachable search states under a given cost bound (Behnke and Speck 2021, Thm. 1). As a result, this search technique is complete, even for HTN domains with infinite zero-cost cycles.

Proposition 2 *The presented symbolic search is* optimal *and* complete *for OSHTN planning.*

Experimental Evaluation

In this section, we empirically evaluate and analyze the presented planning approaches for OSHTN planning on the newly introduced benchmark set.

We evaluated the symbolic planner, which we abbreviate as BDD and the explicit search planner using the LM-Cut heuristic (denoted LMC) and the zero heuristic (denoted zero). For each of the 1,955 OSHTN planning problems, each planner was given 8 GB of RAM and 30 minutes of runtime on a compute cluster with nodes equipped with Intel Xeon Gold 6242 32-core CPUs.

Results The overall coverage of the OSHTN planner on our benchmark set is shown in Tab. 1. We also present the results differentiated over the individual cost bounds. Since

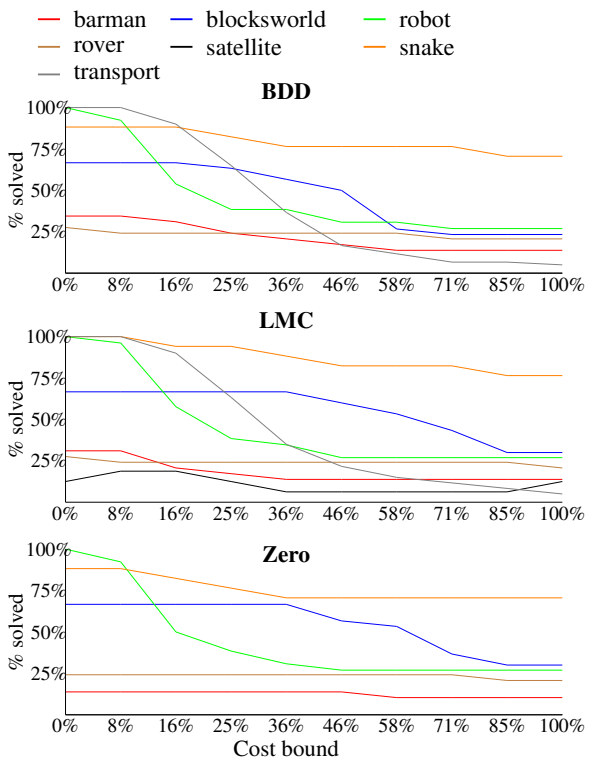

Figure 1: Dependence of coverage on the cost bound, split by planner and domain. Each colored line per domain shows for every interpolation point the coverage of the planner on that domain for that interpolation point.

we have generated for each of the original problem instances (initial state, initial plan, preferences, and utilities) up to 10 instances with a different cost bound, we can differentiate between these cost bounds for our analysis. For each original problem instance, we number the instances with different cost bound from 0 to 9 depending on which cost bound multiplier we used for it. We call these values the interpolation points. I.e. the instance 0 used the multiplier 0 and 9 used 1. As explained above, for some instances, we cannot generate all 10 intermediate cost bounds, if multiple of them would fall onto the same integer. Our benchmark set of 1,955 instances then only includes one of these instances. For analysing the behaviour of the planning algorithms as well as for analysing the properties of the benchmark set, it is more useful to consider these duplicate cost bounds as distinct instances. Else it could, e.g., be the case that there are x instances for interpolation point 4 which are all unsolved, but $2x$ for interpolation point 5, of which x are solved. This would imply a non-monotonicity in the behaviour of the algorithms, which does not exist, as the solvable instances are the same cost bound for interpolation points 4 and 5.

In Tab. 1, we display the coverage values for the interpolation points 0, 5, and 9. As expected, we can observe that the coverage decreases with an increasing cost bound, i.e., higher interpolation point. As the search space has to be explored until the bound is reached increases, the run-

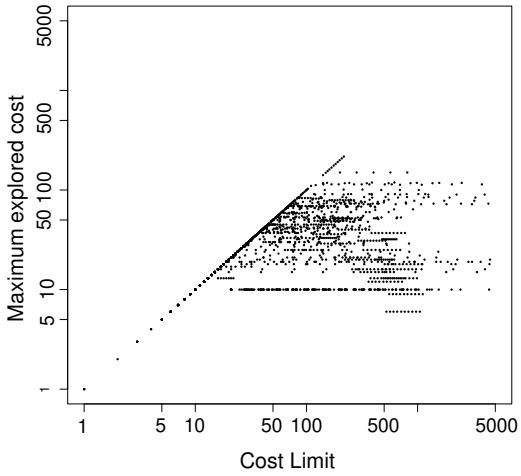

Figure 2: For every instance we show the cost limit against the maximally reached cost

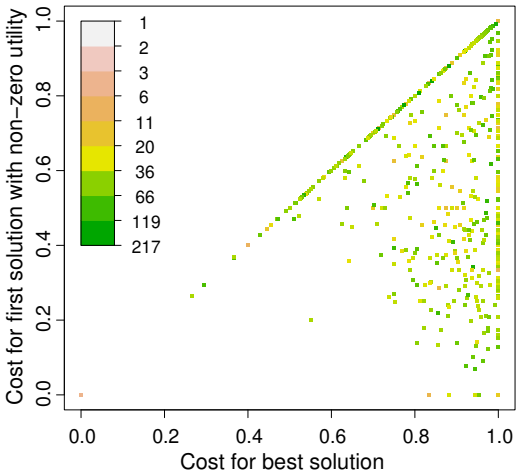

Figure 3: Relative cost needed to achieve the best achievable utility vs the relative cost needed to achieve any non-zero utility. We color-code the absolute value of the cost bound.

time should increase, too. For a more detailed understanding of this dependency, we depict in Fig. 1 for each planner, domain, and interpolation point the coverage of that planner. While coverage of the barman and rover domains is low across the board, the coverage in robot and transport, and to a lesser extent blocksworld, decreases as the cost bound increases. The exact point and shape of this increase is seemingly a domain property. In contrast, the instances in the snake domain don't show a significant decrease in performance. We attribute this to the overall low cost bounds in snake – the largest value for u_c for any snake instance is 23.

Overall, the explicit search with the LM-Cut heuristic performs best, but the symbolic approach is better in barman and robot. As expected, explicit search with LM-Cut dominates explicit search with the zero heuristic. Interestingly, both symbolic search and explicit search with the zero heuristic fail to solve the easiest satellite instances and the latter also fails to solve any transport instances.

Since the planning algorithms leave a lot of instances unsolved, we wanted to investigate how far these algorithms were from solving the respective instance. We have thus recorded for every instance, the maximum cost that was explored by each of the planners. For explicit search, this is the maximum cost of any search node popped from the search queue and for symbolic search the highest cost of a completed search layer. We plot in Fig. 2 the relation of the cost bound and the maximally explored cost for all planners. Note that points on the diagonal represent solved instances.

Lastly, we considered structural properties of the generated benchmark set – mainly to validate that the benchmark set we have generated is interesting. The most important property of an OSHTN benchmark set (and for any OSP benchmark set for that matter) is the distribution of achievable utilities with respect to costs. To allow for an inter instance comparison, we considered this for our analysis relative to the cost bound. We can now consider for every instance the (relative) cost needed to achieve the first non-zero utility (denoted as $r_{>0}$) and the (relative) cost of achieving the best possible utility r_b. If $r_b = 1$ then the cost bound needed to be fully utilised to obtain the maximum achievable utility. In many cases, however, it is possible to achieve the maximum achievable utility already with a lower cost. We depict this data in Fig. 3. Any point on the diagonal is a case where the cost of achieving any non-zero utility was no lower than the cost to achieve the maximum possible utility. Points on the right vertical line represent cases where the maximum achievable utility required the full cost bound. Overall, we can see that that the distribution of the utilities over the necessary cost for achieving it is quite varied, which indicated a well-balanced benchmark set.

Conclusions

In this paper, we have studied the problem of partial satisfaction planning for problems that have inherently hierarchical characteristics. On the theoretical side, we formally defined two hierarchical partial satisfaction planning formalisms, namely net-benefit planning and oversubscription planning, which differ in whether the utilities assigned to the objectives are comparable to the plan cost. We show that this class of problems belongs to the same complexity class as the underlying hierarchical task network planning problem, EXP-time complete. On the practical side, we have constructed a carefully designed and structured set of benchmarks for the hierarchical oversubscription planning. Moreover, we presented and evaluated two different approaches, namely explicit heuristic search and symbolic search, for hierarchical oversubscription planning using the proposed benchmarks. Our evaluation shows that the proposed approaches can solve about a third of the tasks in the proposed benchmark set. Our work contributes a balanced and well-structured benchmark set and strong baselines for future research in a novel and practical planning formalism.

One exciting direction for future work is extending our formalism and planners to support partially ordered hierarchical task network (Behnke, Höller, and Biundo 2019).

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. W. 2016. Bound to Plan: Exploiting Classical Heuristics via Automatic Translations of Tail-Recursive HTN Problems. In *Proc. ICAPS 2016*, 20–28.

Baier, J. A.; and McIlraith, S. A. 2008. Planning with Preferences. *AI Magazine*, 29(4): 25–36.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning through SAT. In *Proc. IJCAI 2018*, 6110–6118.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Bringing Order to Chaos - A Compact Representation of Partial Order in SAT-Based HTN Planning. In *Proc. AAAI 2019*, 7520–7529.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Finding Optimal Solutions in HTN Planning – A SAT-based Approach. In *Proc. IJCAI 2019*, 5500–5508.

Behnke, G.; and Speck, D. 2021. Symbolic Search for Optimal Total-Order HTN Planning. In *Proc. AAAI 2021*, 11744–11754.

Behnke, G.; Speck, D.; Katz, M.; and Sohrabi, S. 2023. Data and Code for "On Partial Satisfaction Planning with Total-Order HTNs". https://doi.org/10.5281/zenodo.7741799.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *AIJ*, 129(1): 5–33.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8): 677–691.

Domshlak, C.; and Mirkis, V. 2015. Deterministic Oversubscription Planning as Heuristic Search: Abstractions and Reformulations. *JAIR*, 52: 97–169.

Eifler, R.; Steinmetz, M.; Torralba, Á.; and Hoffmann, J. 2020. Plan-Space Explanation via Plan-Property Dependencies: Faster Algorithms & More Powerful Properties. In *Proc. IJCAI 2020*, 4091–4097.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In *Proc. AAAI 1994*, 1123–1128.

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

García-Olaya, A.; de la Rosa, T.; and Borrajo, D. 2021. Selecting goals in oversubscription planning using relaxed plans. *AIJ*, 291.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *AIJ*, 173(5–6): 619–668.

Gerevini, A. E.; and Long, D. 2005. Plan Constraints and Preferences in PDDL3. Technical Report R. T. 2005-08-47, University of Brescia, Department of Electronics for Automation.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS 2009*, 162–169.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14: 253–302.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proc. of ECAI 2014*, 447–452. IOS Press.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2019. HDDL - A Language to Describe Hierarchical Planning Problems. In *Second ICAPS Workshop on Hierarchical Planning*.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018. A Generic Method to Guide HTN Progression Search with Classical Heuristics. In *Proc. ICAPS 2018*, 114–122.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020. HTN Planning as Heuristic Progression Search. *JAIR*, 67: 835–880.

Katz, M.; and Keyder, E. 2022. A* Search and Bound-Sensitive Heuristics for Oversubscription Planning. In *Proc. AAAI 2022*.

Katz, M.; Keyder, E.; Pommerening, F.; and Winterer, D. 2019a. Oversubscription Planning as Classical Planning with Multiple Cost Functions. In *Proc. ICAPS 2019*, 237–245.

Katz, M.; Keyder, E.; Pommerening, F.; and Winterer, D. 2019b. PDDL benchmarks for oversubscription planning. https://doi.org/10.5281/zenodo.2576024.

Katz, M.; and Mirkis, V. 2016. In Search of Tractability for Partial Satisfaction Planning. In *Proc. IJCAI 2016*, 3154–3160.

Keyder, E.; and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *JAIR*, 36: 547–556.

Mirkis, V.; and Domshlak, C. 2013. Abstractions for Oversubscription Planning. In *Proc. ICAPS 2013*, 153–161.

Mirkis, V.; and Domshlak, C. 2014. Landmarks in Oversubscription Planning. In *Proc. ECAI 2014*, 633–638.

Muller, D.; and Karpas, E. 2018. Value Driven Landmarks for Oversubscription Planning. In *Proc. ICAPS 2018*, 171–179.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *JAIR*, 20: 379–404.

Sinz, C. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Proc. CP 2005*, 827–831.

Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In *Proc. ICAPS 2004*, 393–401.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. HTN Planning with Preferences. In *Proc. IJCAI 2009*, 1790–1797.

Speck, D.; and Katz, M. 2021. Symbolic Search for Oversubscription Planning. In *Proc. AAAI 2021*, 11972–11980.

van den Briel, M.; Sanchez, R.; Do, M. B.; and Kambhampati, S. 2004. Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. In *Proc. AAAI 2004*, 562–569.

Xie, F.; Müller, M.; and Holte, R. 2014. Jasper: the art of exploration in Greedy Best First Search. In *IPC-8 Planner Abstracts*, 39–42.