Towards Hierarchical Task Network Planning as Constraint Satisfaction Problem

Tobias Schwartz, Michael Sioutis, Diedrich Wolter

University of Bamberg
An der Weberei 5
Bamberg
{tobias.schwartz, michail.sioutis, diedrich.wolter}@uni-bamberg.de

Abstract

In recent years, propositional logic encodings for HTN planning have seen many improvements and resulted in competitive planners. Modeling all kinds of features and constraints imposed by the task hierarchy, however, is very challenging in propositional logic, and has recently led to including preprocessing steps before creating the SAT formulas. Instead of using propositional logic, classical planning problems have previously been encoded as constraint satisfaction problems (CSPs), which are more expressive. Indeed, CSPs allow a more natural and convenient way of representing all constraints of the task hierarchy, yet only little work on using constraint solving methods in HTN planning exist. Hence, in this paper, we outline first ideas for encoding an HTN planning problem into a single CSP. Our motivation lies in obtaining constraint networks for HTN planning that we can solve with state-of-the-art solvers.

Introduction

Many real-world tasks deal with some form of constraints. As such, constraint satisfaction techniques have proven successful as an underlying framework for solving problems of various domains. Constraints, in this context, can express a large bandwidth of problem features, from simple ordering relations to modeling complex numerical resource allocations. Despite its success in many fields, constraint satisfaction methods have been applied only sparsely in the field of AI planning, especially in hierarchical task network (HTN) planning. Instead, recently, HTN planning problems have often been encoded as a sequence of propositional satisfiability (SAT) problems (Behnke, Höller, and Biundo 2018, 2019a; Schreiber et al. 2019; Schreiber 2021). Since HTN planning is generally undecidable, one cannot simply compile the problem into propositional logic, but rather fix the number of possible actions and then employ SAT techniques to verify whether there exists a plan of a particular length. This process may be repeated for a series of compilations of increasing length (Bercher, Alford, and Höller 2019).

Compared to a SAT compilation, formulating a planning problem as a constraint satisfaction problem (CSP) can be characterized as more natural or convenient (Nareyek et al. 2005). This is mostly attributed to the fact that CSPs support higher-level constraints that are able to directly represent domain-specific knowledge. In SAT, for example, quantitative information in the form of discrete numbers may be captured in a propositional formula by using a variable for each value of the discrete domain, but doing so will result in loosing the natural ordering information of the numbers.

As noted by Barták and Toropila (2008), early constraint models for planning had their origins in SAT and thus were only using Boolean variables and constraints in form of logical formulas. Then, they proposed multiple constraint models for classical planning that rely on more sophisticated constraint structures, clearly improving on their logical counterparts in terms of both stronger constraint propagation and faster runtime.

In contrast to classical planning, HTN planning imposes additional structural constraints of task hierarchies that must be satisfied. Those additional constraints increasingly motivate the use of constraint programming. Unfortunately, early reports of ongoing work in this direction (Surynek and Barták 2005) have apparently not been followed through. Stock et al. (2015) provide the only successful application of CSPs for solving HTN planning problems, known to us. They, however, rely on a more complex architecture with multiple constraint networks used independently of one another, each handling a different form of knowledge (e.g., causal, temporal, spatial), called meta-CSP (Mansouri and Pecora 2016).

Instead, in this paper, we outline first ideas for encoding HTN planning as one single CSP. Our motivation lies in obtaining constraint networks for HTN planning that we can solve with state-of-the-art off-the-shelf solvers, and not relying on tailored solution to this problem.

Preliminaries

We start by briefly presenting some frameworks that are relevant to our work and we will be referring to in this paper.

Qualitative Constraint Satisfaction Problems

In the following, we focus on qualitative constraint satisfaction problems (QCSPs), which are typically used to represent and reason about qualitative temporal (or spatial) information. They are defined analogously to classical CSPs (Russell and Norvig 2020), but allow variables to be of infinite domains. QCSPs are often tackled via the use of a qualitative constraint graph, called *Qualitative Constraint Network* (QCN), which is defined as follows.

Definition 1 (QCN). *A* QCN *is a tuple* (V, C) *where:*

- $V = \{v_1, \ldots, v_n\}$ *is a non-empty finite set of variables, each representing an entity of an infinite domain* D*;*
- *and* C *is a mapping* $C : V \times V \rightarrow 2^{\mathsf{B}}$ *such that* $C(v, v) = \{\mathsf{Id}\}$ *for all* $v \in V$ *and* $C(v, v') = C(v', v)^{-1}$ *for all* $v, v' \in V$*.*

Let $\mathcal{N} = (V, C)$ be a QCN, then a *solution* of $\mathcal{N}$ is a mapping $\sigma : V \rightarrow \mathsf{D}$ such that $\forall v, v' \in V$, $\exists b \in C(v, v')$ such that $(\sigma(v), \sigma(v')) \in b$, and $\mathcal{N}$ is *satisfiable* (or consistent) iff it admits a solution (Ligozat 2013; Dylla et al. 2017).

We assume our constraint language to be defined like the well-known Interval Algebra (Allen 1983), which is a first-order theory for representing and reasoning about temporal information. For now, we make use of its equality (eq), inequality ($neq = \{\mathsf{B} \setminus eq\}$), and ordering constraints ($\{<, >\}$).

Hierarchical Task Network Planning

Hierarchical planning extends classical planning by introducing a task hierarchy. Instead of only using the notion of applicable actions, it essentially differentiates between primitive and compound tasks. Primitive tasks are hereby comparable to the actions in classical planning. Compound tasks describe a more abstract notion of a set of actions. This grouping can impose additional restrictions that might not be easily achievable using only preconditions and effects of actions. For example, an imposed ordering constraint can be easily encoded in a compound task and drastically improve efficiency of the planner. In fact, ordering tasks according to a partial order can be seen as the motivation behind hierarchical task network (HTN) planning, perhaps the most basic hierarchical formalism (Bercher, Alford, and Höller 2019). In what follows, we briefly recall the definitions for lifted HTN planning as recently defined in the hierarchical domain definition language (HDDL) (Höller et al. 2020).

The basis for HTN planning is the so-called *task network*, which essentially imposes a strict partial order on a finite set of tasks. A set of variable constraints may constrain certain task parameters to be (non-)equal to other task parameters or constants, or to (not) be of a certain type. A task network is called *ground* if all parameters are bound to constants.

An HTN planning domain D defines the sets of all primitive tasks T_P, compound tasks T_C, and decomposition methods M. A method $m \in M$ is a triple (c, tn, VC) of a compound task name $c \in T_C$, a task network $tn \in T_P \cup T_C$ and some variable constraints VC over the parameters of c and tn. An HTN planning problem P is a tuple (D, s_I, tn_I, g), where D is the planning domain, $s_I \in S$ is the initial state, tn_I is the initial task network, and g optionally defines a goal description.

Although a goal description can be defined, the objective in HTN planning is not to achieve a certain state-based goal. Instead, a *solution* to a given HTN planning problem is a final task network tn_S which is reachable from s_I by only applying methods and compound tasks. In the process, all compound tasks need to be decomposed into primitive actions, such that tn_S does not contain compound tasks anymore. The enforced task hierarchy directly restricts the set of possible solutions to only those that can be obtained by task decomposition (Bercher, Alford, and Höller 2019).

Listing 1: Action `drive` in HDDL

```
(:action drive
  :parameters (?l1 ?l2 - location)
  :precondition (and
    (tAt ?l1)
    (road ?l1 ?l2))
  :effect (and
    (not (tAt ?l1))
    (tAt ?l2)))
...)
```

HTN Planning Constraint Model

Similar to Barták and Toropila (2008), we employ a multi-valued representation of the planning problem. That is, instead of grounding every single fact using enumeration, we create *state variables* for different fragments of the world state, where the domains of values represent exclusive options. For example, given a service robot, the robot may only be at one particular location at any given time. Instead of now generating all combined facts of a robot being at a particular location, all potential locations represent the domain for the state variable of the robots location. Using such a multi-valued representation instead of a purely propositional, fact-based encoding, the number of variables decreases, whereas the size of the domains increases. This is generally recommended for constraint modeling, as opposed to the other way around (many variables of small domains) (Barták and Toropila 2008).

We follow the constraint model proposed by Ghallab, Nau, and Traverso (2004), coined the *straightforward model* by Barták and Toropila (2008). For now, we only consider HTN features defined by the *Hierarchical Domain Definition Language* (HDDL) (Höller et al. 2020).

A CSP denoting the problem of finding a plan of length n, consists of $n+1$ incrementally changing constraint networks $\mathcal{N}$, where the k^{th} network $\mathcal{N}_k^i$ represents the state s_k^i after performing $k-1$ planning operations and i incremental task decompositions. Since the plan length n, i.e., the sequence of primitive actions in the final plan, is generally unknown in advance, we dynamically grow the list of constraint networks $\mathcal{N}$ whenever we perform a primitive action in state s_k^i which results in a new successor state s_{k+1}^i. Describing states as constraint networks $\mathcal{N}$, is a variation in presentation from Barták and Toropila (2008), as they describe states as sets of v multi-valued variables.

In the following, we elaborate on the constraint network design and point out how the features of HDDL can be expressed within this notation.

Variable representation: We model the variables in a special constraint network, which ensures a mapping of each state variable to exactly one object of the problem domain. To illustrate this, consider the action `drive` from the transport domain presented by Höller et al. (2020), given in Listing 1. The corresponding problem file further specifies the

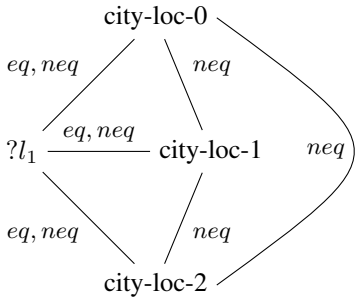

Figure 1: Grounding of location $?l_1$ as qualitative constraint network $\mathcal{N}$.

relevant location objects as city-loc-0, city-loc-1, or city-loc-2. We can formulate a mapping from the location $?l_1$, given as parameter, to a particular object specified by the problem, as a constraint network (Figure 1), where $?l_1$ is either equal (eq) or not equal (neq) to any of the objects. By preferring eq over neq in the search, and ensuring that all objects are different (i.e., neq), we guarantee a mapping to at most one of the objects. Indeed, once we commit to any one of the equality relations, constraint propagation will force all other relations to become neq, due to the neq constraint between all of the objects. We can easily extend this formulation with other variables from the same domain (such as $?l_2$), by adding them as new nodes to the network and creating the same eq, neq constraints to all of the objects. Note that all such variables by default are independent in terms of constraint propagation and thus mapping one variable to an object does not influence the other variable mappings. Obviously, we can change this behavior by adding additional constraints (such as neq) between variables if desired. Furthermore, note that by default this constraint network approach postpones all variable mappings without explicit constraints until eventually the CSP solver is called. For example, imagine that for action `drive` we could have multiple vehicles available at $?l_1$ of which any particular one could be used to drive to $?l_2$. If no direct constraints are imposed, initially we only set the eq, neq constraints which postpone this decision to the latest point in time.

Action representation: Action application works just like in classical planning, where for a given state s_k^i action variable $A^{s_k^i}$ acts as a logical constraint, leading from one constraint network $\mathcal{N}_k^i$ to the next $\mathcal{N}_{k+1}^i$. Clearly, all required preconditions of the action need to be satisfied in $\mathcal{N}_k^i$, then after its execution all effects of the action hold in $\mathcal{N}_{k+1}^i$. We follow the formulation by Barták and Toropila (2008) and model this relation using logical implications, i.e. for any action variable $A^{s_k^i}$ in state s_k^i we have,

$$A^{s_k^i} = a \rightarrow \textit{Pre(a)}^{s_k^i}, \forall a \in Dom(A^{s_k^i}),$$

$$A^{s_k^i} = a \rightarrow \textit{Eff(a)}^{s_{k+1}^i}, \forall a \in Dom(A^{s_k^i})$$

where $\textit{Pre(a)}^{s_k^i}$ is a conjunction of equalities changing the required relations within the constraint network to reflect the preconditions of action a in state s_k^i. Similarly, $\textit{Eff(a)}^{s_{k+1}^i}$ expresses the effects of a in the successor state s_{k+1}^i. Barták and Toropila (2008) additionally introduce constraints for the *frame axioms*, i.e. constraints to ensure that any state variable unaffected by the action remains unchanged. In our representation this naturally holds as we consider state s_{k+1}^i as a copy of its predecessor s_k^i and only change the variables affected by the action. A final refinement, e.g., variable binding $?l_1 \xrightarrow{eq}$ city-loc-0, which is not directly affected by the action, can later not be changed through constraint propagation but would only lead to inconsistency.

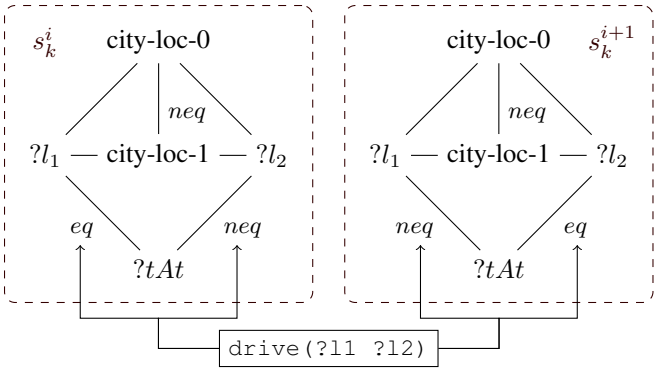

Figure 2: Applying action `drive` (Listing 1) in state s_k^i (left) to derive at s_{k+1}^i (right).

Note that action $a \in Dom(A^{s_k^i})$ may only be one of many choices possible in the given state s_k^i. However, the effect of the action and hence the successor state s_{k+1}^i depends on which action was chosen. To this end, we employ the same implication structure as outlined above, requiring the constraint solver to handle such instances. Figure 2 illustrates this notion on a simple example of applying action `drive`. Formulating this implication directly within the realms of the constraint language can eliminate the need for extending the state-of-the-art solvers that we have today for solving qualitative constraint networks and is part of future work.

Abstract task representation: In HDDL, abstract tasks are defined explicitly in the domain. They represent a form of abstraction from the specific method used to fulfill a certain task, already defining the parameters and their respective input types. We can use this information to establish the same eq, neq constraints to all variables of the domain indicated by the type. We hereby convey the information that each parameter should be linked to exactly one variable of its domain, without making this link explicit yet.

Method representation: As described above, methods are always linked to an abstract task. However, they may define further parameters beyond the ones already stated in the abstract task definition.

Generally, methods describe a fixed number of subtasks that have to be fulfilled in order to complete the task. Some HTN planning systems, in particular those employing SAT compilation techniques (e.g., Behnke, Höller, and Biundo (2018); Schreiber et al. (2019); Schreiber (2021)), rely on

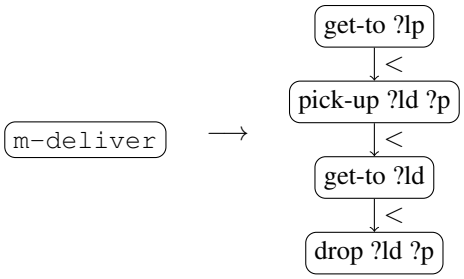

Figure 3: Applying method `m-deliver` to decompose task `deliver` in state s_k^i (left) to derive at s_k^{i+1} (right). For full method and task descriptions, see Höller et al. (2020).

a totally ordered set of those subtasks. By contrast, HDDL also supports partially ordered subtasks. Conveying the relative ordering of subtasks has been described as the main difficulty for a SAT encoding (Behnke, Höller, and Biundo 2019a). Where Behnke, Höller, and Biundo (2019a) rely on a preprocessing step, conducting reasoning on the order before creating the actual SAT formula, CSPs naturally allow the representation of ordering constraints.

Where an action progresses the state one step from k to $k+1$ within a single layer of the decomposition tree, a task decomposition introduces an incremental change along the hierarchy of the decomposition tree, moving from layer i to $i+1$. Given a task variable $T^{s_k^i}$ in state s_k^i, we then have,

$$T^{s_k^i} = m \rightarrow \textit{Dec(m)}^{s_k^{i+1}}, \forall m \in Dom(T^{s_k^i})$$

where *Dec(m)* is the decomposition effect of the method m, adding all subtasks to the constraint network $\mathcal{N}_k^{i+1}$. Any ordering relation between those subtasks can simply be established by introducing ordering constraints in the network.

Similarly to the incremental SAT encoding by Schreiber et al. (2019), we must ensure that for finding a plan of length n, the final increment of all constraint networks does not contain any task or method variables.

Discussion and Related Work

Mali and Kambhampati (1998) were the first to propose a propositional logic encoding for HTN planning problems. However, these encodings were restricted to non-recursive domains. Only in recent years, development of new SAT encodings overcame this restriction and resulted in competitive performance. Initially, these encodings were only applicable to the subset of totally-ordered HTN planning problems (Behnke, Höller, and Biundo 2018; Schreiber et al. 2019). By now, SAT encodings have been extended to partially-ordered problems and used to find optimal plans with respect to the plan length (Behnke, Höller, and Biundo 2019a,b).

As also summarized by Schreiber (2021), all these SAT encodings operate similarly. Their encodings are iteratively extended along the depth of the hierarchy, instead of the length of a final plan (as done in classical planning). Additionally, they all require a prior grounding procedure. While it has been shown that grounding often improves subsequent search algorithms, on some planning domains, grounding suffers from intrinsic scaling problems. Thus, Schreiber (2021) developed a lifted SAT planner that omits grounding, but instead is limited to totally-ordered problems.

A CSP encoding, as discussed in this paper, by design avoids the need for grounding. Furthermore, additional constraints can essentially be modeled for free, which make this approach quite comfortable in dealing with partially-ordered HTN planning problems.

The advantages of a CSP encoding have previously been explored by Stock et al. (2015). But in contrast to our work, they rely on a sophisticated reasoning framework, called meta-CSP (Mansouri and Pecora 2016), as underlying architecture. This framework has the advantage that it allows reasoning with knowledge of different forms (such as temporal, spatial, causal, resources). It comes, however, with the drawback that finding consistent solutions is generally slow, since each form of knowledge is dealt with in a separate constraint network and finding a consistent solution requires all networks to be consistent at once. This requires special solvers that allow interaction among each other. Instead, we are motivated to find an encoding which uses only one single constraint network architecture, such that we can employ state-of-the-art solvers that we have today for solving qualitative constraint networks. This may be seen as an approach in between a pure propositional logic encoding on one side and a quite complex constraint-based encoding on the other side. We argue that this allows us to combine the advantages of both worlds.

For now, a few challenges remain that may impact the success of the proposed CSP encoding. First, we assume that dynamically creating new constraint networks both within one hierarchy for primitive action application and following the hierarchy for compound task decomposition is feasible without computational blow-up. We here expect that state-of-the-art qualitative constraint solvers can help to drastically reduce the state space as inconsistent configurations can be pruned early, as done traditionally in CSP search. Second, action variables are mapped to their respective domain indicated by the variable type. We currently do not consider the case where the type of a variable may change or is unavailable. Without type information a mapping to all ground objects can be done, albeit inefficiently. While a variable is not yet ground, changing its type is simple, as this just changes the possible mappings and consequently disallows all others (by use of neq constraints).

Conclusion and Future Work

In this paper, we have outlined first steps towards constructing a constraint satisfaction problem (CSP) encoding for HTN planning problems. CSPs have been used in classical planning before and allow for a more natural representation compared to an encoding in propositional logic (Nareyek et al. 2005). Using such SAT encodings has recently led to very successful results in HTN planning. Especially the constraints imposed by the task hierarchy present in HTN planning problems motivate the use of more sophisticated constraint satisfaction techniques. To our surprise, only little work has been conducted in this direction. In fact, Stock

et al. (2015) provide the only approach we are aware of, using CSPs in the context of HTN planning. Their approach is based on modeling the planning problem in a more complex reasoning framework, called meta-CSP (Mansouri and Pecora 2016), which allows them to represent different forms of information in separate CSPs independently.

We avoid the overhead of combining multiple CSPs by aiming to encode the HTN planning problem directly into one single qualitative constraint network architecture. Our encoding draws inspiration from recent SAT encodings for expressing the task hierarchy in an incremental fashion and restricting the depth of the decomposition tree instead of the length of the plan (Schreiber 2021). Action encodings follow the structure proposed previously for classical planning (Barták and Toropila 2008), using an implication constraint. We additionally introduce a novel binding mechanism, based on preferring equality-relations over all others. State-of-the-art solvers that we have today for solving qualitative constraint networks can be extended to handle those implication constraints and follow the required preference when solving the constraint problems.

Future work will be further refining our encoding, such that state-of-the-art solvers can be applied directly without the need of adaptations. We believe that the Interval Algebra (Allen 1983) already allows us to model several challenges of encoding HTN planning as CSP, as the relations defined within this qualitative constraint language, such as *during* or *meets*, intuitively describe properties present in HTN planning. Finally, we are interested in actually implementing our encoding, verifying its correctness and comparing its performance with current state-of-the-art HTN planners.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful feedback. This research is partially supported by BMBF AI lab dependable intelligent systems.

References

Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 26(11): 832–843.

Barták, R.; and Toropila, D. 2008. Reformulating Constraint Models for Classical Planning. In *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference*, 525–530. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT - Totally-Ordered Hierarchical Planning Through SAT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 6110–6118. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2019a. Bringing Order to Chaos – A Compact Representation of Partial Order in SAT-Based HTN Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7520–7529. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2019b. Finding Optimal Solutions in HTN Planning - A SAT-based Approach. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 5500–5508.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning - One Abstract Idea, Many Concrete Realizations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 6267–6275.

Dylla, F.; Lee, J. H.; Mossakowski, T.; Schneider, T.; van Delden, A.; van de Ven, J.; and Wolter, D. 2017. A Survey of Qualitative Spatial and Temporal Calculi: Algebraic and Computational Properties. *ACM Comput. Surv.*, 50(1): 7:1–7:39.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning*. Elsevier Science & Technology. ISBN 1558608567.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9883–9891. AAAI Press.

Ligozat, G. 2013. *Qualitative Spatial and Temporal Reasoning*. ISTE Ltd and John Wiley & Sons, Inc. ISBN 978-1-84821-252-7.

Mali, A. D.; and Kambhampati, S. 1998. Encoding HTN Planning in Propositional Logic. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 190–198. AAAI.

Mansouri, M.; and Pecora, F. 2016. A robot sets a table: a case for hybrid reasoning with different types of knowledge. *J. Exp. Theor. Artif. Intell.*, 28(5): 801–821.

Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H. A.; Rintanen, J.; and Tate, A. 2005. Constraints and AI Planning. *IEEE Intell. Syst.*, 20(2): 62–72.

Russell, S. J.; and Norvig, P. 2020. *Artificial Intelligence - A Modern Approach, Fourth Edition*. Pearson Education. ISBN 78-0-13-461099-3.

Schreiber, D. 2021. Lilotane: A Lifted SAT-based Approach to Hierarchical Planning. *Journal of Artificial Intelligence Research*, 70: 1117–1181.

Schreiber, D.; Pellier, D.; Fiorino, H.; and Balyo, T. 2019. Tree-REX: SAT-Based Tree Exploration for Efficient and High-Quality HTN Planning. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, 382–390. AAAI Press.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 6459–6464. IEEE.

Surynek, P.; and Barták, R. 2005. Encoding HTN Planning as a Dynamic CSP. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *LNCS*, 868. Springer.