HTN Plan Verification by Qualitative Temporal Reasoning

Tobias Schwartz, Diedrich Wolter

University of Lübeck, Germany
{tobias.schwartz, diedrich.wolter}@uni-luebeck.de

Abstract

Plan verification is the task of checking whether a proposed plan correctly solves a given planning problem. In Hierarchical Task Network (HTN) planning, this verification problem is known to be NP-hard. Existing approaches to HTN plan verification range from SAT encodings to parser-based techniques. However, existing methods do not explicitly exploit the temporal structure inherent in hierarchical decomposition. In this paper, we establish a formal connection between HTN planning and temporal reasoning by showing how decomposition structures can be naturally represented using qualitative constraint networks. Building on this insight, we present a new top-down encoding that transforms the verification of partially ordered task networks into a temporal reasoning problem. We prove the correctness of this encoding and explain how it accounts for both the hierarchical and temporal aspects of HTN plans. By linking HTN plan verification with qualitative temporal reasoning, our approach introduces a principled formal framework for reasoning about complex temporal relationships in hierarchical plans. This connection offers new perspectives for knowledge representation in structured planning domains.

Introduction

Verifying the correctness of plans produced by automated planning systems is crucial for ensuring reliable autonomous behavior in complex domains. In *Hierarchical Task Network (HTN) planning*, a formalism known for capturing structured domain knowledge through task decomposition hierarchies, verification becomes especially challenging. While verifying solutions in classical planning can be done efficiently by checking preconditions and effects along an action sequence, HTN plan verification must additionally confirm that the given actions can be derived through valid hierarchical decompositions and ordering refinements, making it NP-complete in the grounded setting (Behnke, Höller, and Biundo 2015).

Recent methods addressing HTN plan verification have predominantly relied on syntactic encodings, including SAT-based methods (Behnke, Höller, and Biundo 2017; Lin, Behnke, and Bercher 2023), compilations to HTN planning problems (Höller et al. 2022), and parsing-based techniques (Barták et al. 2020, 2021; Ondrčková et al. 2023; Lin et al. 2023; Pantucková and Barták 2023; Pantucková, Ondrcková, and Barták 2024). These approaches typically focus on verifying a given, fully specified action sequence, motivated by concrete observable execution scenarios. However, this focus limits verification to single, totally ordered plans and does not directly support verifying partially ordered task networks, which inherently offer greater flexibility and capture richer structural relationships. For example, overlapping or parallel execution of actions has been out of scope so far.

In this paper, we present a fundamentally different approach that directly verifies *partially ordered task networks* by leveraging their inherent temporal structure. Our key insight is that HTN decomposition naturally induces qualitative temporal relationships: when a compound task is decomposed into subtasks, the parent task's execution temporally contains its children; and explicit task ordering requirements translate to temporal precedence. By making these temporal relationships explicit, we can transform the verification problem into a temporal reasoning problem.

Specifically, we encode HTN plan verification as a satisfiability problem in *Qualitative Constraint Networks* (QCNs), a widely studied framework for temporal reasoning (Sioutis and Wolter 2021), using the INDU calculus (Pujari, Kumari, and Sattar 1999) that combines Allen's interval relations with duration information. Our encoding employs a three-step transformation: (1) we ensure each task occurrence in the solution has a unique identifier; (2) we transform the domain such that each compound task has at most two decomposition methods; and (3) we map tasks to temporal intervals and map decomposition structures to temporal constraints, using the INDU calculus to ensure subtask durations exactly partition their parent task's duration.

We show that our encoding is sound and complete, and the resulting QCN is polynomial in the size of the input. Additionally, we demonstrate how our approach generalizes existing verification methods by enabling reasoning over partial orders, thus offering new possibilities for verification beyond single action sequences. By establishing this explicit link between HTN planning and qualitative temporal reasoning, our work opens new avenues for linking the fields together. Potential impacts include the ability to introduce more temporal aspects into HTN planning as well as enriching verification methodologies.

Preliminaries

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

Qualitative Constraint-Based Reasoning

A *binary* qualitative constraint language is based on a finite set B of *jointly exhaustive and pairwise disjoint* (JEPD) relations, called the set of *base relations* (*atoms*), that is defined over an infinite domain D (Ligozat 2011). These base relations represent definite knowledge between two entities of D with respect to the level of granularity provided by the domain D. Indefinite knowledge can be specified by a union of possible base relations, and is represented by the set containing them. The set B contains the identity relation Id, and is closed under the *converse* operation ($^{-1}$). The entire set of relations 2^{B} is equipped with the set-theoretic operations of union, intersection, and the converse operation. Our encoding is based on the INDU calculus (INterval and DUration network) (Pujari, Kumari, and Sattar 1999), which extends Allen's Interval Algebra (Allen 1983) by incorporating duration information. While Allen's calculus specifies 13 base relations that can hold between any two intervals based solely on their temporal arrangement, INDU enriches this representation with relative duration information, resulting in 25 base relations (Pujari, Kumari, and Sattar 1999; Balbiani, Condotta, and Ligozat 2006).

In INDU, the domain D consists of time intervals specified by their start and end points (x^-, x^+). Each base relation in INDU combines an Allen relation with duration information, distinguishing three cases: the first interval is shorter than ($<$), equal to ($=$), or longer than ($>$) the second interval. For instance, Allen's *before* relation becomes three INDU relations: *before*$^<$ (first interval is before and shorter), *before*$^=$ (before with equal duration), and *before*$^>$ (before and longer). The *equals* relation from Allen's calculus corresponds to a single INDU relation where intervals are identical in both position and duration. For reference, Allen's Interval Algebra defines 13 base relations between temporal intervals, such as *before*, *meets*, *overlaps*, *starts*, *finishes*, *during*, and *equals*, along with their converses.

Temporal entities and their relations may be represented and reasoned with in a qualitative constraint network (QCN), defined as follows (Sioutis and Wolter 2021).

Definition 1 (Qualitative Constraint Network)**.** *A qualitative constraint network is a tuple $\mathcal{N} = (V, \Gamma)$, where:*

- *V is a set of variables, each representing an entity from an infinite domain D;*
- *Γ is a mapping $V \times V \to 2^{\mathsf{B}}$, representing all possible constraints between two entities, such that for all $v, v' \in V$, $\Gamma(v, v) = \{\mathsf{Id}\}$ and $\Gamma(v, v') = (\Gamma(v', v))^{-1}$.*

Here, Id denotes the identity relation, which in INDU is the *equals*$^=$ relation. Let $\mathcal{N} = (V, \Gamma)$ be a QCN, then a *solution* of $\mathcal{N}$ is a mapping $\psi\colon V \to \mathsf{D}$ that assigns each variable a specific value from the domain. For any pair of variables $v, v' \in V$, their relationship must satisfy at least one base relation from their constraint set – that is, there must exist some $b \in \Gamma(v, v')$ such that $(\psi(v), \psi(v')) \in b$. While $\Gamma(v, v')$ may contain multiple possible base relations, a solution requires selecting exactly one that holds between the assigned values (as relations in B are JEPD).

For example, if $\Gamma(v, v') = \{\textit{before}^<, \textit{meets}^=\}$, a solution might map v to interval $[1, 2]$ and v' to $[2, 3]$, satisfying the *meets*$^=$ relation (they meet and have equal duration). The *before*$^<$ relation, while permitted by $\Gamma(v, v')$, would not hold for these specific assignments.

A QCN is *satisfiable* (or consistent) if and only if it admits such a solution. The satisfiability problem, i.e., determining if there exists an interpretation satisfying these constraints, is NP-complete for most widely-adopted qualitative calculi (Dylla et al. 2017), including INDU (Balbiani, Condotta, and Ligozat 2006).

Hierarchical Task Network Planning

This section introduces the HTN planning formalism used in this paper, following the notation and concepts from Geier and Bercher (2011). HTN planning extends classical planning by organizing tasks hierarchically. An HTN domain consists of a finite set of facts F describing the world, primitive tasks A representing directly executable actions, compound tasks C representing abstract activities that must be refined, and decomposition methods M that specify how compound tasks can be broken down into subtasks. The planning process starts from an initial compound task and recursively decomposes it until only executable actions remain. A state $s \subseteq F$ denotes which facts hold true. Each action $a \in A$ has preconditions $\text{pre}(a) \subseteq F$ and is *applicable* in state s if $\text{pre}(a) \subseteq s$. Executing an applicable action transforms the state. A sequence of actions $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ is *executable* in state s_0 if each a_i is applicable in its respective state. We write $s \to_\pi^\star s'$ to denote that π is executable in s and results in s'.

A *task network* represents a set of tasks together with their ordering requirements.

Definition 2 (Task Network)**.** *A task network is a tuple $\text{tn} = (T, \prec, \alpha)$, where:*

- *T is a finite, non-empty set of task identifiers;*
- *$\prec \subseteq T \times T$ is a strict partial order defining ordering constraints among the tasks;*
- *$\alpha\colon T \to A \cup C$ a labelling that maps each task identifier to either a primitive or compound task.*

Two task networks $\text{tn} = (T, \prec, \alpha)$ and $\text{tn}' = (T', \prec', \alpha')$ are *isomorphic*, written $\text{tn} \cong \text{tn}'$, if there exists a bijection $\varphi\colon T \to T'$ such that for all $t \in T$, $\alpha(t) = \alpha'(\varphi(t))$, and for all $t_1, t_2 \in T$, $(t_1, t_2) \in \prec$ if and only if $(\varphi(t_1), \varphi(t_2)) \in \prec'$. We say tn is an *ordering refinement* of tn', written $\text{tn} \sqsubseteq \text{tn}'$, if $T = T'$, $\alpha = \alpha'$, and $\prec' \subseteq \prec$ (i.e., tn imposes all ordering constraints from tn' and possibly additional ones).

For convenience, we adopt the following restriction operation. Let D and V be two arbitrary sets, $R \subseteq D \times D$ a relation, $f\colon D \to V$ a function, and $\text{tn} = (T, \prec, \alpha)$ a task network. The restrictions of R, f, and tn to some set $X \subseteq D$ are defined by $R|_X := R \cap (X \times X)$; $f|_X := f \cap (X \times V)$; and $\text{tn}|_X = (T \cap X, \prec|_X, \alpha|_X)$, where $\prec|_X$ and $\alpha|_X$ denote the usual restrictions of the relation and function to X.

We can now formally define the HTN planning domain.

Definition 3 (HTN Planning Domain)**.** *An HTN planning domain is a tuple $\mathcal{D} = (F, A, C, M)$ where:*

- *F is a finite set of propositional facts;*
- *A is a finite set of primitive tasks (actions);*
- *C is a finite set of compound tasks;*
- *M is a finite set of decomposition methods, where each method $m = (c, tn_m)$ pairs a compound task $c \in C$ with a non-empty task network tn_m.*

A decomposition method $m = (c, \text{tn}_m)$ describes one way to decompose a compound task c into a non-empty task network $\text{tn}_m = (T_m, \prec_m, \alpha_m)$. This decomposition replaces c with its refinement and preserves all ordering constraints. This idea can naturally be extended to the decomposition of a task network.

Definition 4 (Decomposition)**.** *Let $tn = (T, \prec, \alpha)$ be a task network and $m = (c, tn_m) \in M$ a decomposition method. The method m decomposes the task network tn for a task identifier $t \in T$ with $\alpha(t) = c$ into a task network tn′, written $tn \rightarrow_{t,m} tn'$, iff there exists a task network $tn^\star_m = (T_m, \prec_m, \alpha_m)$ such that $tn_m \cong tn^\star_m$ and*

$$tn' \coloneqq ((T \setminus \{t\}) \cup T_m, \prec', (\alpha \setminus \{t \to c\}) \cup \alpha_m) \text{ where}$$
$$\prec' \coloneqq \{(u,v) \in \prec \mid u \neq t \wedge v \neq t\} \cup \{(u,w) \mid (u,t) \in \prec,\ w \in T_m\} \cup \{(w,v) \mid (t,v) \in \prec,\ w \in T_m\} \cup \prec_m$$

We write $\text{tn} \rightarrow^\star_{\overline{m}} \text{tn}'$ if tn′ can be obtained from tn by applying a finite sequence $\overline{m}$ of zero or more decomposition methods. Similarly, we write $c \rightarrow^\star_{\overline{m}} \text{tn}$ if $\overline{m}$ decomposes task c into task network tn. Further, we write $\text{tn} \rightarrow' \text{tn}'$ respectively $c \rightarrow' \text{tn}'$ if there exists a sequence $\overline{m} \subset M$ of decompositions generating tn′ from tn respectively from c. Note that $\rightarrow'$ is reflexive and transitive. Repeated decomposition steps yield a decomposition tree (Geier and Bercher 2011) that records *how* respectively by which (single) method each task was refined.

Definition 5 (Decomposition Tree)**.** *A decomposition tree $g = (N, E, \prec_g, \alpha_g, \beta_g)$ for an HTN planning problem Π is a labeled directed tree with nodes N and edges E. $\prec_g$ imposes a strict partial order over the nodes; $\alpha_g \colon N \to A \cup C$ labels each node with a task name; and, additionally, $\beta_g \colon N \setminus \mathcal{L}_g \to M$ labels inner nodes with methods, where $\mathcal{L}_g \subset N$ denotes the set of all leaf nodes of g. g is* valid *with respect to Π if its root node $r \in N$ is labeled with the inital task $\alpha_g(r) = c_I$, and for every inner node $n \in N$ with $\beta_g(n) = m$, $m = (c, tn_m)$, $c \in C$, the following holds:*

1. *$\alpha_g(n) = c$ (decomposition at n is applicable to n's task)*
2. *tn_m is* isomorphic *to the task network tn_{Ch} induced by the children of n in g, denoted $\text{Ch}_g(n)$, i.e., $tn_m \cong tn_{Ch}$, where $tn_{Ch} = (\text{Ch}_g(n), \prec_{g|_{\text{Ch}_g(n)}}, \alpha_{g|_{\text{Ch}_g(n)}})$;*
3. *for all $n' \in N$ and all $c' \in \text{Ch}_g(n)$ it holds that*
   (a) *if $(n, n') \in \prec_g$ then $(c', n') \in \prec_g$, and*
   (b) *if $(n', n) \in \prec_g$ then $(n', c') \in \prec_g$*
   (children nodes inherit the order of their parent node);
4. *there are no further ordering constraints in $\prec_g$ beyond those demanded by 2 and 3.*

The yield of g, $\mathsf{yield}(g)$, is the task network restricted to the leafs $\mathcal{L}_g$ of g. Formally, $\mathsf{yield}(g) \coloneqq (\mathcal{L}_g, \prec|_{\mathcal{L}_g}, \alpha|_{\mathcal{L}_g})$.

Finally, an HTN planning problem is a tuple $\Pi = (\mathcal{D}, c_I, s_I)$, where $\mathcal{D}$ is an HTN planning domain as defined in Definition 3, $c_I \in C$ is the initial compound task, and $s_I \subseteq F$ is the initial state. A solution to an HTN planning problem Π is a primitive task network $\text{tn}_S = (T_s, \prec_s, \alpha_s)$, such that 1) tn_S is *executable* in the problem's initial state s_I, i.e., there exists a valid linearization of the task identifiers in T_s, $t_1, \ldots, t_n$ that respects ordering $\prec_s$, representing the plan $\pi = \langle \alpha_s(t_i) \mid i \in \{1, \ldots, n\} \rangle$ and π is *applicable* in the initial state s_I; and 2) there exists a sequence of decomposition methods $\overline{m}$ that decomposes the initial task c_I into tn_S, i.e., $c_I \rightarrow^\star_{\overline{m}} \text{tn}_S$. Hence, there exists a valid decomposition tree g such that for every $m = (c, tn) \in \overline{m}$ there exists a node $n \in N$ in g with $\alpha_g(n) = c$, and $\beta_g(n) = m$.

Intuitively, a solution to an HTN planning problem is a partially ordered sequence of actions, i.e, a task network, which can be derived from the initial task by decomposition methods and that achieves the desired goal.

Notation. Task identifiers are written using angle brackets, where $\langle c \rangle$ is the identifier for task c with $\alpha(\langle c \rangle) = c$. When multiple identifiers for the same task exist, we distinguish them with indices: $\langle c \rangle_1$, $\langle c \rangle_2$. We use the inverse mapping $\alpha^{-1} : A \cup C \to 2^T$ with $\alpha^{-1}(x) = \{t \in T \mid \alpha(t) = x\}$ to map a task to all its identifiers. Given a task network $\text{tn} = (T, \prec, \alpha)$, we write $T(\text{tn}) = T$, $\prec(\text{tn}) = \prec$, and $\alpha(\text{tn}) = \alpha$. In our temporal encoding, $\tau(\cdot)$ maps entities to their interval representations. We also write temporal relations in infix notation and $v \,\{r^\star\}\, v'$, with $r \in \mathsf{B_{IA}}$, when duration comparison is irrelevant.

HTN Plan Verification using QCNs

In this section, we present our novel approach to HTN plan verification through qualitative temporal reasoning. Our key insight is that HTN decomposition structures can be naturally represented as temporal intervals, where hierarchical relationships become temporal containment constraints and task orderings become precedence relations. Unlike traditional approaches that verify only totally ordered action sequences, we directly encode partially ordered primitive task networks as qualitative constraint networks (QCNs). Our encoding verifies whether a given tn_S is an ordering refinement of some decomposable task network (i.e., $\text{tn}_S \sqsubseteq \mathsf{yield}(g)$ for some valid decomposition tree g), while abstracting from action executability.

The encoding uses the INDU calculus, whose duration information ensures that subtask intervals exactly partition their parent task's interval, preventing gaps or spurious tasks that would violate decomposition semantics. Following Vilain and Kautz (1986), we introduce a separator interval S that divides the timeline into valid and invalid regions.

Definition 6 (Validity Semantics)**.** *Given a* QCN *$\mathcal{N} = (V, \Gamma)$ with separator interval $S \in V$, an interval $\mathcal{I} \in V$ is* valid *if $\mathcal{I}$ $\{after^\star\}$ S, and* invalid *if $\mathcal{I}$ $\{before^\star\}$ S.*

The QCN solver determines which decomposition choices belong to the valid solution by placing their corresponding intervals after S, while rejected alternatives remain

before S. This construction allows us to encode all possible decomposition paths within a single QCN, letting constraint propagation identify the valid decomposition tree that yields the given task network.

We construct this encoding through a three-step transformation process: (1) disambiguating task identifiers to ensure each occurrence in the solution has a unique name, (2) transforming the HTN domain into a canonical form where all decomposition choices are binary, and (3) mapping the transformed problem to temporal intervals with appropriate INDU constraints. We proceed by detailing each step and use the following running example for illustration.

Example 1 (Package Delivery Domain)**.** *Consider a simple package delivery domain with the following tasks:*

- *Initial compound task:* $c_I =$ *deliver-all*
- *Compound tasks:* $C = \{$*deliver-all*, *deliver-one*$\}$
- *Primitive tasks:* $A = \{$*pickup*, *move*, *drop*$\}$
- *Decomposition methods:*
  - m_1 : *deliver-all* $\rightarrow \langle$*deliver-one*, *deliver-all*$\rangle$
  - m_2 : *deliver-all* $\rightarrow \langle$*deliver-one*$\rangle$
  - m_3 : *deliver-one* $\rightarrow \langle$*pickup*, *move*, *drop*$\rangle$

Suppose we want to verify a solution $tn_S = (T_s, \prec_s, \alpha_s)$ *for delivering two packages, mapping to action sequence:*

$$\langle \textit{pickup}, \textit{move}, \textit{drop}, \textit{pickup}, \textit{move}, \textit{drop} \rangle$$

Step 1: Task Identifier Disambiguation

A fundamental challenge is that the same task may appear multiple times in a decomposition tree, as is the case in Example 1. In our temporal encoding, each task occurrence must map to a distinct temporal interval, even when multiple occurrences refer to the same task in the domain. While task identifiers in a decomposition tree are unique, the given solution task network tn_S may not preserve the complete decomposition structure – it only contains the primitive tasks at the leaves. To reconstruct which decomposition paths could have generated the solution, we introduce a systematic renaming procedure that ensures each task occurrence in tn_S has a unique identifier. The renaming is essential for our encoding, as it allows us to represent each task occurrence with its own interval while maintaining the connection to the original task type for matching during decomposition. Algorithm 1 implements this renaming for primitive tasks. Note that the ordering relation $\prec_s$ is preserved during renaming, as we only modify task identifiers while maintaining their relative positions.

Example 2 (Primitive Task Renaming)**.** *In our simple delivery domain (Example 1), the solution candidate* tn_S *contains six task identifiers* $t_1, \ldots, t_6$ *with:*

- $\alpha_s(t_1) = \alpha_s(t_4) =$ *pickup*
- $\alpha_s(t_2) = \alpha_s(t_5) =$ *move*
- $\alpha_s(t_3) = \alpha_s(t_6) =$ *drop*

Algorithm 1 (lines 2–4) produces:

- $t_1 \rightarrow \langle$*pickup*$\rangle_1$, $t_4 \rightarrow \langle$*pickup*$\rangle_2$
- $t_2 \rightarrow \langle$*move*$\rangle_1$, $t_5 \rightarrow \langle$*move*$\rangle_2$
- $t_3 \rightarrow \langle$*drop*$\rangle_1$, $t_6 \rightarrow \langle$*drop*$\rangle_2$

Algorithm 1: Unique renaming of primitive tasks

Input: HTN problem $\Pi = ((F, A, C, M), c_I, s_I)$, solution candidate $\text{tn}_S = (T_s, \prec_s, \alpha_s)$

1 Let $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ be any total ordering (linearization) of tn_S respecting $\prec_s$;
2 **for** *each* a_i *in* π **do**
3 Create unique identifier $\langle a \rangle_j$ where j counts occurrences of task a;
4 Update T_s, α_s to use $\langle a \rangle_j$;
5 **for** *each primitive task* $a \in A$ *in* tn_S **do**
6 Create compound task c_a; add to C;
7 Replace all occurrences of a in M with c_a;
8 **for** *each renamed identifier* $\langle a \rangle_i$ *in* T_s **do**
9 Add $(c_a, (\{\langle a \rangle_i\}, \emptyset, \{\langle a \rangle_i \mapsto a\}))$ to M;
10 Remove all methods that would decompose to primitive task occurrences not in tn_S;
11 **return** *modified* $\mathcal{D}$ *and* tn_S

Algorithm 2: Compound task duplication

Input: Modified domain $\mathcal{D} = (F, A, C, M)$, initial task c_I, solution tn_S, depth bound d

1 Initialize $\#(c) \leftarrow 0$ for all $c \in C$, $\#(a) \leftarrow |\alpha(\text{tn}_S)^{-1}(a)|$ for all $a \in A$;
2 **for** d *iterations* **do**
3 **for** $c \in C$, $M_c = \{m \in M \mid m = (c, tn_m)\}$ **do**
4 $s \leftarrow \sum_{m \in M_c} \min_{t \in T(\text{tn}_m)} \#(\alpha(t))$;
5 $\#(c) \leftarrow \max\{\#(c), s\}$;
6 **if** $\#(c_I) = 0$ **then**
7 **return** *"Invalid:* c_I *cannot yield solution"*
8 Remove $c \in C$ and $M_c \in M$ where $\#(c) = 0$;
9 **for** *each compound task* $c \in C$ *with* $\#(c) > 1$ **do**
10 Create task c_0 and copies $c_1, \ldots, c_{\#(c)}$;
11 Replace all instances of c in methods with c_0;
12 Add methods $(c_0, (\{\langle c_i \rangle\}, \emptyset, \{\langle c_i \rangle \mapsto c_i\}))$ for $i = 1, \ldots, \#(c)$;
13 **return** *modified* $\mathcal{D}$

The algorithm introduces compound tasks c_{pickup}, c_{move}, *and* c_{drop} *to replace the primitive tasks in method* m_3 *(l. 5–7), resulting in* m_3' : *deliver-one* $\rightarrow \langle c_{pickup}, c_{move}, c_{drop} \rangle$*. Finally, grounding methods for each new compound task, like* $(c_{move}, \langle move \rangle_1)$ *and* $(c_{move}, \langle move \rangle_2)$*, are created (l. 8–9).*

Compound tasks, like primitive tasks, may also appear multiple times in a valid decomposition tree. Consider a recursive task like deliver-all that decomposes into deliver-one followed by deliver-all again. Algorithm 2 determines how many copies of each compound task are needed through a bottom-up counting procedure.

The counting procedure in Algorithm 2 propagates occurrence bounds from primitive tasks up through the decomposition hierarchy. The depth parameter d is bounded by $2 \times |T_s| \times (|C| + 1)$, following theoretical results on decomposition tree height (Behnke, Höller, and Biundo 2017).

Example 3 (Compound Task Duplication)**.** *Continuing with our modified delivery domain (Example 2), Algorithm 2 computes occurrence bounds:*

- $\#(\textit{pickup}) = \#(\textit{move}) = \#(\textit{drop}) = 2$ *(from tn_S, l. 1)*
- $\#(c_{\textit{pickup}}) = \#(c_{\textit{move}}) = \#(c_{\textit{drop}}) = 2$ *(each refer to two actions in tn_S, l. 2–5)*
- $\#(\textit{deliver-one}) = 2$ *(since each delivery requires all three primitives, resp. their new dispatcher compound tasks, and they occur minimum 2 times each.)*
- $\#(\textit{deliver-all}) = 2$ *(equals $\#(\textit{deliver-one})$ by m_1, m_2)*

The algorithm creates copies (l. 9–12):

- *deliver-one$_0$ dispatches to deliver-one$_1$, deliver-one$_2$*
- *deliver-all$_0$ dispatches to deliver-all$_1$, deliver-all$_2$*
- *Similar copies for $c_{\textit{pickup}}$, $c_{\textit{move}}$, and $c_{\textit{drop}}$*

Lemma 1 (Renaming Preserves Validity)**.** *Let $\Pi = (\mathcal{D}, c_I, s_I)$ be an HTN planning problem and tn_S a primitive task network. Let $\Pi' = (\mathcal{D}', c_I, s_I)$ be the problem obtained after applying Algorithms 1 and 2. Then tn_S is a valid solution to Π if and only if the renamed task network tn'_S is a valid solution to Π'.*

Proof sketch. The renaming only introduces intermediate dispatcher tasks that non-deterministically choose among copies. Any decomposition tree for the original problem can be extended with these dispatcher nodes, and conversely, removing dispatcher nodes from a decomposition tree for the renamed problem yields a valid tree for the original problem. The removal of methods that cannot contribute to tn_S does not affect validity, since these methods were not used in any valid decomposition. □

Some decomposition methods may produce primitive tasks that do not appear in the renamed solution tn'_S. These methods cannot be part of a valid decomposition and can thus be safely removed, reducing the complexity of subsequent encoding steps. Specifically, we remove any method $m = (c, \text{tn}_m)$ where $\langle a \rangle \in T(\text{tn}_m)$ with $a \in A$, but $a \notin \{\alpha'_s(t') \mid t' \in T'_s(\text{tn}'_S)\}$.

This preprocessing ensures that our temporal encoding only considers decomposition paths that could potentially yield the given solution, significantly reducing the size of the resulting QCN while preserving correctness.

Step 2: Reduction to Binary Alternatives

After renaming, we transform the domain to ensure that each compound task has at most two decomposition methods. This simplifies the subsequent temporal encoding by reducing arbitrary n-way choices to binary decisions.

Theorem 2. *For every HTN planning domain $\mathcal{D} = (F, A, C, M)$ there exists an equivalent domain $\mathcal{D}' = (F, A, C', M')$ that admits the same set of primitive task networks and contains at most two applicable decomposition methods per compound task.*

Proof. We give a constructive proof via Algorithm 3 that iteratively replaces 3-way choices with binary ones (l. 3–8) by the equivalent transformation $\{\phi \to \psi_1, \phi \to \psi_2, \phi \to \psi_3\} \rightsquigarrow \{\phi \to \psi_1, \phi \to \omega, \omega \to \psi_2, \omega \to \psi_2\}$, introducing a new compound task c_{aux}. By recursion, the algorithm terminates when no n-way choice for $n \geq 3$ remains. □

Algorithm 3: Reduction to Binary Alternatives

```
1 Function ReduceToBinary(C, M):
2     Let M_c = {m ∈ M | m = (c, ·)} for each c ∈ C;
3     if ∃c ∈ C with |M_c| ≥ 3 then
4         Let m_1, m_2, m_3 ∈ M_c be any three methods;
5         Create auxiliary task c_aux; add to C;
6         Let m_2, m_3 decompose c_aux instead of c;
7         Add (c, ({⟨c_aux⟩}, ∅, {⟨c_aux⟩ ↦ c_aux})) to M;
8         return ReduceToBinary(C, M)
9     return (C, M)
```

Step 3: Temporal Encoding Construction in INDU

We now encode the transformed HTN plan verification instance as a qualitative constraint network using the INDU calculus. Our key insight is that HTN decomposition structures naturally map to temporal intervals. A parent task's execution spans the entire duration of its subtasks. A property we model via a *duration constraint*, enabled by the means of the INDU calculus.

The encoding faces a fundamental challenge: we must represent all possible decomposition choices within a single QCN, yet only one valid decomposition tree should emerge in the solution. Consider a compound task c with two possible decomposition methods m_1 and m_2. In a valid decomposition tree, exactly one method is chosen, but our QCN must encode both possibilities and let constraint propagation determine which belongs to the solution. We enforce this using a *temporal XOR constraint* that encodes exclusive alternatives.

Definition 7 (Temporal XOR Constraint)**.** *Let A and B be intervals representing mutually exclusive options, $\mathcal{K}$ a containment interval spanning both, and F a filler interval. The* temporal XOR-constraint *enforces:*

$A\,\{\textit{starts}^\star\}\,\mathcal{K}$	$A\,\{\textit{meets}^\star\}\,F$
$B\,\{\textit{finishes}^\star\}\,\mathcal{K}$	$B\,\{\textit{met-by}^\star\}\,F$
$X\,\{\textit{starts}^\star, \textit{finishes}^\star\}\,\mathcal{K}$	$X\,\{\textit{meets}^\star, \textit{met-by}^\star\}\,F$

where X is the choice interval that must align with exactly one of $\{A, B\}$.

Lemma 3 (Correctness of Temporal XOR Constraint)**.** *By Definition 7, interval X aligns with exactly one of $\{A, B\}$ in every solution, and both alignments are satisfiable.*

Proof. Assume, for sake of contradiction, that the choice interval X aligns with both A and B. Then, X would simultaneously need to *start** and *finish** the containment interval $\mathcal{K}$, which violates the semantics of INDU calculus. Therefore, at most one case can hold.

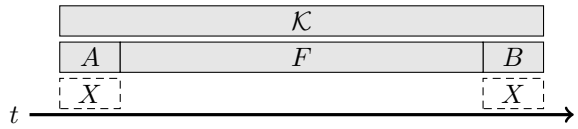

Figure 1: Timeline representation of a temporal *XOR*-constraint as defined in Definition 7. The choice interval X aligns temporally with either A or B.

Similarly, assume neither A nor B aligns with X; this contradicts the imposed constraints, as it must temporally match either endpoint of $\mathcal{K}$. Thus, exactly one interval must align with the choice interval X, ensuring exclusivity.

For showing satisfiability, one can construct models on the real line where X aligns with the temporal pattern of A (*starts** $\mathcal{K}$, *meets** F) or B (*finishes** $\mathcal{K}$, *met-by** F), while satisfying all other constraints. See Figure 1 for a visual timeline representation of these two options. □

Beyond exclusive choices, we require a mechanism to ensure that subtask durations sum exactly to their parent task's duration. The INDU calculus enables this through duration-preserving constraints.

Definition 8 (Duration Constraint)**.** *For reference interval R and element intervals $\mathcal{I}_1, \ldots, \mathcal{I}_n$, the* duration constraint *introduces auxiliary intervals $D_1, \ldots, D_n$ and R' with:*

$$\begin{array}{ll} D_i \,\{\textit{before}^=\}\, \mathcal{I}_i & \textit{for } i = 1, \ldots, n \\ D_i \,\{\textit{meets}^\star\}\, D_{i+1} & \textit{for } i = 1, \ldots, n-1 \\ D_1 \,\{\textit{starts}^\star\}\, R' & D_n \,\{\textit{finishes}^\star\}\, R' \\ R' \,\{\textit{before}^=\}\, R & \end{array}$$

The duration constraint copies each element's duration to D_i while allowing arbitrary positioning of the intervals. Sequencing all D_i creates interval R' with duration $\sum_{i=1}^{n} \mathrm{dur}(\mathcal{I}_i)$, which must equal $\mathrm{dur}(R)$.

Lemma 4. *In any solution to a QCN containing the* duration constraint *for a reference interval R and intervals $\mathcal{I}_1, \ldots, \mathcal{I}_n$, it holds that* $\mathrm{dur}(R) = \sum_{i=1}^{n} \mathrm{dur}(\mathcal{I}_i)$.

Algorithm 4 formalizes the complete encoding process. The resulting QCN $\mathcal{N}$ is satisfiable if and only if tn_S admits a valid decomposition tree g from c_I with $\mathrm{tn}_S \sqsubseteq \mathsf{yield}(g)$. For supporting our correctness proof, we introduce following *coverage property*.

Lemma 5 (Coverage Property)**.** *In any solution to the QCN constructed by Algorithm 4, if an interval $\mathcal{I}$ satisfies $\mathcal{I}\,\{\textit{after}^\star\}\, S$ and represents a compound task $c \in C$, then $\mathcal{I}$ is temporally covered by intervals representing primitive tasks through valid decomposition paths.*

Proof. By the counting procedure (Algorithm 2), compound task c has $\#(c) > 0$ only if there exists at least one decomposition path from c to primitive tasks in tn_S. The encoding ensures that (1) through temporal *XOR* constraints (Definition 7), exactly one decomposition method aligns with c; (2) through duration constraints (Definition 8), the chosen method's subtasks exactly partition c's duration; (3) this process continues recursively until primitive tasks are reached;

Algorithm 4: INDU Encoding Construction

Input: HTN domain $\mathcal{D} = (F, A, C, M)$, initial task c_I, solution candidate tn_S
Output: QCN $\mathcal{N} = (V, \Gamma)$

1 Initialize $V \leftarrow \{S\}$, $\Gamma \leftarrow \{\text{Id-relations}\}$, $A^\perp \leftarrow \emptyset$;
2 Add constraint $\tau(c_I)\,\{\textit{after}^\star\}\, S$ to Γ;
3 **forall** $t \in T_s$ **do**
4 Add $\tau(t)$ to V; add $\tau(t)\,\{\textit{after}^\star\}\, S$ to Γ
5 **forall** $(t_1, t_2) \in \prec_s$ **do**
6 Add $\tau(t_1)\,\{\textit{before}^\star, \textit{meets}^\star\}\,\tau(t_2)$ to Γ
7 **forall** $c \in C$ *with methods* m_1, m_2 **do**
8 Create intervals $\mathcal{K}_c, F_c, \tau(c_1), \tau(c_2)$; add to V;
9 Add XOR constraint (Def. 7): $X = \tau(c)$, $A = \tau(c_1)$, $B = \tau(c_2)$ with $F\{\textit{before}^\star, \textit{contains}^\star\}S$;
10 Rewrite m_i to decompose c_i for $i \in \{1, 2\}$;
11 **forall** $m = (c, \mathrm{tn}_m) \in M$ **do**
12 Add $\{\tau(c_{dcp}), \tau(c^\perp)\}$ to V, $\tau(c^\perp)\{\textit{before}^\star\}S$ to Γ ;
13 Add XOR constraint: $X = \tau(c)$, $A = c^\perp$, $B = c_{dcp}$;
14 Add all $\tau(t)$ for $t \in T(\mathrm{tn}_m)$ to V;
15 Add $\tau(t_1)\,\{\textit{before}^\star, \textit{meets}^\star\}\,\tau(t_2)$ for all $(t_1, t_2) \in \prec\,(\mathrm{tn}_m)$;
16 Add duration constraint (Def. 8): $R = \tau(c_{dcp})$, elems $= \{\tau(t) \mid t \in T(\mathrm{tn}_m)\}$;
17 **return** $\mathcal{N} = (V, \Gamma)$

(4) since $\#(c) > 0$, at least one complete decomposition path exists by construction. □

Theorem 6 (Encoding Correctness)**.** *Let $\Pi = (\mathcal{D}, c_I, s_I)$ be an HTN planning problem transformed according to Algorithms 1–3, and tn_S a primitive task network.* QCN *$\mathcal{N} = (V, \Gamma)$ constructed by Algorithm 4 is satisfiable if and only if there exists a valid decomposition tree g for Π with $\mathrm{tn}_S \sqsubseteq \mathsf{yield}(g)$.*

Proof. We prove both directions separately.

Soundness ($\Rightarrow$): Assume the QCN $\mathcal{N} = (V, \Gamma)$ has a solution $\psi : V \to \mathsf{D}$. We construct a decomposition tree $g = (N, E, \prec_g, \alpha_g, \beta_g)$ as follows:

1. For each interval $v \in V$ with $\psi(v)\,\{\textit{after}^\star\}\,\psi(S)$ (i.e., each valid interval), create a corresponding node n_v in g.
2. The root node corresponds to $\tau(c_I)$, which is valid by construction (Algorithm 4, line 2).
3. For each valid compound task interval $\tau(c)$:
   - By the XOR constraint (Definition 7), exactly one of $\{\tau(c_1), \tau(c_2)\}$ aligns with $\tau(c)$
   - This determines which method m_i was chosen for decomposing c
   - Create edges from n_c to nodes corresponding to tasks in the chosen method's task network
4. By Lemma 5, this process terminates at primitive tasks that match those in tn_S.

The resulting g is a valid decomposition tree because:

- The root is labeled with c_I (by construction)
- For each inner node n with task c, the chosen method m decomposes c (by XOR constraints)
- The children of n form a task network isomorphic to tn_m (by duration constraints and ordering preservation)
- Ordering constraints are preserved through the decomposition (Algorithm 4, line 22)
- The yield consists exactly of tasks from tn_S (by construction and grounding constraints)

Thus, $\mathrm{tn}_S \sqsubseteq \mathsf{yield}(g)$.

Completeness ($\Leftarrow$): Assume there exists a valid decomposition tree g with $\mathrm{tn}_S \sqsubseteq \mathsf{yield}(g)$. We construct a solution $\psi\colon V \to \mathsf{D}$ explicitly. Place the separator S at $[0, 1]$. For each leaf node $n \in \mathcal{L}_g$ with task identifier t, assign $\psi(\tau(t)) = [s_t, e_t]$ where the intervals are pairwise disjoint, after S, and respect the ordering $\prec_s$. Since $\mathrm{tn}_S \sqsubseteq \mathsf{yield}(g)$, we have $\prec_g |_{L_g} \subseteq \prec_s$, ensuring that respecting $\prec_s$ also respects all constraints from the decomposition tree. Specifically, if $(t_1, t_2) \in \prec_s$, ensure $e_{t_1} \leq s_{t_2}$.

For each inner node n with compound task c and chosen method $m = (c, \mathrm{tn}_m)$, let the children implement tasks with intervals $\mathcal{I}_1, \ldots, \mathcal{I}_k$. Set $\psi(\tau(c)) = [\min_i s_{\mathcal{I}_i}, \max_i e_{\mathcal{I}_i}]$ to exactly span its children. For the corresponding XOR structure: if g uses m_1, set $\psi(\tau(c_1)) = \psi(\tau(c))$ and place $\tau(c_2)$ before S; if g uses m_2, do the opposite. Set the filler F_c to connect these regions and $\mathcal{K}_c$ to span both alternatives, satisfying Definition 7.

For each auxiliary interval D_i in duration constraints, place it appropriately to satisfy Definition 8. For primitive tasks $a^{\perp}$ not used in g, place their intervals before S. This construction ensures all constraints are satisfied: validity constraints separate used/unused tasks via S; XOR constraints are satisfied by aligning exactly one alternative with each compound task; duration constraints hold by construction as parent intervals span their children exactly; and ordering constraints from both tn_S and decomposition methods are preserved.

The construction yields a consistent assignment provided decomposition methods have no internal ordering cycles (e.g., $t_1 \prec t_2$ and $t_2 \prec t_1$), which would make them unusable in any HTN context. Therefore, $\mathcal{N}$ is satisfiable. □

It remains to be shown that our construction is indeed polynomial in the size of the input.

Theorem 7 (Polynomial Construction)**.** *The size of the* QCN *constructed by our encoding is polynomial in the size of the input HTN planning problem and solution candidate* tn_S.

Proof. Algorithm 1 introduces at most $|T(\mathrm{tn}_S)|$ new compound tasks and $|T(\mathrm{tn}_S)|$ new methods. Algorithm 2 creates at most $\#(c)$ copies for each compound task $c \in C$. Since $\#(c) \leq |T(\mathrm{tn}_S)|$ for any c, the total number of new compound tasks is bounded by $|C| \times |T(\mathrm{tn}_S)|$. The binary reduction (Algorithm 3) may introduce additional auxiliary tasks, but at most $O(|M|) = O(|C|)$ since we start with at most $|M|$ methods. The final QCN has $O((|C| + |A|) \times |T(\mathrm{tn}_S)|)$ variables and $O(|V|^2)$ constraints. Thus, the overall construction remains polynomial in the input size. □

Corollary 8. *Our encoding of HTN plan verification as a QCN preserves NP-completeness. The upper bound holds because QCN satisfiability is in NP and the encoding runs in polynomial time. The lower bound follows directly from existing NP-hardness results for HTN plan verification. A particular utility of the proposed encoding is its compactness, which may lead to efficiency gains compared to alternative encodings such as SAT.*

Conclusion and Future Work

We propose a novel approach for verifying partially ordered task networks in HTN planning using qualitative constraint networks. Our key contribution is establishing an explicit connection between HTN decomposition structures and temporal reasoning, showing how hierarchical task relationships naturally map to qualitative temporal constraints. Unlike existing approaches requiring totally ordered plans, our method preserves the inherent flexibility of partial orders and enables integration of temporal and duration constraints expressible in INDU. The resulting encoding is exhaustive, enabling a QCN solver to identify all valid decomposition trees that yield (an ordering refinement of) the given primitive task network.

Our encoding verifies whether a given primitive task network can be derived through valid hierarchical decompositions with ordering refinement, abstracting from action executability. A preliminary implementation validates the theoretical encoding, though current general-purpose INDU solvers are not yet optimized for the constraint patterns arising in this domain. Natural extensions include integrating preconditions and effects, and supporting methods with empty task networks.

Our QCN-based framework opens several promising research directions. Recent advances in finding maximally satisfiable subsets of QCNs (Condotta et al. 2015; Condotta, Nouaouri, and Sioutis 2016; Sioutis 2023; Bayerkuhnlein, Schwartz, and Wolter 2024) could identify specific causes of invalidity and suggest repairs.

The framework also shows promise for verifying more expressive planning formalisms. HTN planners like CHIMP (Stock et al. 2015) and FAPE (Dvorak et al. 2014; Bit-Monnot et al. 2020) already employ rich constraint-based task representations. Our temporal encoding could potentially handle such domains directly, as QCNs naturally support various qualitative reasoning modalities beyond pure temporal relations.

Perhaps most intriguingly, the natural correspondence between decomposition trees and temporal interval networks suggests the possibility of solving HTN planning problems directly through temporal reasoning. This would represent a fundamental shift in how we approach hierarchical planning, potentially leading to new algorithms that exploit the deep connection between hierarchical structure and temporal constraints.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 832–843.

Balbiani, P.; Condotta, J.; and Ligozat, G. 2006. On the consistency problem for the *INDU* calculus. *Journal of Applied Logic*, 4(2): 119–140.

Barták, R.; Ondrčková, S.; Behnke, G.; and Bercher, P. 2021. On the Verification of Totally-Ordered HTN Plans. In *Proceedings of 33rd International Conference on Tools with Artificial Intelligence (ICTAI 2021)*, 263–267.

Barták, R.; Ondrčková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A Novel Parsing-based Approach for Verification of Hierarchical Plans. In *Proceedings of 32nd International Conference on Tools with Artificial Intelligence (ICTAI 2020)*, 118–125.

Bayerkuhnlein, M.; Schwartz, T.; and Wolter, D. 2024. From Resolving Inconsistencies in Qualitative Constraints Networks to Identifying Robust Solutions: A Universal Encoding in ASP. In *Proceedings of the 47th German Conference on AI, KI 2024*, volume 14992 of *LNCS*, 3–16. Springer.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*, volume 25, 25–33. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This Is a Solution! (... But Is It Though?) - Verifying Solutions of Hierarchical Planning Problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling, ICAPS 2017*, 20–28. AAAI Press.

Bit-Monnot, A.; Ghallab, M.; Ingrand, F.; and Smith, D. E. 2020. FAPE: a Constraint-based Planner for Generative and Hierarchical Temporal Planning. *CoRR*, abs/2010.13121.

Condotta, J.; Nouaouri, I.; and Sioutis, M. 2016. A SAT Approach for Maximizing Satisfiability in Qualitative Spatial and Temporal Constraint Networks. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, 432–442. AAAI Press.

Condotta, J.-F.; Mensi, A.; Nouaouri, I.; Sioutis, M.; and Saïd, L. B. 2015. A Practical Approach for Maximizing Satisfiability in Qualitative Spatial and Temporal Constraint Networks. In *Proceedings of the 27th International Conference on Tools with Artificial Intelligence (ICTAI 2015)*, 445–452.

Dvorak, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and Acting with Temporal and Hierarchical Decomposition Models. In *Proceedings of the 26th International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, 115–121. IEEE Computer Society.

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 Computing Surveys*, 50. Article 7.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961. IJCAI/AAAI.

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2022. Compiling HTN Plan Verification Problems into HTN Planning Problems. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 145–150. AAAI Press.

Ligozat, G. 2011. *Qualitative Spatial and Temporal Reasoning*. John Wiley & Sons.

Lin, S.; Behnke, G.; and Bercher, P. 2023. Accelerating SAT-Based HTN Plan Verification by Exploiting Data Structures from HTN Planning. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 1489–1496. IOS Press. ISBN 9781643684376.

Lin, S.; Behnke, G.; Ondrčková, S.; Barták, R.; and Bercher, P. 2023. On Total-Order HTN Plan Verification with Method Preconditions – An Extension of the CYK Parsing Algorithm. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10): 12041–12048.

Ondrčková, S.; Barták, R.; Bercher, P.; and Behnke, G. 2023. On the Impact of Grounding on HTN Plan Verification via Parsing. In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023)*, 92–99.

Pantucková, K.; and Barták, R. 2023. Using Earley Parser for Recognizing Totally Ordered Hierarchical Plans. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 1819–1826. IOS Press.

Pantucková, K.; Ondrcková, S.; and Barták, R. 2024. Using Earley Parser for Verification of Totally Ordered Hierarchical Plans. In *Proceedings of the 37th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2024)*. Florida Online Journals.

Pujari, A. K.; Kumari, G. V.; and Sattar, A. 1999. INDU: An Interval and Duration Network. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI 1999)*, volume 1747 of *LNCS*, 291–303. Springer.

Sioutis, M. 2023. Embarrassingly Greedy Inconsistency Resolution of Qualitative Constraint Networks. In *Proceedings of the 30th International Symposium on Temporal Representation and Reasoning (TIME 2023)*, volume 278, 12:1–12:12.

Sioutis, M.; and Wolter, D. 2021. Qualitative Spatial and Temporal Reasoning: Current Status and Future Challenges. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4594–4601.

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 2015)*, 6459–6464. IEEE.

Vilain, M.; and Kautz, H. 1986. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986)*, 377–382. AAAI Press.