Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

# SibylSatOpt: a MaxSAT-based Greedy Optimal Search for TOHTN Planning

**Gaspard Quenard, Damien Pellier, Humbert Fiorino**

University Grenoble Alpes, LIG, F-38000 Grenoble, France
gaspard.quenard@univ-grenoble-alpes.fr damien.pellier@univ-grenoble-alpes.fr, humbert.fiorino@univ-grenoble-alpes.fr

## Abstract

This paper introduces SibylSatOpt, a novel approach to finding optimal plans for Totally-Ordered HTN (TOHTN) problems by leveraging greedy search techniques with MaxSAT. Unlike previous SAT-based HTN planners that employed a blind breadth-first search strategy, SibylSatOpt is guided by an admissible heuristic. This heuristic combines a relaxed MaxSAT encoding of the problem with the Task Decomposition Graph (TDG) heuristic. As we demonstrate, the admissibility of the heuristic guarantees that the found solution is optimal. Experimental results on IPC benchmarks show that SibylSatOpt significantly outperforms existing optimal TOHTN planners in both runtime and problem coverage.

## Introduction

Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1994) is a specific planning technique focused on breaking down complex tasks into simpler subtasks. Unlike classical planning, it introduces the concept of abstract tasks (high-level actions which cannot be executed directly), and methods, which specify a partially ordered set of primitive subtasks and additional abstract subtasks to execute in order to achieve the abstract task. The goal of an HTN planner is to transform an initial abstract task into a plan, i.e., an executable sequence of actions. HTN planning is a prominent area of research, and it is now part of the International Planning Competitions (IPC) (Behnke, Höller, and Bercher 2021; Taitler et al. 2024).

This paper focuses on Totally-Ordered HTN (TOHTN) planning, where the methods specify a totally-ordered sequence of actions and tasks. A considerable corpus of research has been devoted to the development of agile TOHTN planners (i.e., planners designed to find solutions as quickly as possible). These planners can be categorized into three main groups: those utilizing heuristic search techniques specifically tailored for HTN (e.g., (Bercher et al. 2017; Höller et al. 2020; Höller, Bercher, and Behnke 2020; Höller and Bercher 2021; Olz, Lodemann, and Bercher 2024)); those extracting classical (i.e., non-hierarchical) planning heuristics from the HTN-to-STRIPS translation or directly converting HTN problems into STRIPS to use classical planners (e.g., (Alford, Kuter, and Nau 2009; Alford et al. 2016; Höller et al. 2018, 2019; Behnke et al. 2022)), and those encoding planning problems into propositional logic, benefiting from the efficiency of optimized solvers (e.g., (Behnke, Höller, and Biundo 2018; Schreiber et al. 2019; Behnke, Höller, and Biundo 2019; Behnke 2021; Schreiber 2021; Quenard, Pellier, and Fiorino 2024)). In contrast, there has been comparatively less work on *optimal* (TO)HTN planners that seek to find solutions with the cheapest plan. Among the aforementioned works, only (Bercher et al. 2017; Olz, Lodemann, and Bercher 2024); (Höller et al. 2018, 2019); and (Behnke, Höller, and Biundo 2019) are capable of producing optimal plans. However, in many practical applications, finding an optimal plan is as crucial as finding a valid plan. For instance, in logistics domains, cost savings are directly linked to minimizing the overall plan cost.

In the approaches previously mentioned, SAT-based encoding techniques are among the most efficient for TOHTN planning, benefiting significantly from performance improvements of SAT solvers. Prior to recent advancements, SAT-based planners all followed a 'breadth-first search' incremental approach, limiting the decompositions of the initial abstract task to a depth of $k$. A SAT formula is constructed to capture all possible plans in which the initial task can be decomposed up to $k$ times. If no solution is identified by the SAT solver, the value of $k$ is increased, expanding the search to deeper levels of decomposition.

In recent work, the satisficing SibylSat planner (Quenard, Pellier, and Fiorino 2024) introduced an alternative approach that improves SAT-based planners' performance through a greedy search strategy. By approximating abstract tasks with preconditions and effects (Olz, Biundo, and Bercher 2021), SibylSat can temporarily treat them as primitive tasks. Using this relaxed representation, the SAT solver identifies a single 'abstract plan' (i.e., a plan containing both abstract tasks and actions) that appears promising for decomposition into a solution and develops it further. If this expansion doesn't yield a solution, the process repeats: SibylSat finds another abstract plan in the enlarged search space and expand it, continuing until a solution is found. When multiple abstract plans are possible, the SAT solver functions as a black-box oracle, autonomously selecting one to expand. While this approach significantly improves performance, it provides no guarantees about the quality of the final solution, as the selected abstract plans may lead to a

high-cost solution.

The contribution of SibylSatOpt in this paper is to address this limitation of the SibylSat planner by guaranteeing optimal plans. Instead of relying on arbitrary selections of abstract plans by the SAT solver, we introduce an admissible approach that encodes the TDG heuristic with a MaxSAT solver to always expand the abstract plans that would lead to the lowest-cost solution. Experimentally, we show that this approach significantly outperforms other state-of-the-art optimal TOHTN planners on most of the IPC benchmarks.

This paper is organized as follows: First, we introduce the concept of TOHTN planning and the TDG heuristic. Next, we describe the SibylSatOpt planner. Finally, we compare it against state-of-the-art optimal TOHTN planners.

## TOHTN Planning Problem

We present a formalization of TOHTN planning, building on (Behnke, Höller, and Biundo 2018; Behnke 2021; Quenard, Pellier, and Fiorino 2024).

### Task, Action, Methods, and Task Networks

*Tasks* are central to HTN planning. A *task* is defined by a name and a list of parameters. For simplicity, we consider the parameters as constants and HTN planning problems as grounded. There are two types of tasks: *primitive tasks* and *abstract tasks*. Primitive tasks directly affect the state of the world, while abstract tasks do not; instead, abstract tasks must be decomposed into primitive tasks using *methods* before they can be executed. A *task network*, denoted $w$, is a sequence of tasks that can be either abstract or primitive. A *primitive task network* contains only primitive tasks.

A primitive task $a$ is analogous to an action in classical planning and is defined by a tuple $(\mathit{name}(a), \mathit{precond}(a), \mathit{effect}(a), \mathit{cost}(a))$. Here, $\mathit{name}(a)$ represents the name of $a$, $\mathit{precond}(a)$ and $\mathit{effect}(a)$ represent the sets of propositions for preconditions and effects, respectively, and $\mathit{cost}(a)$ represent the cost associated with executing the action (defaulting to 1). An action $a$ is considered executable in a state $s$, defined as a set of propositions describing the world, iff $\mathit{precond}(a) \subseteq s$. If $A$ represents the set of actions and $S$ represents the set of states, then the state transition function $\gamma : S \times A \to S$ is defined as follows: If $a$ is executable in $s$, then $\gamma(s, a) = (s \setminus \mathit{effect}^-(a)) \cup \mathit{effect}^+(a))$; otherwise, $\gamma(s, a)$ is undefined. The extension of $\gamma$ to a primitive task network $w$, denoted $\gamma^* : S \times A^* \to S$, is defined straightforwardly. A primitive task network $w = \langle a_1, \ldots, a_n \rangle$ is executable in a state $s$ iff the sequence of states $\langle s_1, \ldots, s_n \rangle$ exists such that $\forall i \in \{1, \ldots, n\}$, $a_i$ is applicable in $s_i$ and $\gamma(a_i, s_{i-1}) = s_i$. The cost of such a primitive task network is defined as the sum of the costs of its constituent actions, i.e., $\mathit{cost}(w) = \sum_{i=1}^{n} \mathit{cost}(a_i)$.

A method $m$ indicates how an abstract task can be refined into a task network. It is defined as a tuple $(\mathit{name}(m), c, w_m)$ where $c$ is an abstract task and $w_m$ is a task network. We refer to $c$ as the abstract task decomposed by $m$ and $w_m$ as the decomposition of $m$ in subtasks. Given a task network $w = w_1 c w_2$ where $c$ is an abstract task, applying a method $m = (\mathit{name}(m), c, w_m)$ to $w$ will result in the task network $w' = w_1 w_m w_2$ (written as $w \to^m w'$). We denote $w \to^* w'$ the transitive and reflexive closure of $\to$, i.e., $\to^*$ hold if zero or more methods can be applied to obtain $w$ into $w'$. For notation purposes, we define as $M(c) = \{m = (name(m), c, w_m) \mid m \in M\}$ the set of all the methods which can be applied to decompose the abstract task $c$.

### Planning Problem and Solution

**Definition 1** (TOHTN Planning Problem)**.** *A TOTHN Planning problem $P$ is a tuple $(L, C, A, M, c_I, s_I, g)$ where: $L$ is a finite set of propositions; $C$ is a finite set of abstract tasks; $A$ is a finite set of primitive tasks (or actions); $M$ is a finite set of decomposition methods (or methods); $c_I \in C$ is the initial abstract task to be decomposed; $s_I \subseteq L$ is the initial state and $g$ is the (possibly empty) goal state.*

Solving a planning problem $P$ involves finding a primitive task network (or *plan*) $\pi$ which can be decomposed from the initial abstract task ($c_I \to^* \pi$) such that $\pi$ is executable in the initial state $s_I$ and reaches the goal $g$ after execution, i.e., $g \subseteq \gamma(s_I, \pi)$. Unlike classical planning, a solution to a TOHTN planning problem is not merely a sequence of primitive tasks that can be executed in the initial state and achieve a goal. It must also explicitly represent the methods employed to derive this sequence of primitive tasks from the initial abstract task. The decomposition process can be depicted as a tree known as a *decomposition tree* (Geier and Bercher 2011), which illustrates the complete trace of how an abstract task is refined into a task network.

**Definition 2** (Decomposition tree)**.** *A decomposition tree $T = (N, E)$ for a task network $w$ of a planning problem $P = (L, C, A, M, c_I, s_I, g)$ is a tree with:*

- *$N$ a set of nodes labelled by either a primitive task in $A$, an abstract task in $C$ or a method in $M$,*
- *$E : N \to N^*$ an edge function which provides for every node an ordered list of children $\langle e_1, e_2, \ldots, e_k \rangle$,*
- *For every inner node $n \in N$ labelled by an abstract task $c$, $|E(n)| = 1$ and $E(n) = \langle n' \rangle$ where $n'$ is labelled by a method $m \in M(c)$.*
- *For every inner node $n \in N$ labelled by a method $m$, let us consider $\langle t_1, t_2, \ldots, t_k \rangle$ the subtasks of $m$, then $|E(n)| = k$ and $e_i$ is labelled by the task $t_i$ for all $e_i \in E(n) = \langle e_1, e_2, \ldots, e_k \rangle$.*
- *For the sequence of leaves $L = \langle n_1^l, n_2^l, \ldots, n_k^l \rangle$, it holds that each leaf if labelled by either a primitive task or an abstract task and if we consider the corresponding sequence of tasks $\langle t_1, t_2, \ldots, t_k \rangle$, then we have $w = \langle t_1, t_2, \ldots, t_k \rangle$.*

We can now formally define a solution for a TOHTN planning problem as follows:

**Definition 3** (Decomposition Tree solution)**.** *Let $P = (L, C, O, M, c_I, s_I, g)$ be a planning problem. Consider $T_{sol}$ as the decomposition tree for the task network $\pi$ for the problem $P$. The decomposition tree $T_{sol}$ is a solution for $P$ iff it satisfies the following characteristics:*

1. *The root of $T_{sol}$ is the initial abstract task $c_I$.*

2. *$\pi$ only contains primitive tasks.*
3. *$\pi$ is executable in the initial state $s_I$.*
4. *$\pi$ reaches the goal $g$ after execution.*

*The decomposition tree $T_{sol}$ is considered optimal if, for any other solution $T'$ with the the task network $\pi'$, the cost of executing $\pi$ is less than or equal to the cost of executing $\pi'$, i.e., cost$(\pi) \leq$ cost$(\pi')$.*

## Path Decomposition Tree and SAT Planners

Since solving an HTN planning problem is equivalent to finding a decomposition tree (DT) that satisfies specific criteria (Definition 3), one approach is to explore all possible DTs rooted with the initial abstract task $c_I$ for a given problem $P$. However, the search space of such potential DTs is vast in most domains and infinite in recursive domains, where an abstract task can be decomposed into itself.

To address this challenge, (Behnke, Höller, and Biundo 2018; Schreiber et al. 2019) proposed incrementally expandable structures that represent subsets of all possible DTs rooted in $c_I$. When no solution is found within a structure, it can be expanded to include additional DTs, ensuring eventual coverage of all possible solutions. We adopt an isomorphic representation of these structures introduced by Quenard, Pellier, and Fiorino (2024).

**Definition 4** (Path Decomposition Tree)**.** *A path decomposition tree (PDT) for a problem $P = (L, C, A, M, c_I, s_I, g)$ is a tree $\Gamma = (N, E)$ with:*

- *$N$ a set of nodes labelled by either a primitive task in $A$, an abstract task in $C$ or a method in $M$,*
- *$E : N \rightarrow N^*$ an edge function which provides for every node an ordered list of children $\langle e_1, e_2, ..., e_k \rangle$.*

*Let $r \in N$ be the root node of the PDT. The PDT is well-formed iff:*

- *$r$ is labelled by the initial abstract task of the problem $c_I$.*
- *For every inner node $n$ labelled by an abstract task $c$, $|E(n)| = |M(c)|$ and $\forall m_i \in M(c), \exists e_i \in E(n)$ such that $e_i$ is labelled by the method $m_i$.*
- *For every inner node $n$ labelled by a method $m$, let us consider $\langle t_1, t_2, \ldots, t_k \rangle$ the subtasks of $m$, then $|E(n)| = k$ and $e_i$ is labelled by the task $t_i$ for all $e_i \in E(n) = \langle e_1, e_2, \ldots, e_k \rangle$.*
- *Every leaf is labelled by either a primitive task or an abstract task.*

Figure 1 depicts a PDT for a problem $P = (L, C, A, M, c_I, s_I, g)$ at a certain level of decomposition, and highlights a DT for $P$ in light blue. This DT decomposes the initial task $c_I$ into a task network $w = \langle a_2, a_1, a_3 \rangle$. It is solution for $P$ iff $w$ is executable in the initial state $s_I$ and satisfies the goal condition $g \subseteq \gamma(s_I, w)$.

The search for a solution within a given PDT is the approach employed by all current SAT-based TOHTN planners (Schreiber et al. 2019; Schreiber 2021; Behnke, Höller, and Biundo 2018; Behnke 2021; Quenard, Pellier, and Fiorino 2024). The key idea is to encode a PDT (or an isomorphic equivalent structure) into a formula that is satisfiable if and only if a solution exists within the PDT. If the formula is satisfiable, the solution can be extracted from the satisfying assignment. Otherwise, the PDT can be expanded by developing any number of its *pending nodes* (leaf nodes labelled with abstract tasks). For each selected node labelled with abstract task $c$, all applicable methods $M(c)$ are added as children, with each method's subtasks added in order.

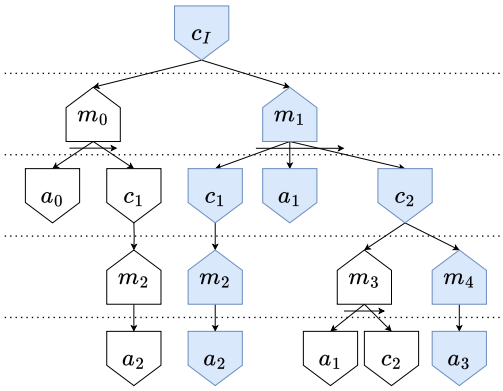


Figure 1: Example illustrating a PDT at an intermediate level of decomposition for a problem. This PDT does not encompass all possible DTs, as the leaf abstract task $c_2$ remains undeveloped. A potential solution is highlighted in light blue.

Prior to the SibylSat planner (Quenard, Pellier, and Fiorino 2024), existing encodings expanded the PDT in a breadth-first manner, developing *all* pending nodes. In contrast, SibylSat relaxed the PDT by temporarily treating each pending node (i.e., leaf abstract task) as an action with preconditions and effects derived from its possible decompositions (details are provided in a later section). A SAT call on this relaxed PDT returns an "abstract" solution whose leaves include both actions and abstract tasks. Only the abstract leaves appearing in the solution are subsequently developed in the actual PDT. This method led to significantly fewer nodes being developed before finding a solution on most benchmarks (Quenard, Pellier, and Fiorino 2024) and forms the foundation for the optimal search strategy introduced in the following sections.

## On the TDG Heuristic and its Extension to Decomposition Trees

To effectively guide its search toward optimal plans, SibylSatOpt relies on an admissible heuristic that estimates the lowest cost plan achievable from any DTs. This heuristic is computed from a dedicated structure known as the Task Decomposition Graph (TDG). In the following, we introduce the TDG, define how it can provides admissible cost estimates, and extend it to apply over DTs.

The TDG is a finite AND/OR representation of the task hierarchy structure, first introduced by Elkawkagy et al. (2012), and later used by Bercher et al. (2017) to propose an admissible heuristic function. Its construction is similar to that of the PDT, but in the TDG, each method or task appears only once, ensuring a finite structure that may contain

cycles. Formally, it can be defined as follows:

**Definition 5** (Task Decomposition Graph)**.** *A task decomposition graph $\mathcal{G} = (N, E)$ for a problem $P = (L, C, A, M, c_I, s_I, g)$ is a directed graph with:*

- *$N$, a set of nodes labelled by either a primitive task in $A$, an abstract task in $C$, or a method in $M$,*
- *$E : N \to N^*$, an edge function that assigns to each node an ordered list of children $\langle e_1, e_2, \ldots, e_k \rangle$.*

*We call the $TDG$ well-formed iff:*

- *One node in $\mathcal{G}$ is labelled with the initial abstract task of the problem, $c_I$.*
- *For every node $n$ labelled by an abstract task $c$, $|E(n)| = |M(c)|$ and $\forall m_i \in M(c), \exists e_i \in E(n)$ such that $e_i$ is labelled by the method $m_i$.*
- *For every node $n$ labelled by a method $m$, let $\langle t_1, t_2, \ldots, t_k \rangle$ be the subtasks of $m$; then $|E(n)| = k$, and each $e_i$ in $E(n) = \langle e_1, e_2, \ldots, e_k \rangle$ is labelled by the task $t_i$.*
- *No two nodes in $\mathcal{G}$ can be labelled with the same task or method.*

Let $v_t$ be the node labelled with task $t$ in the TDG. Bercher et al. (2017) propose the following recursive formula for an admissible estimate of the minimal-cost primitive task network obtained by decomposing $t$:

$$h(v_t) = \begin{cases} cost(t) & \text{if } t \text{ is as a primitive task} \\ \min\limits_{v_i \in child(v_t)} h(v_i) & \text{if } t \text{ is an abstract task} \\ \sum\limits_{v_i \in child(v_t)} h(v_i) & \text{otherwise} \end{cases}$$

They also proposed a polynomial-time fixpoint algorithm to compute the heuristic values for all nodes in the TDG, even in the presence of cycles. This algorithm iteratively applies the above formula until convergence.

The TDG for the problem depicted in Figure 1 is presented in Figure 2, where each node is annotated with its corresponding heuristic value. Building on this, we extend the TDG heuristic to decomposition trees as follows:

**Definition 6** (DT heuristic)**.** *Let $P$ be a planning problem and its corresponding TDG. Consider $T$ as a decomposition tree of $P$ whose leaves define a task network $w = \langle t_1, t_2, \ldots, t_n \rangle$. The heuristic value $h(T)$ of $T$ is defined as the sum of the heuristic value of its leaf tasks:*

$$h(T) = \sum_{k=1}^{n} h(t_k)$$

*where $h(t_k)$ is the TDG heuristic value of the task $t_k$.*

For a solution, this heuristic value gives the cost of the plan solution. Let us show that this heuristic is admissible, that is, the heuristic value of a DT $T$ underestimates the cost of any solution that can be obtained by decomposing $T$. Or, in other words, for a solution $T_{sol}$, any DT that can be refined into $T_{sol}$ must have a heuristic value less than or equal to $h(T_{sol})$. To formalize this, we introduce the concept of $PredecessorDTs$, which represents the set of all DTs that can be refined into a given solution $T$.

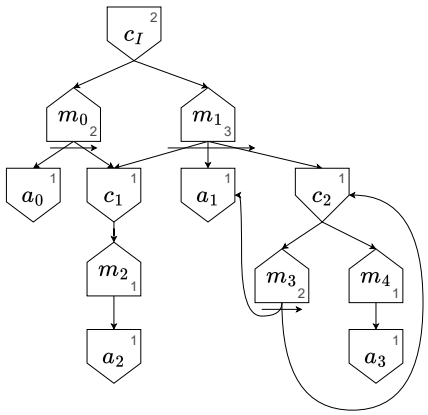


Figure 2: TDG associated with the problem illustrated in Figure 1. Each node is given its TDG heuristic value.

**Definition 7.** *Given two DT $T_1$ and $T_2$, we say that $T_1$ is a refinement predecessor of $T_2$ (denoted $T_1 \prec T_2$) if $T_2$ can be obtained from $T_1$ by further decomposing abstract tasks in $T_1$. This means $T_1$ can be expanded into $T_2$ through a series of method applications.*

**Definition 8** (Predecessor Decomposition Trees Function)**.** *Let $T$ be a DT. We define the function $PredecessorDTs(T)$ as the set of all DTs that can be refined into $T$:*

$$PredecessorDTs(T) = \{T' \mid T' \prec T\}$$

For example, in Figure 1, The set of DTs leading to the potential solution highlighted in light blue are as follows:

- $c_I \to^{m_1} (c_1 \to^{m_2} (a_2)\ a_1\ c_2)$,
- $c_I \to^{m_1} (c_1\ a_1\ c_2 \to^{m_4} (a_3))$,
- $c_I \to^{m_1} (c_1\ a_1\ c_2)$,
- $c_I$.

**Theorem 1.** *Let $P = (L, C, A, M, c_I, s_I, g)$ be a planning problem. For any solution $T_{sol}$ with the heuristic value $h(T_{sol})$, any DT $T' \in PredecessorDTs(T)$ satisfies $h(T') \leq h(T_{sol})$.*

*Proof sketch.* Given a solution $T_{sol}$, consider any DT $T' \in PredecessorDTs(T_{sol})$. For each leaf $t$ in $T'$:

1. If $t$ is a primitive task, it is also a leaf in $T_{sol}$, so its heuristic value is $h(t) = cost(t)$.
2. If $t$ is an abstract task, then in $T_{sol}$, $t$ is fully decomposed into primitive tasks whose total cost is $costdecomp(t)$. Since the TDG-based heuristic $h(t)$ assigns to $t$ the minimal cost of any primitive task network obtainable by refining $t$, we have $h(t) \leq costdecomp(t)$.

By summing the heuristic values over all leaves in $T'$, we get $h(T') = \sum_{t \in Leaves(T')} h(t) \leq \sum_{t \in Leaves(T_{sol})} cost(t) = h(T_{sol})$. Therefore, the heuristic value of any DT $T' \in PredecessorDTs(T_{sol})$ is less than or equal to that of the solution $T_{sol}$. □

## SibylSatOpt

### Planning Approach

Like other SAT-based TOHTN planners, SibylSatOpt uses a PDT to represent its current exploration of the search space and alternates between searching for a solution in the PDT and expanding it until a solution (here optimal) is found. SibylSatOpt follows the same core expansion approach proposed by SibylSat (Quenard, Pellier, and Fiorino 2024): both planners use relaxation by approximating abstract tasks with preconditions and effects to identify promising parts of the PDT to expand. However, while SibylSat lets the SAT solver freely choose which promising part to develop, SibylSatOpt adds constraints based on an admissible heuristic. These constraints ensure that the solver only considers parts of the search space that could lead to optimal solutions.

We present a high-level overview of the main procedure of SibylSatOpt, outlined in Algorithm 1. The algorithm begins by grounding the problem using the pandaPIgrounder (Behnke et al. 2020). Next, it constructs the TDG with the grounded methods and tasks that might appear in this problem, and computes their admissible costs using the approach proposed by Bercher et al. (2017) (line 2). After this preprocessing step, the PDT — which represents the current exploration of the search space — is initialized with a single node labelled with the problem's initial abstract task (line 3). With this setup, the main loop of SibylSatOpt can begin.

The main loop begins by assigning costs to the leaves of the PDT (line 5) using the TDG heuristic computed during the preprocessing. The algorithm then identifies two solutions. First, it finds the optimal solution within the current PDT (line 6), referred to as the optimal primitive solution, as its task network contains only primitive tasks. The specific process of finding an optimal primitive solution in a PDT will be detailed in a subsequent section.

This optimal primitive solution, however, is not guaranteed to be optimal for the entire problem, as some abstract task leaves may still be developed further in the PDT, potentially leading to a lower-cost solution. To address this, SibylSatOpt applies a relaxation technique introduced in the original SibylSat paper (Quenard, Pellier, and Fiorino 2024), which translates abstract task leaves into primitive tasks with inferred preconditions and effects. These preconditions and effects are inferred to ensure that any refinement predecessor of a solution (as defined in Definition 7) will also be a valid solution in the relaxed PDT. However, since this inference is heuristic-based, further developing a solution found in the relaxed PDT may not always yield a valid solution.

Once the PDT has been relaxed, the planner searches for the optimal abstract solution within this relaxed PDT (line 7). This abstract solution's task network can contain both primitive and abstract tasks, and is optimal in the sense that no other abstract solution within the relaxed PDT has a lower cost. The details of finding this optimal abstract solution will be explained in a subsequent section.

If no optimal abstract solution is found (which also means that no optimal primitive solution is found, since every primitive solution is an abstract one), the problem is declared unsatisfiable, and the algorithm terminates (line 9). If both solutions are found, the algorithm compares their costs (line 10). If the abstract solution has a lower cost, then it could potentially be developed into a cheaper solution, due to the admissibility of the heuristic (Theorem 1). Conversely, if the cost of the abstract solution is equal to or greater than that of the primitive solution, it cannot be further refined into a lower cost solution, again by the admissibility of the heuristic (Theorem 1). If the cost comparison shows that the primitive solution is optimal for the entire problem, it is returned (line 11). Otherwise, or if no primitive solution has been found, the algorithm expands the PDT using the expansion method proposed in the SibylSat paper (Quenard, Pellier, and Fiorino 2024) (line 13). Specifically, any abstract task that is part of the optimal abstract solution's task network will have its corresponding leaf in the PDT developed.

Algorithm 1: SibylSatOpt Planner

```
1:  procedure SibylSatOp(P = (L, C, A, M, c_I, s_I, g))
2:      TDG ← ComputeTDG(P)
3:      PDT ← InitializePDT(P)
4:      while True do
5:          PDT ← SetCostLeaves(PDT, TDG)
6:          DT_sol ← FindOptSol(PDT)
7:          DT_abstract ← FindOptAbstractSol(PDT)
8:          if DT_abstract == ∅ then
9:              return ∅
10:         else if DT_sol ≠ ∅ and
                Cost(DT_sol) == Cost(DT_abstract) then
11:             return DT_sol
12:         end if
13:         PDT ← ExpandPDT(PDT, DT_abstract)
14:     end while
15: end procedure
```

### Example

To illustrate the SibylSatOpt algorithm, we describe how it develops the PDT for the problem $P$ in Figure 1. The planner begins by grounding the problem and applying the TDG heuristic to find admissible values for all ground tasks and methods. It then initializes the PDT with a root node, $c_I$, representing the initial abstract task, and enters the main loop.

In the first iteration, it starts by assigning the heuristic cost to each leaf of the PDT (in this case, only $c_I$). It then tries to find an optimal primitive solution in the PDT but finds none. Next, it relaxes the PDT by treating all abstract task leaves (here, $c_I$) as actions, and searches for an optimal abstract solution in the relaxed PDT. The abstract solution with the task network $\langle c_I \rangle$ is identified as the optimal abstract solution. Since no optimal primitive solution has been found, the PDT is expanded with the optimal abstract solution, resulting in the first three layers of the PDT in Figure 1.

In the second iteration, after assigning costs to the leaves, the algorithm fails again to find an optimal primitive solution. It then searches in the relaxed PDT, which contains two abstract solutions: one with the task network $\langle a_0, c_1 \rangle$ and another with $\langle c_1, a_1, c_2 \rangle$. Based on the TDG in Figure 2, the cost of $\langle a_0, c_1 \rangle$ is 2, while $\langle c_1, a_1, c_2 \rangle$ has a cost of

3. Since no primitive solution has been found, the algorithm identifies and expands the PDT with the optimal abstract solution which has a cost of 2, expanding the second node in the third layer of Figure 1, labelled with the abstract task $c_1$.

In the third iteration, the algorithm identifies an optimal primitive solution with the task network $\langle a_0, a_2 \rangle$ and a cost of 2. In the relaxed PDT, there are two abstract solutions: one with the task network $\langle a_0, a_2 \rangle$ (as a primitive solution is also an abstract solution) and another with the task network $\langle c_1, a_1, c_2 \rangle$. Here the optimal abstract solution is the same as the optimal primitive solution, meaning there are no abstract solutions in the PDT with a lower cost than the primitive solution found. Therefore, the algorithm terminates and returns this primitive solution as the optimal solution.

If $\langle a_0, a_2 \rangle$ is not executable (e.g., due to unsatisfied preconditions for $a_2$), and given that no other primitive solutions exist in this PDT, the algorithm would expand the PDT using the optimal abstract solution in this PDT (i.e., the solution with the task network $\langle c_1, a_1, c_2 \rangle$), resulting in the full PDT shown in Figure 1. In this PDT, the optimal primitive solution would be the solution with the task network $\langle a_2, a_1, a_3 \rangle$. Since there are no abstract solution with a lower cost than this primitive solution, it would be returned as the optimal solution for this problem.

## Searching for an Optimal Solution in the PDT

To search for an optimal solution in a PDT, SibylSatOpt follows a similar approach to other SAT-based methods by encoding the PDT into a SAT formula, which is satisfiable if and only if a solution exists within the PDT. The encoding process is flexible, as various encoding techniques have been proposed to transform a PDT (or similar isomorphic structures) into a SAT formula for finding a solution (e.g., (Schreiber et al. 2019; Schreiber 2021; Behnke, Höller, and Biundo 2018)), and any valid encoding can be used. In this case, we use the Lilotane encoding, following the method from SibylSat, to transform the PDT into a SAT formula (Quenard, Pellier, and Fiorino 2024).

However, using this encoding alone would allow the solver to return any solution within the current PDT, not necessarily the optimal one. To ensure that the solver returns the optimal solution, we employ a Weighted Partial MaxSAT (WPMS) solver instead of a regular SAT solver. MaxSAT extends the SAT problem into an optimization problem, where the goal is to find an assignment that maximizes the number of satisfied clauses. WPMS further generalizes MaxSAT by introducing two key features: hard and soft clauses. Hard clauses must be satisfied for an assignment to be valid, while soft clauses may be violated at a certain cost, represented by weights. Solving a WPMS instance involves finding an assignment that satisfies all hard clauses while maximizing the total weight of satisfied soft clauses.

In our case, the rules proposed to search for a solution in the PDT are encoded as hard clauses, ensuring that any valid solution has a corresponding satisfiable assignment. To constrain the solver to only return the optimal solution within the PDT, we introduce additional soft clauses that incorporate our admissible heuristic for each DT, guiding the solver toward the lowest-cost solution.

For every leaf node $l$ in the PDT, labelled by task $t$, we add the following soft clause:

$$(\neg t_l, h(t)) \tag{1}$$

where $t_l$ is the Boolean variable associated with the task $t$ of the leaf node $l$, and $h(t)$ is the weight assigned to satisfying this clause, which corresponds to the TDG heuristic value of $t$. The variable is negated because we want the solver to minimize the cost of the solution. The hard clauses ensure that for every possible solution, the leaves in the PDT that are not part of the solution's task network are set to false. This negation enables finding the lowest cost solution: if the total weight of all leaves is $s$ and the valid solution has a cost $k$, the soft clauses ensure that the MaxSAT cost of this solution becomes $s - k$. Since the MaxSat solver aims to maximize the total weight of satisfied soft clauses, it will work to minimize $k$, effectively finding the lowest-cost solution.

## Relaxing and Searching for an Optimal Abstract Solution in the PDT

To find the optimal abstract solution in a PDT, SibylSatOpt applies the relaxation procedure of the PDT proposed by SibylSat (Quenard, Pellier, and Fiorino 2024), in which abstract task leaves are converted into primitive tasks with relevant preconditions and effects. The preconditions encoded for an abstract task are the set of propositions that must be true before any refinement of this abstract task can be executed, referred to as *mandatory preconditions* (Olz, Biundo, and Bercher 2021). For the effects, the *possible* positive (respectively, negative) *effects* of an abstract task are considered (Olz, Biundo, and Bercher 2021), that is, the sets of propositions that may be added (respectively, deleted) by any refinement of the task. The post-execution state of an abstract task is defined as leading to a nondeterministic state, determined by subtracting any subset of *poss-effect*$^-(t)$ and adding any subset of *poss-effect*$^+(t)$ to the pre-execution state. This is an over-approximation of the actual post-execution state that any specific refinement of an abstract task can produce, as it includes possible effects from various refinements.

As an illustrative example, suppose that in Figure 1, $a_1$ has preconditions $\{p, q\}$ and effects $\{\neg p, \neg q\}$, and $a_3$ has preconditions $\{p\}$ and effects $\{q\}$. Then, the mandatory preconditions of $c_2$ would be $\{p\}$, since $p$ is required by every possible refinement of $c_2$. The possible effects would be $\{\neg p, \neg q, q\}$, as they cover all effects that might be produced by refining $c_2$. Thus, any refinement of $c_2$ satisfies the inferred mandatory preconditions, and its effects are a subset of the inferred possible effects.

Once the PDT has been relaxed, its encoding is identical to the encoding of the classic PDT, with the only difference being that the relaxed effects of an abstract task leaf are nondeterministic, and as such, cannot be encoded in the same way as the effects of a primitive task. As in the SibylSat paper, we employ the frame axioms proposed by the Lilotane paper (Schreiber 2021) to encode these effects.

As with the non-relaxed PDT, finding a solution in the relaxed PDT without soft clauses yields any abstract solution within the relaxed PDT. Therefore, we must include the

same soft clauses as in Equation 1 to ensure that the solver returns the optimal abstract solution.

Searching for an abstract solution in this relaxed PDT serves as a heuristic method for identifying the $predecessorDTs$ of any solution (i.e., finding all DTs that could be refined into a solution). The careful inference of preconditions and effects ensures that, for any solution, all of its $predecessorDTs$ are identified as abstract solutions by the planner. This is a property critical for the planner's completeness, as formalized in the following theorem.

**Theorem 2.** *Let $P = (L, C, M, M, c_I, s_I, g)$ be a planning problem and $T_{sol}$ any solution for this problem. Any $T'$ in $predecessorDTs(T_{sol})$ is an abstract solution.*

*Proof sketch.* Given a solution $T_{sol}$ with the task network $\pi$, consider any DT $T' \in predecessorDTs(T_{sol})$ with the task network $\pi'$. By definition, the root task of $T'$ is the same as $T_{sol}$ and is $c_I$. For each abstract task $t$ in $\pi'$, we select its effects to exactly match the actual effects from its full decomposition in $\pi$. This choice is valid since the possible effects for an abstract task are an over-approximation of the effects that any specific refinement of this task can produce, allowing us to select any of these as the effects for the abstract task. This alignment ensures that executing $\pi'$ from $s_I$ replicates the state transitions in $\pi$. The preconditions and inferred mandatory preconditions of each task in $\pi'$ are satisfied because they were satisfied in $\pi$, and our chosen effects maintain the identical state for subsequent tasks. Therefore, $\pi'$ leads to the goal $g$ just as $\pi$ does. As such, $T'$ satisfies all criteria of a solution, making it an abstract solution. □

## Proof of Optimality

The optimality of SibylSatOpt stems from two key properties: first, its use of an admissible heuristic (the TDG-based heuristic extended to DTs) that never overestimates the cost to reach a solution from any DT; second, its strategy of expanding abstract solutions with lower costs before accepting any primitive solution as optimal. Together, these properties ensure that no better solution can exist when the algorithm terminates, as formalized in the following theorem.

**Theorem 3.** *If SibylSatOpt returns a solution, then it is optimal with respect to action cost.*

*Proof sketch.* Assume, for contradiction, that SibylSatOpt returns a solution that is not optimal. Let $T_{sol}$ be the solution returned by the algorithm, and let $T^*$ be an optimal solution with $h(T^*) < h(T_{sol})$. Since $T^*$ derives from the initial task $c_I$, and by construction of the PDT, there exists a predecessor DT $T'$ of $T^*$ present in the current PDT when $T_{sol}$ is accepted as optimal, such that all pending leaves of $T'$ are undeveloped. By Theorem 1, we have $h(T') \leq h(T^*) < h(T_{sol})$. By Theorem 2, $T'$ is an abstract solution. According to the algorithm, SibylSatOpt should not accept $T_{sol}$ as optimal while an abstract solution $T'$ with a lower heuristic cost exists. Therefore, $T_{sol}$ would not have been returned, contradicting our assumption. Hence, if SibylSatOpt returns a solution, it is optimal with respect to action cost. □

While Theorem 3 guarantees that SibylSatOpt returns only optimal solutions, it does not ensure termination. If the TDG is acyclic, the search space is bounded and termination is guaranteed. When cycles are present, SibylSatOpt does not diverge as long as each occurrence of a cycle in the solution increases its cost. For instance, in Figure 2, the cycle formed by $c_2$ and $m_3$ cannot cause divergence, as its presence in a plan necessarily increases the cost. An abstract plan $\langle a_2, a_1, c_2 \rangle$ of cost 3 can be refined either into the primitive plan $\langle a_2, a_1, a_3 \rangle$ (cost 3) or into an extended abstract plan $\langle a_2, a_1, a_1, c_2 \rangle$ (cost 4). As the cost rises with each cycle, the planner eventually favors the optimal solution.

A special case, however, may cause issues: *empty cycles*, defined by Behnke, Höller, and Biundo (2019) as a non-empty sequence of decompositions for an abstract task $A$ that results in a task network containing only $A$ itself. These cycles neither achieve progress nor increase cost. Such cycles can be detected during preprocessing. To prevent their complete realization in a solution, we can add specific SAT constraints for each empty cycle $C$ in the PDT as follows:

$$\bigvee_{t_l \in C} \neg t_l$$

where each $t_l$ is the Boolean variable associated with a task $t$ in the cycle. This ensures that at least one task in the cycle is false, preventing solutions from containing empty cycles.

## Handling the Lifted Case

The procedure in Algorithm 1 is applicable only to the ground case, where the PDT contains ground tasks and methods. However, SibylSatOpt uses the Lilotane encoding, which transforms a lifted PDT into a SAT formula (Schreiber 2021). As a result, the TDG costs cannot be directly assigned to the leaves, as the TDG costs are associated with ground tasks and methods, while the PDT in SibylSatOpt labels the nodes with lifted tasks and methods. So, when dealing with a lifted PDT, we set the heuristic value of a lifted task as the minimum value over all possible groundings of that task, as proposed by Bercher et al. (2017), which preserves admissibility with respect to action costs.

## Evaluation

### Planners and Settings

We evaluated SibylSatOpt against state-of-the-art optimal TOHTN planners. The evaluation involved four planners: *SibylSatOpt*[1] linked with the UwrMaxSat solver (Piotrów 2020); a SAT-based planner by Behnke, Höller, and Biundo (2019); and two progression-based search planners, *PandaDealer*$^{P\text{-}TODR^{lp}}$ and *PandaDealer*$^{RC(lmc)}$. For simplicity, we will refer to the SAT planner as SAT-BIN, and the latter two planners as P-TODR and RC(lmc), respectively.

SAT-BIN proposes an encoding to verify, for a given plan of length $l$, that no solution exists with a cost lower than $l$. After computing an initial upper and lower bound on the optimal cost, a binary search using this encoding is performed to find the optimal plan.

[1] https://github.com/gaspard-quenard/sibylsat

Both P-TODR and RC(lmc) employ progression-based search with configurations including $A^\star$ with loop detection (Höller and Behnke 2021) and dead-end analysis with lookaheads and early refinements (Olz and Bercher 2023; Olz, Höller, and Bercher 2023), representing the best configurations of the *PANDA*$_\pi$ system for optimal planning according to the results of the 2023 TOHTN IPC (Olz, Höller, and Bercher 2023). P-TODR employs an ILP-based HTN planning heuristic specifically designed for TOHTN domains (Olz, Lodemann, and Bercher 2024). We use the variant introduced in their paper, where the ILP model is relaxed to an LP model, as it yields improved results. RC(lmc) uses the relaxed composition heuristics (RC) (Höller et al. 2018, 2019, 2020) in combination with the classical admissible LM-cut heuristics (Helmert and Domshlak 2009) .

The experiments were conducted on a system with an Intel Core i7-12700H CPU and 32GB of RAM. Each problem instance had a maximum runtime of 10 minutes. The evaluations covered the 24 benchmarks of the IPC 2020 and the two additional ones (Lamps and SharpSAT) from the IPC 2023. The planners were evaluated using two classical metrics: *coverage* (number of solved problems) and *IPC scores*, computed as $\min\left(1, 1 - \frac{\log(t)}{\log(600)}\right)$, where $t$ is the time to solve the problem.

## Results

| Domain | | SibylSatOpt | | RC(lmc) | | P-TODR$^{lp}$ | | SAT-BIN | |
|---|---|---|---|---|---|---|---|---|---|
| | | **Cov** | **IPC** | **Cov** | **IPC** | **Cov** | **IPC** | **Cov** | **IPC** |
| Assembly | 30 | **5** | **4.2** | 4 | 3.5 | 3 | 2.4 | 4 | 2.3 |
| Barman-BDI | 20 | **16** | **14.7** | 10 | 7.0 | 8 | 4.3 | 5 | 1.7 |
| Blocksw-G | 30 | **28** | **24.7** | 25 | 20.9 | 25 | 22.8 | 2 | 0.7 |
| Blocksw-H | 30 | 0 | 0.0 | 5 | **3.8** | **6** | 3.7 | 2 | 0.9 |
| Childsnack | 30 | **25** | **21.6** | 0 | 0.0 | 8 | 2.5 | 6 | 1.1 |
| Depots | 30 | **26** | **23.8** | 18 | 17.2 | 18 | 16.6 | 3 | 1.5 |
| Elevator-L | 147 | **147** | **141.1** | 92 | 51.9 | 77 | 41.5 | 2 | 0.9 |
| Entertain. | 12 | **5** | 4.9 | **5** | **5.0** | **5** | **5.0** | **5** | 3.5 |
| Factories-S | 20 | **6** | 4.7 | **6** | **4.8** | **6** | 4.4 | 1 | 0.5 |
| Freecell-L | 60 | **0** | **0.0** | **0** | **0.0** | **0** | **0.0** | **0** | **0.0** |
| Hiking | 30 | **19** | **12.1** | 7 | 2.2 | 9 | 2.9 | 0 | 0.0 |
| Lamps | 30 | 13 | 12.5 | **15** | **13.7** | 14 | 12.8 | 10 | 6.9 |
| Logistics-L | 80 | **58** | **36.8** | 25 | 20.6 | 23 | 19.3 | 0 | 0.0 |
| Minecraft-P | 20 | **4** | **1.0** | 1 | 0.6 | 1 | 0.3 | 0 | 0.0 |
| Minecraft-R | 59 | 28 | 14.7 | 35 | 21.4 | **38** | **23.9** | 1 | 0.2 |
| Monroe-FO | 20 | **20** | **14.1** | 19 | 9.2 | 0 | 0.0 | 0 | 0.0 |
| Monroe-PO | 20 | **16** | **8.1** | 12 | 4.0 | 0 | 0.0 | 0 | 0.0 |
| Multiarm-B | 74 | 2 | 1.5 | 11 | 9.3 | **16** | **10.7** | 5 | 1.8 |
| Robot | 20 | **11** | 10.8 | **11** | **11.0** | **11** | 10.7 | **11** | 8.4 |
| Rover-G | 30 | **21** | **15.2** | 8 | 6.8 | 8 | 7.1 | 4 | 2.6 |
| Satellite-G | 20 | **19** | **13.4** | 6 | 4.4 | 10 | 6.3 | 3 | 2.3 |
| SharpSAT | 21 | **9** | **8.4** | **9** | 7.5 | 8 | 6.4 | 0 | 0.0 |
| Snake | 20 | **20** | **18.4** | **20** | 16.1 | 6 | 3.8 | 4 | 2.5 |
| Towers | 20 | 12 | 9.9 | **13** | **10.2** | 9 | 6.2 | 6 | 4.0 |
| Transport | 40 | **31** | **28.5** | 10 | 6.1 | 15 | 12.4 | 14 | 6.9 |
| Woodwork | 30 | **30** | **25.4** | 17 | 15.6 | 15 | 13.5 | 17 | 11.3 |
| Overall | 943 | **571** | **470.5** | 384 | 272.6 | 339 | 239.6 | 105 | 60.0 |
| Normalized | 26 | **15.6** | **12.6** | 11.0 | 8.2 | 8.9 | 6.7 | 4.0 | 2.4 |

Table 1: Coverage and IPC scores for optimal planning on the IPC 2020 and IPC 2023 benchmarks.

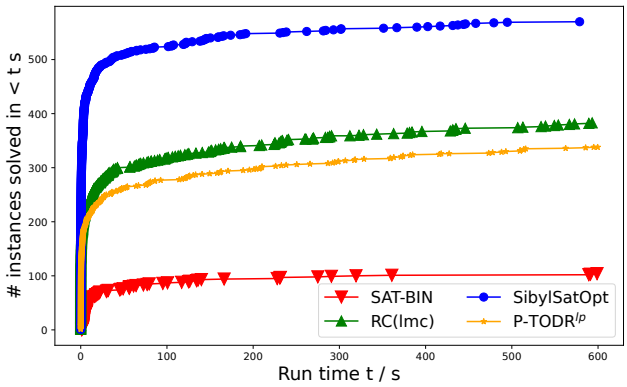


Figure 3: Cumulative number of instances solved per planner over time.

The results for coverage and IPC scores across the benchmarks are summarized in Table 1. SibylSatOpt shows a significant advantage over other state-of-the-art optimal planners. In total, SibylSatOpt optimally solved 571 instances, compared to 384 solved by RC(lmc), 339 by P-TODR, and 105 by SAT-BIN. We also analyze the speed with which each planner reaches solutions. Notably, while all planners plateau after 10 minutes, SibylSatOpt solves most instances much faster. As shown in Figure 3, SibylSatOpt was able to solve a large portion of instances within the first few seconds, reaching 422 solutions in less than 5 seconds (approximately 74% of its total). In contrast, RC(lmc) found 215 solutions (56%), P-TODR found 192 (57%), and SAT-BIN found 23 (22%) within the same timeframe.

We now turn to domain-specific results. For comparison, a planner is said to "dominate" another if it solves at least three more instances in a given domain. Using this criterion, SibylSatOpt dominates both RC(lmc) and P-TODR in 13 domains, and SAT-BIN in nearly all domains. However, SAT-BIN dominates SibylSatOpt in one domain (Multiarm-Blocksworld), where it solves three more instances. SibylSatOpt is also dominated by both RC(lmc) and P-TODR in three domains (Blocksworld-HPDDL, Minecraft-Regular, and Multiarm Blocksworld). In the remaining domains, differences are smaller, with no clear dominance between SibylSatOpt and at least one competitor.

## Conclusion

We introduced SibylSatOpt, a novel MaxSAT-based greedy optimal search approach for TOHTN planning. By integrating an admissible heuristic that combines a relaxed MaxSAT encoding of the problem with the TDG heuristic, SibylSatOpt efficiently guides the search toward optimal solutions. Experimental results on IPC benchmarks demonstrate that SibylSatOpt significantly outperforms existing state-of-the-art optimal TOHTN planners in both runtime and problem coverage. This work highlights the potential of combining greedy search with HTN heuristics in SAT-based planning. Future work will explore the use of other HTN heuristics to further enhance the planner's performance.

## Acknowledgments

This work has been partially supported by MIAI@Grenoble Alpes, (ANR-19-P3IA-0003).

## References

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. 2016. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, 20–28.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *IJCAI*, volume 9, 1629–1634.

Behnke, G. 2021. Block compression and invariant pruning for SAT-based totally-ordered HTN planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 25–35.

Behnke, G.; Höller, D.; and Bercher, P., eds. 2021. *Proceedings of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT-Totally-ordered hierarchical planning through SAT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

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

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On succinct groundings of HTN planning problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9775–9784.

Behnke, G.; Pollitt, F.; Höller, D.; Bercher, P.; and Alford, R. 2022. Making translations to classical planning competitive with other HTN planners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 9687–9697.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An Admissible HTN Planning Heuristic. In *IJCAI*, 480–488.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 1763–1769.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Aips*, volume 94, 249–254.

Geier, T.; and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, 1955.

Helmert, M.; and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 19, 162–169.

Höller, D.; and Behnke, G. 2021. Loop Detection in the PANDA Planning System. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 168–173.

Höller, D.; and Bercher, P. 2021. Landmark generation in HTN planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11826–11834.

Höller, D.; Bercher, P.; and Behnke, G. 2020. Delete-and Ordering-Relaxation Heuristics for HTN Planning. In *IJCAI*, 4076–4083.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018. A generic method to guide HTN progression search with classical heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 114–122.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2019. On Guiding Search in HTN Planning with Classical Planning Heuristics. In *IJCAI*, 6171–6175.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020. HTN planning as heuristic progression search. *Journal of Artificial Intelligence Research*, 67: 835–880.

Olz, C.; and Bercher, P. 2023. A Look-Ahead Technique for Search-Based HTN Planning: Reducing the Branching Factor by Identifying Inevitable Task Refinements. In *Proceedings of the International Symposium on Combinatorial Search*, volume 16, 65–73.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing hidden preconditions and effects of compound HTN planning tasks–a complexity analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11903–11912.

Olz, C.; Höller, D.; and Bercher, P. 2023. The PANDADealer System for Totally Ordered HTN Planning in the 2023 IPC. *sat*, 1(3): 14–89.

Olz, C.; Lodemann, A.; and Bercher, P. 2024. A Heuristic for Optimal Total-Order HTN Planning Based on Integer Linear Programming. In *27th European Conference on Artificial Intelligence*, volume 392, 4303–4310.

Piotrów, M. 2020. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 132–136. IEEE.

Quenard, G.; Pellier, D.; and Fiorino, H. 2024. SibylSat: Using SAT as an Oracle to Perform a Greedy Search on TO-HTN Planning. In *27th European Conference on Artificial Intelligence*, volume 392, 4157–4164.

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.; et al. 2019. Tree-REX: SAT-based tree exploration for efficient and high-quality HTN planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 382–390.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; et al. 2024. The 2023 International Planning Competition.