35th International Conference on
Automated Planning and Scheduling

November 9–14, 2025, Melbourne, Victoria, Australia

HPlan 2025

Proceedings of the 8th ICAPS Workshop on

Hierarchical Planning

Program Committee

Ron Alford	MITRE
Gregor Behnke	University of Amsterdam
Pascal Bercher	the Australian National University (ANU)
Maurice Dekker	University of Amsterdam
Humbert Fiorino	Université Grenoble Alpes (UGA)
Birte Glimm	Ulm University
Daniel Höller	Saarland University
Ugur Kuter	Smart Information Flow Technologies (SIFT)
Alexander Lodemann	Ulm University
Mario Schmautz	independent researcher
Mauro Vallati	University of Huddersfield
Michael Welt	Ulm University
Mohammad Yousefi	the Australian National University (ANU)
Yifan Zhang	the Australian National University (ANU)

Organizing Committee

Pascal Bercher	The Australian National University
Mauro Vallati	University of Huddersfield
Conny Olz	Independent Researcher
Ron Alford	The MITRE Corporation

Preface

The motivation for using hierarchical planning formalisms is manifold. It ranges from an explicit and predefined guidance of the plan generation process and the ability to represent complex problem solving and behavior patterns to the option of having different abstraction layers when communicating with a human user or when planning cooperatively. This led to numerous hierarchical formalisms and systems. Hierarchies induce fundamental differences from classical, non-hierarchical planning, creating distinct computational properties and requiring separate algorithms for plan generation, plan verification, plan repair, and practical applications. Many techniques required to tackle these – or further – problems in hierarchical planning are still unexplored.

With this workshop, we bring together scientists working on many aspects of hierarchical planning to exchange ideas and foster cooperation.

2025 is the 8$^{\text{th}}$ edition of HPlan, which was hosted by the 35$^{\text{th}}$ ICAPS conference, taking place in Melbourne, Australia. In this year, we received 5 submissions, of which one got simultaneously submitted and accepted at another conference.

The accepted papers cover a broad spectrum of current research in hierarchical planning, ranging from new theoretical insights and complexity analyses to advances in modeling support and learning. Contributions include probabilistic extensions of hierarchical goal networks, formal analyses of multi-level abstraction in HTN planning, LLM-based frameworks for generating hierarchical models, expressive PSPACE-complete planning formalisms that bridge unordered HTNs with numerical and logical constraints, and one paper on a parsing-based Task Insertion HTN planner.

The workshop took place for half a day on Monday morning, November 10, 2025. It featured an invited talk by Roman Barták, who presented recent results on hierarchical plan validation and correction and their connections to plan recognition, plan repair, and autonomous agents capable of maintaining their own planning models. The talk description can be found further down in these proceedings.

The accepted papers were presented as short teaser talks, followed by an extended poster session, encouraging in-depth discussions and exchange among participants.

In line with previous years, we again encouraged presentations from other venues. This year, one author used the opportunity and presented his work from the ICAPS workshop on Bridging the Gap Between AI Planning and (Reinforcement) Learning that describes a learned heuristic function for HTN planning.

As always these proceedings and its papers are non-archival.

Pascal, Mauro, Conny, and Ron
HPlan Workshop Organizers,
December 2025

Invited Talk

This year, our invited speaker was *Roman Barták*. He made contributions to the field for many years now, mostly in using and refining parsing techniques for HTN plan verification and correction.

From Validation to Correction: Towards Autonomous Agents Based on Hierarchical Planning

Hierarchical plan validation gets a sequence of actions and determines if it is a valid hierarchical plan, that is, it is executable and can be obtained by decomposing some task. If the plan is valid, then a proof in the form of the task decomposition structure can be released. However, if the plan is not valid, plan validators simply state this. This is where plan correction may help by suggesting modifications to invalid plans, such as inserting or deleting actions, to make them valid hierarchical plans. Although the original motivation for plan correction was to enhance plan validation, the concept proved to apply to various other problems, including plan recognition, plan repair, and even planning itself.
The talk illustrates a journey from plan validation to plan correction, advocating plan correction as a general concept that encompasses various hierarchical planning tasks. As a future research direction, it suggests that a similar approach can be applied to planning domain models to obtain truly autonomous planning agents capable of learning and maintaining models of own behavior.

Bio

Roman Barták, Professor of Computer Science
Charles University
Faculty of Mathematics and Physics

Roman Barták is a professor of computer science at Charles University, Prague. He works in the area of Artificial Intelligence with particular emphasis on automated planning and scheduling, constraint satisfaction, and knowledge representation. His research goal is to design intelligent autonomous agents (e.g. robots) that can plan their activities, do decisions, and act in real environment. He put strong emphasis on model-based approaches to problem-solving as a means to achieve trustworthy and explainable AI. Recently, his research has focused on two areas: multi-agent path finding and hierarchical planning.

Table of Contents

Hierarchical Goal Networks for Probabilistic Planning: Preliminary Results

David H. Chan[1,3], **Mark Roberts**[3], **Dana S. Nau**[1,2]

[1]Department of Computer Science and [2]Institute for Systems Research, University of Maryland, College Park, MD, USA
[3]Navy Center for Applied Research in Artificial Intelligence, U.S. Naval Research Laboratory, Washington, DC, USA
dhchan@cs.umd.edu, mark.c.roberts20.civ@us.navy.mil, nau@umd.edu

Abstract

Hierarchical goal networks (HGNs) provide a framework for goal-directed planning by decomposing high-level goals into ordered subgoals. While effective in deterministic settings, HGN planning has not been extended to handle stochastic domains. We introduce a new formalism for probabilistic HGN planning with task-insertion semantics, enabling probabilistic planners to incorporate domain knowledge from goal decomposition methods. This formalism retains the efficiency and scalability of classical HGN planning while supporting probabilistic reasoning and online search. We present PHGN-UCT, a UCT-based online planner that leverages hierarchical methods when available and falls back on UCT search when they are not, enabling planning under partial domain models. Preliminary experiments demonstrate that PHGN-UCT can effectively exploit even incomplete domain knowledge to outperform standard UCT planning, suggesting that probabilistic HGN methods are an effective way to incorporate domain knowledge in probabilistic planning.

1 Introduction

Hierarchical planning formalisms introduce a task hierarchy to allow rich domain-specific guidance and facilitate reasoning at multiple levels of abstraction (Ghallab, Nau, and Traverso 2016, 2025). Hierarchical task network (HTN) planning achieves this through methods which recursively break down compound tasks into sub-tasks until a sequence of primitive actions is identified. However, HTN planning suffers from several key limitations, including the difficulty of designing domain-independent heuristics and the need for a complete set of expert-defined methods. Hierarchical goal networks (HGNs) address these challenges by decomposing goals—rather than tasks—into subgoals (Shivashankar et al. 2012, 2013). This enables the ability to specify goals declaratively and leverage reasoning techniques from classical planning to augment goal decomposition, all while maintaining the expressivity of HTN planning (Alford et al. 2016).

Real-world domains often involve uncertainty, where actions can produce nondeterministic outcomes (Patra et al. 2020, 2021). Fully observable nondeterministic (FOND) models capture such behavior, and recent work has extended HTN planning to operate within the FOND framework (Chen and Bercher 2021, 2022; Yousefi and Bercher 2024). However, many of the properties of HGN planning that make it an attractive formalism for deterministic planning also extend to the probabilistic setting. In particular, the goal-based structure of HGNs enables planners to reason over ordered sequences of goals, a form of temporally extended goal. This makes HGNs especially well-suited for integration with probabilistic planners that rely on online search, where flexibility and adaptability to dynamic environments and stochastic action outcomes are critical, and strict procedural task sequences may be too brittle. Moreover, task-insertion semantics in HGN planning allows executing actions that do not immediately contribute to a goal, thus supporting the interleaving of goal decomposition and heuristic-guided action selection. This provides a principled basis for planning approaches that integrate hierarchical domain knowledge with search.

In this paper, we introduce probabilistic HGN planning, a new formalism that extends HGN planning to probabilistic domains—analogous to how FOND HTN planning extends classical HTN planning. To leverage the new formalism, we also develop an online probabilistic planning algorithm, PHGN-UCT, which embeds the hierarchical guidance and domain knowledge of HGNs into UCT search. We adopt task-insertion semantics, which provides a crucial mechanism for planning with incomplete domain models. The planner takes a hybrid approach that can leverage methods when available, but will otherwise fall back on standard search.

We present preliminary results on a hand-crafted domain suggesting that PHGN-UCT can effectively leverage hierarchical domain knowledge. In our tests, providing more methods consistently improves performance over a standard UCT planner with no methods, indicating that even imperfect or incomplete domain knowledge can significantly accelerate probabilistic planning compared to standard UCT. In summary, our contributions are a novel probabilistic HGN planning formalism and a UCT-based algorithm, demonstrating the benefit of combining hierarchical structure and domain knowledge with search in stochastic domains.

2 Probabilistic HGN Planning

Following the definitions of partial-order classical HGNs from Alford et al. (2016), let $\mathcal{L}$ be a propositional language with a set atoms $\mathcal{X}$ and propositional formulae $\mathcal{F}$. A **goal network** is a tuple $gn = (I, \prec, \alpha)$, where I is a set of symbols which are indices, or labels, for goals, $\prec \subseteq I \times I$ is a partial order on I, and $\alpha : I \to \mathcal{F}$ maps each symbol in I to a goal for-

mula. This labeling of goals is necessary to differentiate multiple occurrences of the same goal in the goal network—the labels will be unique while the goal formulae they map to under α may be equivalent. We denote the set of *unconstrained goals* as $UC(gn) = \{i \in I \mid i \text{ has no } \prec\text{-predecessors}\}$.

Two goal-networks $(I, \prec, \alpha)$ and $(I', \prec', \alpha')$ are *isomorphic* if there exists a bijection $f : I \to I'$ such that $i \prec i' \iff f(i) \prec f(i')$ and $\alpha(i) \equiv \alpha'(f(i))$ for all $i \in I$. This isomorphism relation induces a quotient set of equivalence classes of all goal networks. Going forward, we fix $G = \{(I_1, \prec_1, \alpha_1), (I_2, \prec_2, \alpha_2), \dots\}$ to be a complete set of representatives of this quotient set of all goal networks such that for all $i \neq j$, $I_i \cap I_j = \varnothing$.

A **method** m is a tuple $(\mathrm{pre}(m), \mathrm{post}(m), \mathrm{sub}(m))$, where $\mathrm{pre}(m), \mathrm{post}(m) \in \mathcal{F}$ are the pre- and postcondition of m, respectively, and $\mathrm{sub}(m) = (I_m, \prec_m, \alpha_m) \in G$ is a goal network. A method is *relevant* to a subgoal $i \in I$ in a goal network $gn = (I, \prec, \alpha)$ if at least one literal in the negation-normal form (NNF) of $\mathrm{post}(m)$ matches a literal in the NNF of $\alpha(i)$. This ensures that at least part of $\alpha(i)$ is true by accomplishing $\mathrm{post}(m)$. To ensure that m accomplishes its own goal, we require that there exists $i \in I_m$ such that $\alpha_m(i) \equiv \mathrm{post}(m)$ and for all $j \in I_m$, $j \preceq i$.

A **probabilistic HGN planning domain** is a tuple $\mathcal{D} = (S, M, A, \gamma, \Pr)$, where

- $S \subseteq 2^{\mathcal{X}}$ is a finite set of states;
- $M \subseteq \mathcal{F} \times \mathcal{F} \times G$ is a finite set of methods;
- $A \subseteq \mathcal{F} \times 2^{2^{\mathcal{X}} \times 2^{\mathcal{X}}}$ is a finite set of nondeterministic actions $a = (\mathrm{pre}(a), \mathrm{eff}(a)) \in A$ where $\mathrm{pre}(a) \in \mathcal{F}$ is the precondition, and $\mathrm{eff}(a) \in 2^{2^{\mathcal{X}} \times 2^{\mathcal{X}}}$ is a set of pairs $(\mathrm{add}(a), \mathrm{del}(a))$ of add effects $\mathrm{add}(a) \subseteq \mathcal{X}$ and delete effects $\mathrm{del}(a) \subseteq \mathcal{X}$;
- $\gamma : S \times A \to 2^S$ is a transition function where $\gamma(s, a)$ is defined iff $s \models \mathrm{pre}(a)$, and $\gamma(s, a) = \{(s \backslash \mathrm{del}(a)) \cup \mathrm{add}(a) \mid (\mathrm{add}(a), \mathrm{del}(a)) \in \mathrm{eff}(a)\}$; and
- $\Pr(\cdot \mid s, a)$ is a probability distribution over $\gamma(s, a)$, where for all $s' \in \gamma(s, a)$, $\Pr(s' \mid s, a)$ is the probability of reaching state s' when action a is executed in state s.

An action or method u is *applicable* in state s if $s \models \mathrm{pre}(u)$.

A **node** is a pair $n = (s, gn)$, where $s \in S$ is a state, and $gn = (I, \prec, \alpha) \in G$ is a goal network. In classical HGN planning, there are three forms of node progression: action application, goal decomposition, and goal release. Analogous forms of node progression extend to probabilistic HGN planning as follows: Let (s, gn) be a node, where $gn = (I, \prec, \alpha)$.

- **action application:** Let $a \in A$ be an action applicable in s. Application of a probabilistic action a yields the node (s', gn) with probability $\Pr(s' \mid s, a)$, for all $s' \in \gamma(s, a)$. We denote this by $(s', gn) \sim P^a_{app}(s, gn)$. Note that under task-insertion semantics, actions need not be "relevant" to an unconstrained subgoal to be applied.
- **goal decomposition:** Let $i \in UC(gn)$, and let m be a method relevant to i and applicable in s, where $\mathrm{sub}(m) = gn_m = (I_m, \prec_m, \alpha_m)$. Goal decomposition by m prepends gn with gn_m to yield the node (s, gn'), where $gn' = (I \cup I_m, \prec \cup \prec_m \cup (I_m \times \{i\}), \alpha \cup \alpha_m)$. We denote this by $(s, gn') = P^{i,m}_{dec}(s, gn)$.
- **goal release:** Let $i \in UC(gn)$, and suppose $s \models \alpha(i)$. Release of subgoal i removes it from gn if it is satisfied in the current state, to yield the node (s, gn'), where $gn' = (I \backslash \{i\}, \{(i_1, i_2) \in \prec \mid i_1 \neq i)\}, \{(i', g) \in \alpha \mid i' \neq i\})$. We denote this by $(s, gn') = P^i_{rel}(s, gn)$.

This model of probabilistic HGN planning operates under task-insertion semantics, which allows an action to be applied whenever an action's preconditions are supported by the current state. This is an extension of the typical HGN planning semantics where actions must be motivated by the hierarchy, i.e. relevant to an unconstrained goal in the goal network, to be executed. This more flexible formalism is helpful in probabilistic settings and with incomplete domain models.

A **probabilistic HGN planning problem** is a tuple $\mathcal{P} = (\mathcal{D}, s_0, gn_0)$, where $\mathcal{D}$ is a probabilistic HGN planning domain, $s_0 \in S$ is the initial state, and gn_0 is the initial goal network. Note that the standard planning paradigm with a standalone goal g can be represented as a goal network $(\{\mathfrak{g}\}, \varnothing, \{(\mathfrak{g}, g)\})$, where $\mathfrak{g}$ is any symbol for g. We use $gn_\varnothing$ to denote the empty goal network $gn_\varnothing = (\varnothing, \varnothing, \varnothing)$.

A solution to a probabilistic HGN planning problem takes the form of an **action-method policy**, which is a partial mapping $\pi : S \times G \to A \cup M$ from a node to either an action $a \in A$ or a method $m \in M$. For the policy to be well-formed, we require that for all (s, gn) in the domain of π the following conditions hold: $gn = (I, \prec, \alpha) \neq gn_\varnothing$; if $\pi(s, gn) = a \in A$ then $s \models \mathrm{pre}(a)$; and if $\pi(s, gn) = m \in M$ then $s \models \mathrm{pre}(m)$ and m is relevant to some $i \in UC(gn)$.

To classify the different types of solutions to a probabilistic HGN planning problem, let $\mathcal{P} = (\mathcal{D}, s_0, gn_0)$ and let π be an action-method policy for $\mathcal{P}$. We define $\mathrm{Graph}(\pi, s_0, gn_0) = (V, E, p)$, where $V \subseteq S \times G$ is a set of nodes, $E \subseteq V \times V$ is a set of edges, and $p : E \to (0, 1]$ is a weight function. We construct $\mathrm{Graph}(\pi, s_0, gn_0)$ as follows:

- $(s_0, gn_0) \in V$
- If $(s, gn) \in V$ and $\pi(s, gn) = a \in A$, then for all $s' \in \gamma(s, a)$, $(s', gn) \in V$ and $e = ((s, gn), (s', gn)) \in E$, with $p(e) = \Pr(s' \mid s, a)$.
- If $(s, gn) \in V$, and $\pi(s, gn) = m \in M$ is relevant to $i \in UC(gn)$, then $(s, gn') \in V$ and $e = ((s, gn), (s, gn')) \in E$ with $p(e) = 1$, where $(s, gn') = P^{i,m}_{dec}(s, gn)$.
- If $(s, gn) \in V$, and there exists $i \in UC(gn)$ s.t. $s \models \alpha(i)$, then $(s, gn') \in V$ and $e = ((s, gn), (s, gn')) \in E$ with $p(e) = 1$, where $(s, gn') = P^i_{rel}(s, gn)$.

$\mathrm{Graph}(\pi, s_0, gn_0) = (V, E, p)$ defines the reachability graph, or "*execution structure*" (Yousefi and Bercher 2024), induced by policy π from initial node $n_0 = (s_0, gn_0)$, where E is a set of directed edges connecting nodes in V via node progression by π, and p are edge weights corresponding to the probability of the node progression. Note that goal decomposition and goal release are deterministic.

Let $n = (s, gn) \in V$. n is *terminal* if there are no outgoing edges from n, and n is a *goal node* if $gn = gn_\varnothing$; note that all goal nodes are necessarily terminal. A *history* σ from n is a finite sequence of nodes $\sigma = \langle n_0, n_1, \dots, n_h \rangle$ such that $n_0 = n$, $\forall j \in \{1, \dots, h\}, (n_{j-1}, n_j) \in E$, and n_h is terminal. We write $H(\pi, n)$ to denote the set of all histories of

Algorithm 1: A UCT planning algorithm for probabilistic HGN planning problems. $\mathcal{D} = (S, M, A, \gamma, \Pr)$ is a probabilistic HGN planning domain, s is the current state, $gn = (I, \prec, \alpha)$ is the current goal network, n_{ro} is the number of rollouts to perform at each step, d is the maximum rollout depth, $w : I \to \mathbb{R}^{\geq 0}$ is a weighting function, and $\Omega : S \to 2^M$ is the current map from states to the sets of methods that have been used in those states.

1: **procedure** PHGN-UCT($\mathcal{D}, s, gn, n_{ro}, d, w, \Omega$)
2: **if** $gn = gn_{\varnothing}$ **then return** success
3: **for all** $i \in UC(gn)$ **do**
4: **if** $s \models \alpha(i)$ **then**
5: $(s, gn) \leftarrow P^i_{rel}(s, gn)$
6: **return** PHGN-UCT($\mathcal{D}, s, gn, n_{ro}, d, w, \Omega$)
7: **for** n_{ro} times **do**
8: ROLLOUT($\mathcal{D}, s, gn, d, w, \Omega$) ▷ learn $Q_{\alpha(i)}$
9: $U \leftarrow \{m \in M \mid m$ is applicable in s, relevant to some $i \in UC(gn)$, and not in $\Omega(s)\}$
10: $U \leftarrow U \cup \{a \in A \mid a$ is applicable in $s\}$
11: **if** $U = \varnothing$ **then return** failure
12: $u \leftarrow \operatorname{argmax}_{u \in U} \sum_{i \in I} w(i) Q_{\alpha(i)}(s, u)$
13: **if** u is a method **then**
14: $\Omega(s) \leftarrow \Omega(s) \cup \{u\}$
15: **for all** $i \in UC(gn)$ s.t. u is relevant to i **do**
16: $(s, gn) \leftarrow P^{i,u}_{dec}(s, gn)$
17: **return** PHGN-UCT($\mathcal{D}, s, gn, n_{ro}, d, w, \Omega$)
18: **else** ▷ u is an action
19: $s' \leftarrow$ APPLY(s, u)
20: **return** PHGN-UCT($\mathcal{D}, s', gn, n_{ro}, d, w, \Omega$)

policy π from node n. The probability of history is defined as $\Pr(\sigma \mid \pi, n) = \prod_{j=1}^{|\sigma|} p(n_{j-1}, n_j)$. A policy π is a *solution policy* for $\mathcal{P} = (\mathcal{D}, s_0, gn_0)$ if $\mathrm{Graph}(\pi, s_0, gn_0)$ contains a goal node. A solution policy π is

- *safe* if for all nodes n in $\mathrm{Graph}(\pi, s_0, gn_0)$, there exists $\sigma \in H(\pi, n)$ that terminates at a goal node; or equivalently, $\sum_{\sigma \in H^*} \Pr(\sigma) = 1$, where $H^* = \{\sigma \in H(\pi, (s_0, gn_0)) \mid \sigma$ terminates at a goal node $\}$. Otherwise, π is *unsafe*.
- *cyclic safe* if π is safe and $\mathrm{Graph}(\pi, s_0, gn_0)$ contains a cycle; it is *acyclic safe* if it is safe and $\mathrm{Graph}(\pi, s_0, gn_0)$ contains no cycles.

3 Probabilistic HGN Planning Algorithm

We propose an online HGN-guided probabilistic planning algorithm based on Monte Carlo tree search (MCTS). MCTS is a general-purpose MDP search algorithm, well-known for its success in computer Go (Silver et al. 2016). It is well-suited to online probabilistic planning tasks (Keller and Eyerich 2012), where it incrementally builds a search tree using Monte Carlo rollouts to estimate the values of states. Many variants of MCTS exist (Gelly and Silver 2011; Feldman and Domshlak 2014; Painter, Lacerda, and Hawes 2020), but the traditional and most popular is Upper Confidence bounds applied to Trees (UCT) (Kocsis and Szepesvári 2006), which

Algorithm 2: The Monte Carlo rollout procedure for PHGN-UCT. $\mathcal{D}$, s, gn, w, and Ω are as defined in Algorithm 1. d is the remaining rollout depth, and $util$ is a strictly decreasing utility function over cost. Q_* and N_* are global maps with default value 0. ROLLOUT returns a map $\lambda : I \to \mathbb{R}^{\geq 0}$ where $\lambda(i)$ is the rollout cost for subgoal i.

1: **procedure** ROLLOUT($\mathcal{D}, s, gn, d, w, \Omega$)
2: **if** $gn = gn_{\varnothing}$ **then return** $\varnothing$
3: **for all** $i \in UC(gn)$ **do**
4: **if** $s \models \alpha(i)$ **then**
5: $(s, gn) \leftarrow P^i_{rel}(s, gn)$
6: **return** ROLLOUT($\mathcal{D}, s, gn, d, w, \Omega$) $\cup \{(i, 0)\}$
7: $U \leftarrow \{m \in M \mid m$ is applicable in s, relevant to some $i \in UC(gn)$, and not in $\Omega(s)\}$
8: $U \leftarrow U \cup \{a \in A \mid a$ is applicable in $s\}$
9: **if** $d = 0$ or $U = \varnothing$ **then return** $I \times \{d\}$
10: $u \leftarrow \operatorname{argmax}_{u \in U} \sum_{i \in I} w(i) \mathrm{UCB1}(Q_{\alpha(i)}, N_{\alpha(i)}, s, u)$
11: **if** u is a method **then**
12: $\Omega(s) \leftarrow \Omega(s) \cup \{u\}$
13: **for all** $i \in UC(gn)$ s.t. u is relevant to i **do**
14: $(s, gn) \leftarrow P^{i,u}_{dec}(s, gn)$
15: $\lambda \leftarrow$ ROLLOUT($\mathcal{D}, s, gn, d, w, \Omega$)
16: **else** ▷ u is an action
17: sample $(s', gn) \sim P^u_{app}(s, gn)$
18: $\lambda \leftarrow$ ROLLOUT($\mathcal{D}, s', gn, d - 1, w, \Omega$)
19: $\lambda \leftarrow \{(i, \lambda(i) + 1) \mid i \in I\}$
20: **for all** $i \in I$ **do**
21: $Q_{\alpha(i)}(s, u) \leftarrow \frac{N_{\alpha(i)}(s,u) Q_{\alpha(i)}(s,u) + util(\lambda(i))}{1 + N_{\alpha(i)}(s,u)}$
22: $N_{\alpha(i)}(s) \leftarrow N_{\alpha(i)}(s) + 1$
23: $N_{\alpha(i)}(s, u) \leftarrow N_{\alpha(i)}(s, u) + 1$
24: **return** λ

leverages principles from the multi-armed bandit problem to strike a balance between exploration and exploitation during planning (Auer, Cesa-Bianchi, and Fischer 2002). We choose UCT because its simplicity and well-understood behavior make it a suitable baseline for analyzing the impact of HGNs on probabilistic planning performance.

Our PHGN-UCT algorithm (Algorithm 1), is an online probabilistic planner which uses the rollout procedure shown in Algorithm 2. At each planning step, a fixed number of rollouts are performed (Lines 7–8), after which the most promising node progression is selected (Line 12) and applied. The resulting state is then observed, and the process repeats until a goal node is reached. Importantly, PHGN-UCT operates under our formalism of HGNs with task-insertion semantics, allowing any action applicable in the current state to be executed, regardless of its relevance to any subgoal.

PHGN-UCT makes several modifications to the standard UCT planning procedure. First, UCT must be adapted to select both actions and methods, mirroring the form of an action-method policy. At a node $n = (s, gn)$, UCT must search among all actions applicable in s, as well as all methods applicable in s and relevant to some $i \in UC(gn)$. To capture this, we extend the standard UCT action-value func-

tion Q to estimate values for both actions and methods. That is, $Q(s, u)$ is the estimated reward of progressing the current node n using u, where u is either an action (corresponding to action application P^u_{app}) or method (corresponding to goal decomposition $P^{i,u}_{dec}$). We also restrict the set of applicable methods to those which have not been used in s, precluding the same method from being used repeatedly in the same state. Whenever a subgoal $i \in UC(gn)$ is satisfied in s, it is immediately released from gn.

Additionally, to manage multiple goals in the goal network, PHGN-UCT learns a separate Q-function for each subgoal. That is, for all $g \in \alpha[I]$ (the image of I under α), $Q_g(s, u)$ estimates the expected reward of applying action or method u in state s with respect to g. It also maintains corresponding values for $N_{\alpha(i)}$, where $N_{\alpha(i)}(s, u)$ is the number of times u was selected in s, and $N_{\alpha(i)}(s)$ is the number of times s was visited. This contrasts with standard UCT planning, which learns a single Q-function aimed at the final planning goal.

To progress nodes, PHGN-UCT chooses the action or method that maximizes a weighted combination of these Q-values (Algorithm 1 Line 12). During rollouts, UCB1-values are used instead of Q-values, where

$$\mathrm{UCB1}(Q, N, s, u) = Q(s, u) + C\sqrt{\frac{\log(N(s))}{N(s,u)}} \qquad (1)$$

and C is the exploration constant (Algorithm 2 Line 10). The weighting is governed by a function $w : I \to \mathbb{R}^{\geq 0}$, which assigns the importance to each subgoal. A simple greedy implementation would define $w(i) = \mathbf{1}_{UC(gn)}(i)$, prioritizing immediate subgoals. However, such a strategy can be myopic, potentially leading to states that make future subgoals harder or impossible to achieve. More sophisticated weighting schemes that balance short-term and long-term goal achievement may yield better results, but we leave the exploration of such approaches to future work.

4 Preliminary Results

We ran some preliminary tests of PHGN-UCT on a small warehouse planning problem. In a grid world, a robot must move a box from a locked room to a target location by first retrieving a key. Three HGN methods were created for the domain, and our experiments varied the number of methods made available to the planner, from 0 methods (corresponding to standard UCT) to all 3. Our experiments only captured the special case of totally-ordered probabilistic HGNs, where in all goal networks $gn = (I, \prec, \alpha)$, $\prec$ is a total order. In this case, $UC(gn)$ is always a singleton set, so the weighting function $w = \mathbf{1}_{UC(gn)}(i)$ guides the planner to exclusively search for the next immediate subgoal.

We used a non-heuristic implementation of UCT with exploration constant $C = \sqrt{2}$, a maximum rollout depth $d = 20$, and a utility function $util : cost \mapsto \exp(-cost)$. The number of rollouts was varied in $n_{ro} \in \{5, 10, 20, 50, \ldots, 10000\}$. The cost of a run of PHGN-UCT is measured as the number of actions executed (Algorithm 1 Line 19). A maximum cost budget of 100 was imposed. Table 1 reports the average costs of running PHGN-UCT across 100 trials on the warehouse problem, varying the number of methods provided to the planner and the number of rollouts performed. When no methods are provided, PHGN-UCT reduces to standard UCT, which serves as the baseline. Runs which exceeded the cost budget of 100 were halted, and contributed 100 to the average.

Table 1: Average cost $\overline{x}$ (lower is better) and standard deviation σ of runs of the PHGN-UCT algorithm in the warehouse planning problem over 100 trials.

	0 methods		1 method		2 methods		3 methods	
n_{ro}	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
5	100.0	0.0	41.6	18.7	13.1	1.6	13.4	1.7
10	100.0	0.0	29.7	10.4	13.8	1.8	13.8	1.6
20	100.0	0.0	21.1	4.8	13.7	1.5	13.3	1.4
50	100.0	0.0	15.5	2.5	12.9	1.3	12.8	1.3
100	100.0	0.0	13.3	1.3	12.8	1.5	12.5	1.2
200	100.0	0.0	12.9	1.3	13.2	1.6	12.5	1.3
500	100.0	0.0	13.2	1.6	12.7	1.5	12.8	1.2
1000	100.0	0.0	13.0	1.4	13.0	1.4	12.7	1.5
2000	96.6	14.2	12.8	1.2	12.9	1.5	12.6	1.4
5000	68.9	36.1	13.1	1.3	12.9	1.5	12.4	1.2
10000	65.6	40.0	12.7	1.1	12.6	1.3	12.3	1.0

Standard UCT without any HGN methods provided struggles to find a solution in the warehouse planning problem. However, even a single HGN method boosts performance significantly, immediately outperforming standard UCT at 10000 rollouts while only using 5 rollouts, and converging to a near-optimal solution at only 100 rollouts per step. Performance consistently improves as more methods are provided, demonstrating that PHGN-UCT can effectively leverage hierarchical domain knowledge—even when it is incomplete—to achieve significantly better performance than standard UCT. While preliminary, these results suggest that HGNs may be useful for capturing domain knowledge for probabilistic planning, and future work should characterize the kinds of problems where these results hold.

5 Conclusion and Future Work

We introduced a new formalism for partial-order, probabilistic HGN planning with task-insertion semantics. This formalism preserves many of the key benefits of classical HGN planning while extending its applicability to stochastic domains. To leverage this formalism, we developed PHGN-UCT, a UCT-based planner that dynamically interleaves method-based decomposition and action-level search. Our approach balances structured hierarchical guidance with adaptive online search, enabling effective planning even in the presence of incomplete or imperfect domain models.

Preliminary results suggest PHGN-UCT effectively utilizes domain knowledge captured in HGN methods, but future work should expand the empirical evaluation across a broader range of benchmarks. Additionally, future work should refine the greedy weight function that prioritizes immediate goals, exploring strategies that more effectively trade off short- and long-term goals. Finally, a formal analysis of the expressiveness and complexity of probabilistic HGN planning, particularly in relation to FOND HTN planning (Chen

and Bercher 2022), will help to understand the theoretical foundations, capabilities, and limitations of this framework.

6 Acknowledgments

This work was supported in part by the U.S. Naval Reseach Laboratory.

References

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. 2016. Hierarchical Planning: Relating Task and Goal Decomposition with Task Sharing. In *Proc. IJCAI 2016*, 3022–3029.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.*, 47(2-3): 235–256.

Chen, D. Z.; and Bercher, P. 2021. Fully Observable Nondeterministic HTN Planning - Formalisation and Complexity Results. In *Proc. ICAPS 2021*, 74–84.

Chen, D. Z.; and Bercher, P. 2022. Flexible FOND HTN Planning: A Complexity Analysis. In *Proc. ICAPS 2022*, 26–34.

Feldman, Z.; and Domshlak, C. 2014. Simple Regret Optimization in Online Planning for Markov Decision Processes. *J. Artif. Intell. Res.*, 51: 165–205.

Gelly, S.; and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.*, 175(11): 1856–1875.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press. ISBN 978-1-107-03727-4.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2025. *Acting, Planning, and Learning*. Cambridge University Press. ISBN 9781009579346.

Keller, T.; and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proc. ICAPS 2012*.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *Proc. ECML 2006*, 282–293.

Painter, M.; Lacerda, B.; and Hawes, N. 2020. Convex Hull Monte-Carlo Tree-Search. In *Proc. ICAPS 2020*, 217–225.

Patra, S.; Mason, J.; Ghallab, M.; Nau, D. S.; and Traverso, P. 2021. Deliberative acting, planning and learning with hierarchical operational models. *Artif. Intell.*, 299: 103523.

Patra, S.; Mason, J.; Kumar, A.; Ghallab, M.; Traverso, P.; and Nau, D. S. 2020. Integrating Acting, Planning, and Learning in Hierarchical Operational Models. In *Proc. ICAPS 2020*, 478–487.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. S. 2013. The GoDeL Planning System: A More Perfect Union of Domain-Independent and Hierarchical Planning. In *Proc. IJCAI 2013*, 2380–2386.

Shivashankar, V.; Kuter, U.; Nau, D. S.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. AAMAS 2012*, 981–988.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587): 484–489.

Yousefi, M.; and Bercher, P. 2024. Laying the Foundations for Solving FOND HTN Problems: Grounding, Search, Heuristics (and Benchmark Problems). In *Proc. IJCAI 2024*, 6796–6804.

A Formal Analysis of Hierarchical Planning with Multi-Level Abstraction

Michael Staud[1,2]

[1]StaudSoft UG, D-88213 Ravensburg, Germany
[2]University of Ulm, Institute of Artificial Intelligence
D-89069 Ulm, Germany
michael.staud@staudsoft.com

Abstract

We analyze Hierarchical World State Planning (HWSP), a novel algorithm that tackles the scalability limitations of Hierarchical Task Network (HTN) planning. By combining multi-level state abstraction with predictive task decomposition, HWSP reduces exponential search space growth. We formalize predictive separable tasks, classify planning domains, and derive tight bounds on search complexity. Our results show that HWSP transforms exponential interactions into linear coordination, enabling polynomial-time top-level planning in favorable domains where sibling tasks are independent. This enables efficient planning in complex domains that were previously intractable, opening up new possibilities for real-world applications.

Introduction

Many real-world domains – from urban planning to large-scale logistics – require reasoning over vast, structured state spaces. Hierarchical Task Network (HTN) planning mitigates this growth by decomposing high-level tasks, yet even totally ordered HTN search can be overwhelmed: the number of sibling-task combinations often grows exponentially, straining both memory and computation. Existing scalability techniques, such as macro-operators, factored planning, and state abstraction, either sacrifice HTN semantics or still require exhaustive inter-sibling search. To address this, we build on the recently proposed Hierarchical World State Planning (HWSP) algorithm (Staud 2023), which introduces multi-level state abstraction and separable abstract tasks to break large problems into smaller, soundly connected subproblems.

We ground our work in the standard HTN planning formalism (Erol, Hendler, and Nau 1994), which models domains in terms of *primitive* tasks (directly executable actions) and *abstract* tasks (compound tasks) that are refined via *methods* into constrained task networks with ordering, binding, and dependency constraints. Our analysis focuses on **totally ordered HTN, acyclic (TO-HTN) planning**, where all task networks are strictly ordered. This restriction is motivated by two factors: (1) TO-HTN underlies many widely deployed planning systems, and (2) its decidability and complexity are well-characterized (Geier and Bercher 2011; Alford et al. 2012), making it a solid baseline for isolating the effects of multi-level state abstraction in HWSP.

In this paper, we provide a theoretical analysis of HWSP's planning efficiency across varying domain structures. We introduce a classification of domains – EX1, EX2, and EX3 – that captures how separable tasks interact and when backtracking is necessary. Our results establish tight upper bounds on the number of search nodes visited by HWSP under each domain class. We contrast these results with a progression-based HTN search algorithm, showing that under certain structural assumptions (e.g., sibling task independence), HWSP avoids the exponential exploration of sibling task combinations that characterizes interleaved HTN search. Furthermore, we incorporate a probabilistic analysis of binary decomposition predictors, showing how predictor quality impacts search effort in EX2 domains.

We organize the paper as follows: Section "Formal Framework" introduces the formal framework of Hierarchical World State Planning (HWSP), including its planning model, semantics, and interface definitions. We also present a city planning domain that is used to illustrate the key concepts. In Section "Domain Classes", we define three domain classes (EX1 – EX3) that constrain how separable abstract tasks behave, particularly with respect to decomposition and backtracking. Section "Backtracking" analyzes the backtracking behavior of HWSP under these domain classes. Section "Results" presents the theoretical results comparing HWSP and classical HTN planning.

Related Work

The basic idea of HWSP is to divide a planning problem into multiple smaller ones.

Early work by Korf (1987) demonstrated that decomposing a planning problem into subproblems can yield significant improvements in search efficiency. This idea was further developed by Knoblock (1990), who introduced algorithms for automatically creating abstraction hierarchies and proved that solving problems on multiple levels of abstraction can be exponentially more efficient than flat planning. The concept of factored planning, introduced by Brafman and Domshlak (Brafman and Domshlak 2006), also shows that problem factorization can lead to exponential reductions in effort.

Prior work has recognized macro-operators as an effective tool for improving planning efficiency. In classical STRIPS planning, a macro-operator is a precompiled or learned high-

level action that encapsulates a sequence of primitive actions, thereby reducing search depth. In HWSP, separable abstract tasks play a related role: they encapsulate recurring hierarchical substructures, offering a similar reduction in search depth while also supporting richer ordering constraints. Korf's (1985) showed that, in certain domains, appropriately chosen macro-operators can substantially reduce effective search depth, turning exponential-time search into polynomial-time search for those cases. More recent work by Botea et al. (2005) demonstrated the effectiveness of automatically learned macro-operators in improving planning performance.

Hierarchical reinforcement learning (HRL) leverages both temporal and state abstraction to accelerate decision-making in long-horizon tasks. Like HWSP, HRL maintains representations of the environment at multiple abstraction levels; in HRL, these representations are coupled with a hierarchy of policies that operate over different temporal scales. Dietterich (2000)'s foundational MAXQ framework formalizes how a Markov Decision Process (MDP) can be decomposed into a hierarchy of subtasks, each with its own value function and (potentially abstracted) state representation. More recent work by Kulkarni et al. (2016) demonstrated that such multi-level abstractions can achieve strong performance in challenging, visually rich game environments.

Beyond classical HTN planning, several lines of work have addressed hierarchical abstraction, including independent subproblem identification (Lotem and Nau 2000), decomposition axioms (Biundo and Schattenberg 2001), and hierarchical goal networks (Shivashankar et al. 2012). HWSP builds on this tradition, but focuses on formalizing multi-level world state abstraction for scalability while retaining the soundness guarantees of HTN semantics. Sacerdoti's ABSTRIPS system (Sacerdoti 1974) introduced a form of state-space abstraction in which selected predicates are omitted at higher levels, yielding an abstract search space whose solutions are incrementally refined – though refinements can require backtracking. Angelic hierarchical planning (Marthi, Russell, and Wolfe 2008) instead abstracts actions, associating them with optimistic and pessimistic effect summaries to produce provable cost bounds during search. While this resembles separable abstract tasks in summarizing lower-level effects, focuses on bounded-cost optimality rather than multi-level state abstraction for scalability. More recent advances include compiling HTNs to SAT for optimal search (Behnke, Höller, and Biundo 2019), defining standard hierarchical languages such as HDDL (Höller et al. 2020a), and incorporating HTN planning into the IPC competition track to support empirical evaluation.

The interaction between task ordering, commitment strategies, and backtracking in HTN planning has been extensively investigated. Olz and Bercher (2023) proposed a look-ahead evaluation method for partially decomposed plans, enabling early pruning of unpromising decompositions and thereby reducing backtracking. Tsuneto et al. (1996) examined variable and method commitment policies in HTN planning, comparing early-commitment and delayed-commitment strategies and their impact on search efficiency.

Formal Framework

This Section describes the Hierarchical World State Planning (HWSP) algorithm, which extends the traditional planning paradigm by introducing a multi-level world state (Staud 2022, 2023). The HWSP algorithm is based on the concept of *separable abstract tasks*, which enable more efficient planning by partitioning the planning into smaller subplanning processes across multiple levels of abstraction. The following version of the HWSP algorithm is a more formal one compared to the original paper. We assume a progressive planning strategy.

Let A denote the set of propositional atoms. A literal is an atom or its negation. Each atom $a \in A$ represents a basic proposition that can be TRUE or FALSE. For any set of literals E, we define two functions $E^+ = \{a \in A \mid a \in E\}$ and $E^- = \{a \in A \mid \neg a \in E\}$.

Planning Domain

A planning domain is denoted as $D = (A, T_a, T_p, M, L, L_d, D_{\text{der}}, \Phi)$ where A is the set of propositional atoms, T_a is the set of all abstract tasks (including both standard and separable tasks), T_p is the set of primitive tasks, $T_S \subset T_a$ denotes the set of separable abstract tasks, M is the set of methods, and $L \in \mathbb{N}$ the number of detail levels (i.e., abstraction levels) present in the domain. The detail level function $L_d : A \cup T_a \cup T_p \to \{1, \dots, L\}$ assigns each atom or task a detail level ℓ. The derived atoms (derived predicates (Edelkamp and Hoffmann 2004)) are defined in D_{der}, with their corresponding logical formulas specified in Φ.

Tasks are tuples in the form $t = \langle prec_t, \mathit{eff}_t \rangle$. For primitive tasks and separable abstract tasks, $prec_t$ and eff_t are sets of propositional literals with semantic meaning in the planning process. For non-separable abstract tasks, $prec_t = \mathit{eff}_t = \emptyset$; we maintain the tuple structure for formal consistency, but these tasks derive their semantics solely through decomposition methods.

For separable abstract tasks $t \in T_S$, we write $\mathit{eff}_t = \mathit{meff}_t \cup \mathit{peff}_t$ where meff_t are mandatory effects and peff_t defines the set of *possible* non-guaranteed effects. During planning, the prediction function $\varepsilon(s, t_S) \subseteq \mathit{peff}_t$ selects which subset of these possible effects will actually occur (see Definition 1). Let $\mathcal{L}$ be a set of unique identifiers (labels), which we assume to be countable and disjoint from all other syntactic entities in the domain. A plan step is a uniquely labeled task $l : t$, $l \in \mathcal{L}$. A *method* is a tuple $m = \langle t_a, P_m \rangle$, where t_a is the abstract task it can decompose, and $P_m = (l_1 : s_1, l_2 : s_2, \dots, l_k : s_k)$ is a sequence of labeled plan steps (totally ordered).

A *state* s is an element of $\mathcal{P}(A)$, where $\mathcal{P}(A)$ denotes the power set of A. The *step function* $\mathsf{step} : \mathcal{P}(A) \times (T_p \cup T_S) \to \mathcal{P}(A)$ applies a task's effects to the world state: $\mathsf{step}(s, t) = (s \cup \mathit{eff}_t^+) \setminus \mathit{eff}_t^-$. We require that a separable abstract task t_S at detail level ℓ decomposes only into tasks at detail level $\ell + 1$, while other tasks decompose at the same level ℓ. Hence, separable tasks must occur at levels $\ell < L$. Let $s \downarrow \ell = \{a \in s | L_d(a) = \ell\}$ denote the projection of the state s to the detail level ℓ. A **planning problem** is defined as a tuple $\Pi = (D, s_0, t_0)$, where D is the planning domain, $s_0 \in \mathcal{P}(A)$ is

the initial world state, and $t_0 \in T_a$ is the initial abstract task to be decomposed. If a goal is needed, it is added as an end task.

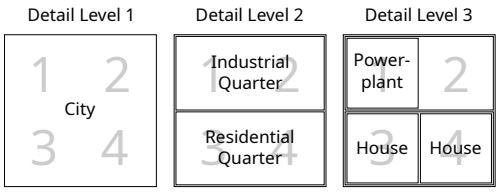

Figure 1: The different detail levels of the example domain in the goal state.

All detail layers in the world state with index $\ell < L$ consist solely of derived atoms and depend only on the atoms at the next finer detail level $\ell + 1$, thereby determining how information is abstracted in the world state. Let $D_{\text{der}} \subset A$ be the set of derived atoms (Edelkamp and Hoffmann 2004), which includes all atoms that are not on the most concrete (non-abstract) detail level. For each derived atom $a_{\text{der}} \in D_{\text{der}}$ at detail level ℓ, there exists a propositional formula $\phi_{a_{\text{der}}} \in \Phi$ involving only atoms from the next detail level $(\ell + 1)$. Formally:

$$a^{\ell}_{\text{der}} := \phi_{a_{\text{der}}}(a^{\ell+1}_1, a^{\ell+1}_2, \ldots, a^{\ell+1}_k)$$

where $a^{\ell+1}_i$ denotes atoms at detail level $\ell + 1$.

This definition of derived atoms is more restrictive than in PDDL 2.2 (Edelkamp and Hoffmann 2004), and no dependency cycles are allowed, even across multiple predicates.

We restrict our attention to domains with *finite task depth*. The *task depth* of a task is defined as the length of the longest sequence of tasks obtainable through successive method applications. For any domain D, let $\text{depth}_{\max}(D)$ denote its maximum task depth. We assume that there exists a fixed natural number $\Delta \in \mathbb{N}$ such that $\text{depth}_{\max}(D) \leq \Delta$ for every domain D considered in this paper.

Example Domain To illustrate HWSP concepts, we consider a city planning domain with three abstraction levels. The domain models a 2×2 city containing 2 quarters, each with 2 building cells. Planning tasks include constructing the city at level 1, individual quarters at level 2 (industrial or residential), and placing specific buildings at level 3. Industrial quarters require a single power plant, while residential quarters need two houses. Certain cells may be blocked, preventing construction, and power requirements can create dependencies between quarters depending on the preconditions.

We formalize this as domain $D = (A, T_a, T_p, M, 3, L_d, D_{\text{der}}, \Phi)$ for a 2×2 city with quarters $q_j = \{2j-1, 2j\}$ for $j \in \{1, 2\}$. The atoms are distributed across three levels: level 1 contains $\{\text{city\_built}, \text{has\_power\_city}\}$, level 2 contains $\{\text{industrial\_built}_{q_j}, \text{residential\_built}_{q_j}, \text{has\_power}_{q_j}\}$ for $j \in \{1, 2\}$, and level 3 contains $\{\text{house}_i, \text{powerplant}_i, \text{blocked}_i\}$ for $i \in \{1, 2, 3, 4\}$.

The tasks include build_city : $\langle \emptyset, \{\text{city\_built}\} \rangle$ (separable), a universal quarter task (separable) $\text{build\_quarter}_{q_j}$ and quarter tasks (abstract) $\text{build\_}type_{q_j}$: $\langle \emptyset, \{type\text{\_built}_{q_j}\} \rangle$ for $type \in \{\text{industrial}, \text{residential}\}$ and $j \in \{1, 2\}$. Primitive tasks are $\text{place\_}building_i : \langle \{\neg\text{blocked}_i\}, \{building_i\} \rangle$ for $building \in \{\text{house}, \text{powerplant}\}$ and $i \in \{1, 2, 3, 4\}$.

The derived atoms capture hierarchical relationships: $\text{industrial\_built}_{q_j} \equiv \vee_{i \in q_j} \text{powerplant}_i$; $\text{residential\_built}_{q_j} \equiv \wedge_{i \in q_j} \text{house}_i$; $\text{has\_power}_{q_j} \equiv \vee_{i \in q_j} \text{powerplant}_i$; $\text{has\_power\_city} \equiv \vee_{j \in \{1,2\}} \text{has\_power}_{q_j}$; $\text{city\_built} \equiv (\exists j \ \text{industrial\_built}_{q_j}) \wedge (\exists k \ \text{residential\_built}_{q_k})$. Each separable task has mandatory effects $meff$ corresponding to its completion predicate, while houses can additionally have preconditions $prec_t = \{\text{has\_power\_city}\}$.

Methods for the domain (totally ordered) are: m_{city}: $\text{build\_city} \Rightarrow (\text{build\_quarter}_{q_1}, \text{build\_quarter}_{q_2})$; $m_{q_j,\text{res}}$: $\text{build\_quarter}_{q_j} \Rightarrow (\text{build\_residential}_{q_j})$; $m_{q_j,\text{ind}}$: $\text{build\_quarter}_{q_j} \Rightarrow (\text{build\_industrial}_{q_j})$; m_{res,q_j}: $\text{build\_residential}_{q_j} \Rightarrow (\text{place\_house}_{i_1}, \text{place\_house}_{i_2})$, $i_1, i_2 \in q_j$; m_{ind,q_j}: $\text{build\_industrial}_{q_j} \Rightarrow (\text{place\_powerplant}_i)$, $i \in q_j$.

Predictor To enable efficient decomposition, HWSP incorporates predictor functions (Staud 2023) that estimate separable abstract task outcomes.

Definition 1 (Predictor Functions)**.** *Given a world state $s \in \mathcal{P}(A)$ and a separable task $t_S \in T_S$, the predictor function returns*

$$\pi(s, t_S) := \big(\varepsilon(s, t_S), \sigma(s, t_S)\big) \in \mathcal{P}(A) \times \{\text{TRUE}, \text{FALSE}\},$$

where

- $\varepsilon : \mathcal{P}(A) \times T_S \to \mathcal{P}(A)$ *is an* effect predictor *that proposes a set of* non-mandatory *effects (an estimate for peff_{t_S}), and*
- $\sigma : \mathcal{P}(A) \times T_S \to \{\text{TRUE}, \text{FALSE}\}$ *is the* decomposition predictor *whose output is to decide in a planning process if a separable abstract task can be decomposed or not.*

The predictor can be fixed-effect rules, Conv2D, ResNet or any other approximation function (Staud 2023).

HTN plans with labeled occurrences For proofs of the upcoming theorems, we need a description of a plan that stores the decomposition tree:

Definition 2 (HTN plan with explicit decomposition tree)**.** *A* plan (or partial plan) *is a tuple*

$$Q := \langle V, E, r, \lambda_T, \lambda_M, \prec \rangle$$

that is obtainable from the initial task network through valid method decompositions, where

- (V, E, r) *is a finite rooted tree whose root is $r \in V$,*
- *V is partitioned into the set of* task nodes *V_T and the set of* method nodes *V_M,*
- *$\lambda_T : V_T \to \mathcal{L} \times (T_a \cup T_p)$ is a labeling function assigning each task node a* plan step *$l : t$ with $l \in \mathcal{L}$ and $t \in T_a \cup T_p$,* $\mathsf{task} : V_T \to (T_a \cup T_p)$ *will return the task of a plan step,*
- *$\lambda_M : V_M \to M$ is a labeling function assigning each method node a method $m \in M$,*
- *$\prec$ is a strict total order on V_T (the task nodes) that is* consistent with the child order of each method node*,*

Edges alternate strictly between task and method nodes: if $(u, v) \in E$ with $u \in V_T$, then $v \in V_M$ and $\lambda_M(v) = \langle t, P_m \rangle$ where $\lambda_T(u) = l : t$ for some $l \in \mathcal{L}$; conversely, all children of a method node $u \in V_M$ are task nodes $v \in V_T$ whose plan steps $\lambda_T(v)$ correspond exactly, in order, to the plan steps in P_m.

Let $\mathsf{state\_at} : \mathcal{Q} \times V_T \to \mathcal{P}(A)$ be the function $\mathsf{state\_at}(Q, v)$ that returns the world state obtained from the initial state s_0 by applying, in the order $\prec$, the effects of all primitive task nodes $w \in V_T$ such that $w \prec v$ in plan Q. By Definition 5, all task nodes w such that $w \prec v$ must be fully decomposed before v is considered for decomposition. Hence, the cumulative world state $\mathsf{state\_at}(Q, v)$ is well-defined and includes only effects of primitive tasks. We define:

Definition 3 (Decomposability Test)**.** *Let Q be a plan and $v \in V_T$ a task node with $\lambda_T(v) = l : t$, and let $s := \mathsf{state\_at}(Q, v)$. We define* $\mathsf{can\_decompose} : \mathcal{Q} \times V_T \to \{\text{TRUE}, \text{FALSE}\}$ *by:*

$$\mathsf{can\_decompose}(Q, v) := \text{TRUE}$$

iff there exists a sequence of method applications starting from t such that:

1. *for every task t' in the resulting decomposition tree, the precondition of t' holds in the state in which t' is applied, and*
2. *all leaves in the decomposition subtree rooted at v are primitive tasks.*

Sibling-Rewriting Relation For two plans Q, Q' and a separable abstract task node $n_s \in V_T$ in Q, we write $Q \rightsquigarrow_{n_s} Q'$ iff Q' is obtained from Q by replacing only the subtree rooted at n_s. That is, the subtree at n_s is first removed (inverting its decomposition) and then replaced by a newly constructed decomposition (possibly different from the original one).

Definition 4 (Fully Decomposed Task)**.** *Let Q be a plan and $v \in V_T$ a task node. We define:* $\mathsf{is\_decomposed}(Q, v) := \forall u \in \text{leaves}_Q(v)\ \mathsf{task}(u) \in T_p$ *That is, v is* fully decomposed *in Q iff all of its leaf task nodes in the subtree rooted at v are primitive tasks.*

Definition 5 (Progressive Method Decomposition)**.** *Let $Q = \langle V, E, r, \lambda_T, \lambda_M, \prec \rangle$, $u \in V_M$ with parent $t \in V_T$, and method $\lambda_M(u) = \langle t_a, (l_1 : s_1, \ldots, l_k : s_k) \rangle$.* $\mathsf{decompose}(Q, u)$ *is defined* $\iff \forall w \in V_T : w \prec t \Rightarrow \mathsf{is\_decomposed}(Q, w)$.

If defined, $\mathsf{decompose}(Q, u) = Q' = \langle V', E', r, \lambda'_T, \lambda_M, \prec' \rangle$ *where:*

$$V' = V \cup \{v_1, \ldots, v_k\} \text{ (fresh task nodes)}$$

$$E' = E \cup \{(u, v_i) : i = 1, \ldots, k\}$$

$$\lambda'_T(v_i) = \mathsf{newlabel}(l_i) : s_i \text{ for } i = 1, \ldots, k; \lambda'_T(v) = \lambda_T(v)$$

$$\prec' = \big(\prec \setminus \big(\{(w, t) \mid w \prec t\} \cup \{(t, z) \mid t \prec z\}\big)\big)$$
$$\cup \quad \{(w, v_i) : w \prec t, i = 1 \ldots k\}$$
$$\cup \quad \{(v_i, z) : t \prec z, i = 1 \ldots k\}$$
$$\cup \quad \{(v_i, v_{i+1}) : i = 1, \ldots, k-1\}$$

The above definitions will be used in subsequent proofs.

HWSP Planning Algorithm

The most important difference between HTN planning and HWSP is the planning process (Staud 2023). In HTN planning, we decompose abstract tasks until only primitive tasks remain. In HWSP, decomposition stops at separable abstract tasks, unless we are on the most detailed level. This allows the planner to generate a top-level plan composed of separable abstract tasks, which are expected to work under typical conditions; otherwise, backtracking is performed.

For each separable abstract task emitted, a new planning process is launched to generate a subplan that achieves its mandatory effects $meff_t \subseteq D_{\text{der}}$. As the effects are derived predicates, the actual goal is the union of their logical formulas $\phi_{a_{\text{der}}}, a_{\text{der}} \in meff_t$.

Definition 6 (Planning Process Solution)**.** *A solution for planning process at level ℓ is a plan Q where every leaf node is either: (1) a primitive task ($\mathsf{task}(v) \in T_p$) if $\ell + 1 = L$, or (2) a separable abstract task ($\mathsf{task}(v) \in T_S$) if $\ell + 1 < L$ and all preconditions of these tasks must hold at execution time.*

Definition 7 (Planning Process Interface)**.** *A* planning process *(PP) is a 5-tuple $(D, I, \ell, \mathcal{I}, \mathcal{F})$ where:*

- *D is the planning domain*
- *$I \in I_S$ is the planning state of the planning process, where I_S is the set of all planning states. Given a separable task t, $\mathsf{Init}(t) \in I_S$ produces the initial planning state for planning t,*
- *$\ell \in \{1, ..., L\}$ is the current detail level*
- *$\mathcal{I} : I_S \times \mathcal{P}(A) \to I_S \times \big((\mathcal{L} \times (T_a \cup T_S \cup T_p)) \cup \{\textit{prec\_failure}, \textit{proc\_failure}, \textit{finished}\}\big)$*
  Given the current planning state and the world state, this function returns the updated planning state and either the next plan step $l : t$ to be added to the plan or a status indicator.
- *$\mathcal{F} : I_S \times \mathcal{P}(A) \times T_S \to I_S \times \mathbb{N}$*
  Handles the failure of a previously generated separable task $t \in T_S$ under the given world state. Returns the updated planning state and the number of previously generated tasks to retract (i.e., undo). The next call to $\mathcal{I}$ will generate a new alternative task or proc_failure which indicates that the planning process failed.

Internally, depending on the implementation, $\mathcal{F}$ may trigger backtracking or replanning.

Implementation Notes. The interface can be implemented using various search strategies. In an *MCTS-based* approach (Browne et al. 2012), backtracking simply returns to a previous node and selects a task with a lower visit count than the one discarded. An alternative is *HTN with repair* (Höller et al. 2020b), which rolls back the failed plan, removes the faulty task, and tries to complete a new plan by reusing parts of the old one.

Example In the city example, a planning process generates a plan to construct the buildings at the highest detail level 3 when given a quarter task as input.

Relationship to classical HTN planning. Unlike classical HTN problem spaces (Alford et al. 2012) where nodes represent single planning problems, HWSP operates over a hierarchy of interconnected planning processes, each working at different abstraction levels. If the domain provides only one level ($L = 1$) and contains no separable abstract tasks, HWSP degenerates into standard HTN planning: the single PP is invoked exactly once, no spawning occurs, and the algorithm behaves identically to a progression-based HTN search (i.e., standard forward-chaining HTN planning, as used in planners like SHOP2 (Nau et al. 2003)).

Algorithm 1: HTN planning with stack-based algorithm. Detailed backtracking procedures using $\mathcal{F}$ are implementation-dependent and are therefore omitted.

Input: Domain $D = (A, T_a, T_p, M, L, L_d, D_{\text{der}}, \Phi)$
Input: Problem $\Pi = (D, s, t_0)$
Initialize: Stack $S \leftarrow \langle (D, \mathsf{Init}(t_0), 1, \mathcal{I}, \mathcal{F}) \rangle$, MainPlan $p_M \leftarrow \langle\,\rangle$
while $S \neq \emptyset$ **do**
  $\langle D, I, \ell, \mathcal{I}, \mathcal{F} \rangle \leftarrow S.top()$
  $\langle I', x \rangle \leftarrow \mathcal{I}(I,\ s \downarrow \ell)$
  update $S.top()$ by replacing I with I'
  if $x =$ finished **then**
    $S.pop()$
  else if $x =$ prec_failure **or** $x =$ proc_failure **then**
    backtrack via using $\mathcal{F}$ or by selecting new planning processes.
  else ▷ x is a labeled step
    let $x = (l : t)$
    if $t \in T_p$ **then**
      $p_M \leftarrow p_M \oplus \langle l : t \rangle$
      $s \leftarrow \mathsf{step}(s, t)$
    else if $t \in T_S$ **then** ▷ assume $L_d(t) + 1 < L$
      $S.push((D, \mathsf{Init}(t), L_d(t) + 1, \mathcal{I}, \mathcal{F}))$
    end if
  end if
end while
return MainPlan p_M

Definition 8 (Overall Solution)**.** *A solution to the overall planning problem* $\Pi = (D, s_0, t_0)$ *is a plan* Q *where: (1) every leaf node is a primitive task (*$\mathsf{task}(v) \in T_p$*), and (2) all primitive task preconditions hold during sequential execution from* s_0*.*

Example In the city domain, the main planning algorithm would be coordinating the planning processes so that a complete city is built.

Domain Classes

The performance of planning algorithms, such as Hierarchical Task Network (HTN) planning and its extensions like Hierarchical World State Planning (HWSP), can be significantly influenced by the structure of the planning domain. This observation aligns with the downward refinement property introduced by Bacchus and Yang (Bacchus and Yang 1991), which emphasizes how domain structure impacts the

Algorithm 2: HWSP Planning Process Implementation: Core search procedure that handles three task types: primitive tasks (add to plan directly), separable abstract tasks (handled via predictive delegation to subprocesses), and non-separable abstract tasks (expanded using method decomposition). The interface functions $\mathcal{I}$ and $\mathcal{F}$ coordinate between the main algorithm and individual planning processes.

function $\mathcal{I}$(I, s) *The planning process maintains a local plan and incrementally returns tasks to the main algorithm. If no plan exists, it calls* `SolveLevel` *to generate one using the current world state projection. Returns the next unexecuted task from the plan, or status indicators:* `finished` *(all tasks complete),* `prec_failure` *(preconditions violated), or* `proc_failure` *(no valid plan exists).*
end function
function $\mathcal{F}$(I, s, t) *Called when separable task* t *fails during execution at a lower abstraction level and attempts to backtrack to find a new solution. Returns the number of previously-generated tasks that must be retracted from the main algorithm's plan.*
end function
function SOLVELEVEL(Q, s)
  $v = \mathsf{nextTaskNode}(Q)$, $\langle l : t \rangle \leftarrow \lambda_T(v)$
  if $t \in T_p$ **then** ▷ primitive task, add to plan directly
    $s' \leftarrow \mathsf{step}(s, t)$
    $(p_R, s'') \leftarrow$ SOLVELEVEL(Q, s')
    if $p_R =$ **failure then**
      return failure
    end if
    return $(\langle l : t \rangle \oplus p_R, s'')$
  else if $t \in T_S$ **then** ▷ separable abstract, treated like primitive task in planning process
    $(\varepsilon, \sigma) \leftarrow \pi(s, t)$ ▷ Get predicted effects
    if $\sigma =$ FALSE **then**
      ▷ Predictor says decomposition will fail
      return failure
    end if
    ▷ Apply mandatory + predicted effects
    $s' \leftarrow \mathsf{step}(s, \mathit{meff}_t \cup \varepsilon)$
    $(p_R, s'') \leftarrow$ SOLVELEVEL(Q, s')
    if $p_R =$ **failure then**
      return failure
    end if
    ▷ Include separable task in solution
    return $(\langle l : t \rangle \oplus p_R, s'')$
  else ▷ $t \in T_a \setminus T_S$ = non-separable abstract
    ▷ Select method and decompose it.
    $\mathcal{M} \leftarrow \mathsf{methods}(t)$ $m = \mathsf{select}(\mathcal{M})$
    $u := \mathsf{makeMethodNode}(v, m)$
    $Q' \leftarrow \mathsf{decompose}(Q, u)$
    $(p', s') \leftarrow$ SOLVELEVEL(Q', s)
    if $p' \neq$ **failure then**
      return (p', s')
    end if
    return failure
  end if
end function

feasibility and efficiency of hierarchical refinement. To formalize these influences in the context of separable abstract tasks, we introduce domain classes based on task interaction and backtracking behavior. In particular, we analyze how sibling tasks (parallel subtasks at the same abstraction level) may or may not interfere with one another.

Let us consider our urban planning example again, where a city manager plans construction in different quarters. We illustrate the three domain classes (EX1 – EX3) by toggling certain constraints in the same base domain, as summarized in Table 1.

Class	Active constraints
EX1	No blocked cells; no electricity requirement
EX2	Blocked cells present; no electricity requirement
EX3	Blocked cells present; we require has_power_city

Table 1: Configurations of the running example corresponding to the domain classes.

1. **EX1: Execution Guaranteed**
   Intuitive Definition: Execution (decomposition of the separable abstract task) is guaranteed once its preconditions hold in the current state.
   Example: In our urban planning scenario, when there are no blocked cells and no electricity requirement, the quarter can always be constructed.

   Definition 9 (Domain class EX1)**.** *A domain D is in **EX1** iff for every plan Q and every separable task occurrence $v \in V_T$ of Q, preconditions of* $\mathsf{task}(v)$ *are satisfied in* $\mathsf{state\_at}(Q, v)$*, then it holds* $\mathsf{can\_decompose}(Q, v) =$ TRUE

2. **EX2: Execution Can Fail but Won't Benefit from Backtracking into Siblings**
   Intuitive Definition: Execution can fail, but a different solution from another planning process on the same detail level won't help.
   Example: If we have blocked cells in the initial state (e.g., $\{\text{blocked}_2\} \subset s_0$), some quarter constructions can fail. For instance, build_residential$_{q_1}$ fails because it requires both cells 1 and 2 to be unblocked for house placement, but blocked$_2 \in s_0$. However, build_industrial$_{q_1}$ can still succeed using only cell 1 for power plant placement. Crucially, the decomposability of $build\_residential\_q_1$ depends only on $\{\text{blocked}_1, \text{blocked}_2\}$ and is independent of any choices made for build_industrial$_{q_2}$ or build_residential$_{q_2}$ in quarter q_2. When build_residential$_{q_1}$ fails, the planner backtracks to the parent build_city task and tries build_industrial$_{q_1}$ instead, without reconsidering previously completed quarters.

   Definition 10 (Domain class EX2)**.** *D is in **EX2** iff for all plans Q, separable occurrences $v \in V_T$ and for all sibling abstract task occurrences s located* strictly before *v in the depth-first ordering with $L_d(\mathsf{task}(s)) = L_d(\mathsf{task}(v))$, $\mathsf{task}(s) \in T_S \wedge \mathsf{task}(v) \in T_S \wedge Q \rightsquigarrow_s Q' \implies \mathsf{can\_decompose}(Q, v) = \mathsf{can\_decompose}(Q', v)$.*

   In words, replacing the internal subtree of an earlier separable sibling s does not affect the decomposability of a later separable task at the same detail level.

3. **EX3: Execution Can Fail and May Benefit from Backtracking into Siblings**
   Intuitive Definition: Execution can fail, and another solution from a prior planning process might help.
   Example: When residential quarters require power to become buildable, it creates dependencies between quarter choices. Attempting build_residential$_{q_1}$ as the first quarter fails because the precondition requires has_power_city to be true, but no power plants exist yet. The planner must backtrack to build_quarter$_{q_1}$ and try build_industrial$_{q_1}$ first to establish power generation, then build_residential$_{q_2}$ can succeed.

   Definition 11 (Domain class EX3)**.** *The class **EX3** includes all HWSP planning domains.*

   This formulation represents the general case without assuming task independence. Note, by propagating power availability upward via derived atoms (e.g., has_power$_{q_j}$, has_power_city), remove build_quarter$_{q_j}$, making build_residential$_{q_j}$, build_industrial$_{q_j}$ to separable tasks and including the electricity check in preconditions of, cross-sibling dependencies are captured at higher levels. This effectively transforms the domain from EX3 to EX2.

Separable abstract tasks isolate planning to local *islands* of the world state. How strongly such islands interact (EX1 – EX3) determines the degree to which global backtracking is required (in Algorithm 1).

Proposition 1 (Independence chain)**.**

$$EX1 \subset EX2 \subset EX3.$$

Proof. Immediate from the definitions □

Domain Conditions

Instead of a full classification, we formulate sufficient conditions which analyze how the preconditions of subtasks relate to the guarantees provided by their abstract parent. If every subtask t in a separable abstract task T has $\text{prec}(t) \subseteq \text{prec}(T)$, or $\text{prec}(t)$ can be established by effects of other subtasks of T (according to the decomposition defined by the chosen method), then the decomposition is self-contained: all requirements are either inherited or locally achievable. In this case the domain behaves like **EX1**, since successful execution is guaranteed once $\text{prec}(T)$ holds.

If, more generally, each subtask may also rely on derived predicates from a lower abstraction level, the task network is still independent of its siblings. This setting corresponds to **EX2**: execution may fail locally, but feasibility does not depend on choices made in other sibling tasks.

Backtracking

In HWSP, backtracking occurs when a planning process fails to find a viable solution. At this point, the algorithm must reconsider its previous decisions and explore alternative paths.

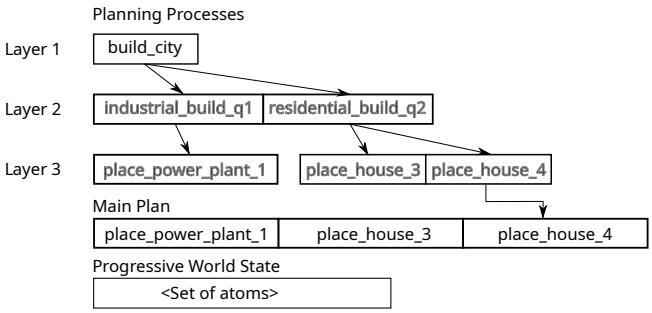

Figure 2: A typical state of the HWSP algorithm.

This is where the domain classes (EX1, EX2, and EX3) come into play, as they significantly influence the backtracking behavior.

- **EX1**: In domains where execution is guaranteed once preconditions of a separable abstract task are met, backtracking is relatively straightforward. Since each task's success is assured, the main planner never needs to backtrack due to separable abstract tasks.
- **EX2**: In EX2 domains, the failure of a separable abstract task requires restricted backtracking behavior.

  Proposition 2 (EX2 Backtracking Behavior)**.** *In an EX2 domain, if a separable abstract task v fails during decomposition, the planner can backtrack to the parent planning process without considering siblings of v.*

  Proof. By Definition 10, for any plan Q and separable task v, modifying any sibling s (with $s \prec v$) does not affect the decomposability of v. Formally: $\mathsf{can\_decompose}(Q, v) = \mathsf{can\_decompose}(Q', v)$ where $Q \rightsquigarrow_s Q'$.
  If v fails ($\mathsf{can\_decompose}(Q, v) = \text{FALSE}$), altering any prior sibling s through backtracking cannot make $\mathsf{can\_decompose}(Q', v) = \text{TRUE}$. Thus, the only recourse is to inform the parent process of v's failure. The parent must then generate a new separable abstract task to replace v or one of its siblings. If the parent exhausts all alternatives and fails, the failure propagates upward recursively. This eliminates the need to explore siblings of v, as their modifications cannot resolve v's failure under EX2's independence. The algorithm remains sound and complete (Staud 2022). □

- **EX3**: In these domains, where execution can fail and backtracking may help, we must perform full backtracking as in the original HWSP algorithm (Staud 2023).

Results

We compare *Hierarchical World-State Planning* (HWSP) on the three domain classes EX1 $\subset$ EX2 $\subset$ EX3 with a progression-based HTN algorithm.

Complexity model

Consider a fixed detail level ℓ. Let n_ℓ denote the number of *sibling* separable task occurrences emitted at that level, and let each sibling have at most k_ℓ decomposition methods. The refinement of *one* sibling induces a local search tree of branching factor $b_{\ell+1}$ and depth $h_{\ell+1}$; we abbreviate $B_{\ell+1} = b_{\ell+1}^{h_{\ell+1}}$.

Let N denote the total input size (i.e., the size of the domain and the initial task network). We assume that $n_\ell = \mathcal{O}(N)$; that is, the number of sibling tasks at each level grows at most linearly with the input.

We analyze the number of states visited during search, denoted as:

- S_{HTN}: states visited by classical HTN planning
- S_{EX1}, S_{EX2}, S_{EX3}: states visited by HWSP in the respective domain classes

State-space bounds

Theorem 3 (EX3 versus progression-based HTN)**.** *For any level ℓ*

$$S_{\mathrm{HTN}} = B_{\ell+1}^{n_\ell}, S_{\mathrm{EX3}} = n_\ell\, k_\ell^{n_\ell}\, B_{\ell+1}, \frac{S_{\mathrm{HTN}}}{S_{\mathrm{EX3}}} = \Theta\left(\frac{B_{\ell+1}^{n_\ell - 1}}{n_\ell}\right)$$

Proof. Classical HTN planning *inlines* the local search of each sibling. Consequently the global search tree contains $B_{\ell+1}$ possibilities for the first sibling, $B_{\ell+1}$ for the second, and so on, for a total of $B_{\ell+1}^{n_\ell}$ states.

In EX3, the parent process still explores every combination of high-level alternatives, hence $k_\ell^{n_\ell}$ parent choices. Once such a combination of high-level method choices $(m_1, \ldots, m_{n_\ell})$ has been fixed, each sibling is then solved *independently* by its own local search; at this stage the parent does not interleave those searches. The resulting state count is therefore $n_\ell\, k_\ell^{n_\ell}\, B_{\ell+1}$. Taking the quotient of the two expressions yields the claimed factor. □

Theorem 4 (EX2 versus EX3)**.** *For any level ℓ*

$$S_{\mathrm{EX2}} = n_\ell\, k_\ell\, B_{\ell+1}, \qquad \frac{S_{\mathrm{EX3}}}{S_{\mathrm{EX2}}} = \Theta\left(\frac{k_\ell^{n_\ell - 1}}{n_\ell}\right)$$

Proof. Definition 10 states that substituting an *earlier* sibling by an alternative decomposition never changes the decomposability of a *later* sibling. Hence, when a sibling fails, the planner backtracks exactly one level up, replaces *only* that sibling, and leaves all others untouched. Worst case is it will test every one of the k_ℓ methods before a success or final failure is found. Since there are n_ℓ siblings, at most $n_\ell\, k_\ell$ alternatives are generated, each of which spawns one local search of size $B_{\ell+1}$. The product yields S_{EX2}. Dividing by S_{EX3} establishes the gap. □

The bounds given in Theorems 3 and 4 are tight in the following sense: for each setting of n_ℓ, k_ℓ, and $B_{\ell+1}$, there exists a family of domains and initial task networks for which the number of visited states matches the bound asymptotically.

HTN Tightness Example. Let the initial task be a sequence of n_ℓ abstract tasks, each with k_ℓ methods, where each method expands to a local plan space of size $B_{\ell+1}$ (e.g., a binary tree of depth $\log_2 B_{\ell+1}$). If no heuristic guidance is used, the planner must exhaustively search all $B_{\ell+1}^{n_\ell}$ combinations.

EX3 Tightness Example. Consider a domain with n_ℓ sibling tasks (e.g., city quarters). Each has k_ℓ decomposition methods, exactly one of which builds a power plant. Suppose that every quarter requires electricity to decompose, and power flows only from left to right: quarter q_j can be decomposed only if some earlier sibling q_i with $i < j$ contains a `power plant`. This induces a causal dependency chain across siblings: choosing non-power plant methods early may prevent later decompositions, forcing backtracking and revision of earlier choices. In the worst case, it explores all $k_\ell^{n_\ell}$ global method assignments; for each one, it performs n_ℓ independent subsearches of size $B_{\ell+1}$. The total number of visited states is therefore $n_\ell\, k_\ell^{n_\ell}\, B_{\ell+1}$, which matches the upper bound for EX3. The cross-sibling dependency violates the EX2 invariance condition (Definition 10), so this domain lies strictly in EX3. Thus, the upper bound for S_{EX3} is *tight*.

EX2 Tightness Example. Construct a domain with n_ℓ separable sibling tasks (e.g., one per city quarter), each having k_ℓ distinct task alternatives. For each sibling, only one of its k_ℓ alternatives is decomposable, while the others are designed to fail due to local, hidden constraints (e.g., unobservable predicates or infeasible layouts). These constraints are specific to each sibling and cannot be inferred or affected by the other siblings' choices. When a sibling task fails, the parent planning process must replace it with another alternative, up to k_ℓ times in the worst case. Each attempted task triggers a local planning process that explores a subsearch space of size $B_{\ell+1}$. Since all sibling tasks operate over disjoint parts of the world state and have no shared predicates, changing one sibling's decomposition does not influence the decomposability of another – satisfying the EX2 invariance condition (Definition 10). The total number of visited states in this construction is $n_\ell k_\ell B_{\ell+1}$, matching the upper bound for EX2 domains and proving its tightness asymptotically in the worst case.

On EX1. Once the preconditions of a separable task are satisfied, its decomposition cannot fail. Inside the planning processes, however, non-monotonic search (e.g., local backtracking) is still permitted. Therefore EX1 improves the constant factors of Theorem 4 but does not change its asymptotic form. More precisely, the number of search states in EX1 is $S_{\mathrm{EX1}} = n_\ell\, B_{\ell+1}$, as each of the n_ℓ separable tasks is successfully decomposed on the first attempt, requiring a single local search of size $B_{\ell+1}$ per task.

Corollary 5 (Strict inclusion chain)**.** *It holds $S_{\mathrm{EX1}} < S_{\mathrm{EX2}} < S_{\mathrm{EX3}} < S_{\mathrm{HTN}}$ as each transition removes one exponential factor in the input size N, assuming $n_\ell = O(N)$ and $L > 1$.*

Connection to classical complexity theory

Bacchus and Yang's *Downward Refinement Property* (DRP) (Bacchus and Yang 1991) formalizes a key structural criterion for efficient hierarchical planning: once a high-level plan is found, it can be refined monotonically without modifying earlier decisions. Our EX1 domain class satisfies DRP by construction in Algorithm 1, as successful decomposition is guaranteed once preconditions are met. This is however not the case in Algorithm 2.

EX2 domains do not satisfy DRP in the strict sense, since the planner may backtrack and replace earlier siblings. However, they enforce a weaker property: the feasibility of decomposing a task is invariant under changes to the internal decomposition of prior siblings at the same abstraction level. Korf's macro-operator theorem (Korf 1985) predicts an exponential-to-linear collapse in ideal DRP-compatible settings, which applies to EX1 with respect to Algorithm 1 and partially to EX2. Note that while prior work (Alford et al. 2012) establishes bounds on HTN problem space size for termination guarantees, our analysis focuses on search effort complexity.

Effect of a (fallible) binary predictor

In the following, N is the combined size of the initial task network and domain description. Let $\hat{p}$ denote the predictor accuracy introduced below (to avoid notational conflict with earlier polynomials). In HWSP every separable task t_S comes with a binary *decomposition predictor* $\sigma : \mathcal{P}(A) \times T_S \to \{\textsc{true}, \textsc{false}\}$. Intuitively, $\sigma(s, t_S) = \textsc{true}$ says that local planning *ought* to succeed.[1] Mis-predictions are the only reason why an EX2 planner ever needs to backtrack.

Error model. For the sake of analysis we assume that, *conditioned on the current state*, the predictor returns the correct answer with probability $\hat{p}$ (*accuracy*) and the wrong answer with probability $\bar{p} = 1 - \hat{p}$.

Theorem 6 (Expected search effort with a fallible predictor)**.** *Let $S^\star_{\mathrm{EX2}} = n_\ell B_{\ell+1}$ be the number of search states when the predictor is* perfect*, so that each task is solved on the first attempt ($k_\ell = 1$ effectively).*

Assume that, each time a separable task is encountered, the predictor is correct with probability $\hat{p}$ and wrong with probability $\bar{p} = 1 - \hat{p}$ (independently of previous calls). If the predictor errors, the parent process tries a different alternative; at most $k_\ell - 1$ further alternatives exist. Then the expected number of visited states is bounded by

$$\mathbb{E}[S_{\mathrm{EX2}}(p)] \;\le\; S^\star_{\mathrm{EX2}}\,\big(1 + \bar{p}\,(k_\ell - 1)\big).$$

Proof. Let n_ℓ be the number of separable task occurrences at level ℓ, and let each task require a local search of size $B_{\ell+1}$ once a valid decomposition is selected. Under a perfect predictor, each task is decomposed successfully on the first try, so the total number of visited states is: $S^\star_{\mathrm{EX2}} = n_\ell B_{\ell+1}$. Now consider a fallible predictor with accuracy $\hat{p}$, and let $\bar{p} = 1 - \hat{p}$ denote the probability of an incorrect prediction. If the predictor fails (predicts incorrectly, whether a false positive or false negative), the planner must backtrack and try a different decomposition. Since each task has at most k_ℓ possible alternatives, the number of additional alternatives to try after a misprediction is at most $k_\ell - 1$.

Thus, the expected number of task attempts for each separable task is at most: $1 \cdot \hat{p} + (1 + \beta) \cdot \bar{p}$, with

[1] We treat σ as a black box: it may be a learned classifier, a logical test or a mix thereof.

$\beta \leq k_\ell - 1$. Substituting, we obtain the upper bound: Expected attempts per task $\leq 1 + \bar{p} \cdot \beta \leq 1 + \bar{p} \cdot (k_\ell - 1)$. Multiplying this overhead by the base cost $S^\star_{\mathrm{EX2}}$ yields: $\mathbb{E}\left[S_{\mathrm{EX2}}(p)\right] \leq S^\star_{\mathrm{EX2}} \cdot (1 + \bar{p} \cdot (k_\ell - 1))$, which proves the claim. □

A perfect predictor collapses EX2 to EX1. Setting $\hat{p} = 1$ makes the overhead term $(1 + \bar{p}\,\beta)$ equal to 1, so every separable task is decomposed successfully on the very first try. This eliminates *all* backtracking in Algorithm 1 – precisely the hallmark of the EX1 class (Def. 9). Formally, the planner's behavior on this instance becomes behaviorally indistinguishable from running HWSP on an EX1 domain, although the structural EX1 property may not hold for all instances. □

Interpretation. Theorem 6 shows that the predictor's accuracy enters the search complexity as a *linear* scalar factor. Even modest accuracies (say $\hat{p} = 0.9$) leave the $\Theta\big(k_\ell^{n_\ell - 1}\big)$ gap of Theorem 4 largely intact (here treating n_ℓ as fixed for ratio analysis), because $\bar{p} \leq 0.1$ (Staud 2023).

Conclusion

We analyzed Hierarchical World State Planning (HWSP), a planning framework that extends classical HTN planning through multi-level state abstraction and separable abstract tasks. While HWSP was introduced in prior work, its theoretical properties and complexity characteristics had not been formally studied.

References

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. 2012. HTN Problem Spaces: Structure, Algorithms, Termination. In *Proceedings of the International Symposium on Combinatorial Search*, volume 3, 2–9.

Bacchus, F.; and Yang, Q. 1991. The Downward Refinement Property. In *IJCAI*, 286–293.

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

Biundo, S.; and Schattenberg, B. 2001. From Abstract Crisis to Concrete Relief – A Preliminary Report on Combining State Abstraction and HTN Planning. In *ECP 2001*, 157–168.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research*, 24: 581–621.

Brafman, R. I.; and Domshlak, C. 2006. Factored Planning: How, when, and When Not. In *AAAI*, volume 6, 809–814.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303.

Edelkamp, S.; and Hoffmann, J. 2004. PDDL 2.2: The Language for the Classical Part of IPC-4. In *International Planning Competition*.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In Hayes-Roth, B.; and Korf, R. E., eds., *Proceedings of the 12th National Conference on Artificial Intelligence, Volume 2*, 1123–1128.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *IJCAI*, 1955–1961.

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

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020b. HTN Plan Repair Via Model Transformation. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, 88–101.

Knoblock, C. A. 1990. Learning Abstraction Hierarchies for Problem Solving. In *AAAI*, 923–928.

Korf, R. E. 1985. Macro-Operators: A Weak Method for Learning. *Artificial Intelligence*, 26(1): 35–77.

Korf, R. E. 1987. Planning as Search: A Quantitative Approach. *AI 87*, 33(1): 65–88.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *Advances in Neural Information Processing Systems*, 29.

Lotem, A.; and Nau, D. S. 2000. New Advances in GraphHTN: Identifying Independent Subproblems. In *Proc. AIPS 2000*, 206–215.

Marthi, B.; Russell, S.; and Wolfe, J. A. 2008. Angelic Hierarchical Planning: Optimal and Online Algorithms. In *ICAPS*, 222–231.

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

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.

Sacerdoti, E. D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial intelligence*, 5(2): 115–135.

Shivashankar, V.; Kuter, U.; Nau, D. S.; and Alford, R. 2012. Hierarchical Goal Networks: Initial Results. In *Proc. ICAPS 2012*, 235–243.

Staud, M. 2022. Urban Modeling via Hierarchical Task Network Planning. In *HPLAN Workshop*.

Staud, M. 2023. Integrating Deep Learning Techniques into Hierarchical Task Planning for Effect and Heuristic Predictions in 2D Domains. In *HPLAN Workshop*.

Tsuneto, R.; Erol, K.; Hendler, J.; and Nau, D. 1996. Commitment Strategies in Hierarchical Task Network Planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 536–542.

Towards a General Framework for HTN Modeling with LLMs

Israel Puerta-Merino[1], Carlos Núñez-Molina[2], Pablo Mesejo[1], Juan Fernández-Olivares[1]

[1]University of Granada, Spain
[2]RWTH Aachen University, Germany
israelpm01@ugr.com, carlos.nunez@ml.rwth-aachen.de, pmesejo@ugr.es, faro@decsai.ugr.es

Abstract

The use of Large Language Models (LLMs) for generating Automated Planning (AP) models has been widely explored; however, their application to Hierarchical Planning (HP) is still far from reaching the level of sophistication observed in non-hierarchical architectures. In this work, we try to address this gap. We present two main contributions. First, we propose L2HP, an extension of L2P (a library to LLM-driven PDDL models generation) that support HP model generation and follows a design philosophy of generality and extensibility. Second, we apply our framework to perform experiments where we compare the modeling capabilities of LLMs for AP and HP. On the PlanBench dataset, results show that parsing success is limited but comparable in both settings (around 36%), while syntactic validity is substantially lower in the hierarchical case (1% vs. 20% of instances). These findings underscore the unique challenges HP presents for LLMs, highlighting the need for further research to improve the quality of generated HP models.

Code — https://github.com/Corkiray/L2HP

Introduction

Automated Planning (AP) is a foundational and widely applied area within AI. Despite its successes, AP continues to face two major challenges: (1) the high computational cost associated with solving large-scale problems, and (2) the difficulty of accurately formalizing complex world models. The first challenge can be mitigated through the Hierarchical Planning (HP) approach. HP leverages multiple levels of abstraction to accelerate planning and produce more interpretable, human-like plans.

Concurrently, recent advancements in AI, particularly the development of Large Language Models (LLMs), offer promising tools to address the second challenge. LLMs, trained on massive textual corpora using deep neural architectures, exhibit strong natural language (NL) understanding and generation capabilities, resulting in a great adaptability to a wide variety of contexts without requiring additional task-specific training. Their proficiency in translating information into structured outputs positions them as promising candidates for model generation (i.e. domain plus instance files generation) in AP.

Given this potential, integrating LLMs into HP systems emerges as a compelling research direction. We posit that hybrid planning frameworks, where LLMs generate hierarchical models and symbolic planners leverage these models to search for a solution plan, represent a promising avenue.

LLMs have received increasing attention for classical AP modeling (Tantakoun, Zhu, and Muise 2025). However, as shown by Puerta-Merino et al. (2025), there remains a substantial gap between research on the integration of in LLMs into AP and their specific integration into the HP life-cycle. To address this gap and encourage the research of LLMs in HP, we introduce two main contributions. First, we propose **L2HP**, a framework built upon the L2P library (Tantakoun, Zhu, and Muise 2025) to support LLM-driven model generation for HP. We designed L2HP with a focus on generality and extensibility, to facilitate and encourage the implementation of future LLM-driven HP systems. Second, we apply L2HP to conduct a preliminary **empirical study** that shows the limitations of current LLM-based HP modeling systems, highlighting the need for further research into this field.

We use PlanBench (Valmeekam et al. 2023), a dataset designed to evaluate planning and reasoning capabilities in NL, to compare the quality of AP and HP models generated with a baseline LLM method.[1] Our results reveal two key findings. First, LLMs exhibit significant difficulty in adhering to the expected output structure, with over 60% parsing failure across both AP and HP. Second, the syntactic validity is substantially lower for HP compared to AP (1% vs. 20% of instances, respectively). Our findings demonstrate the limitations of standalone LLMs for AP model generation, which accentuate in the case of HP. This highlights the need for further research into this area, for which we hope our framework will serve as a valuable tool.

Background

Hierarchical Planning

HP extends AP by solving problems through multiple levels of abstraction (Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013). This approach is grounded in two foundational principles: (1) many real-world problems exhibit an inherent hierarchical structure that can be exploited to guide planning, and (2) this structure mirrors human problem-solving strategies, typically more hierarchical than sequen-

[1]https://huggingface.co/datasets/tasksource/planbench

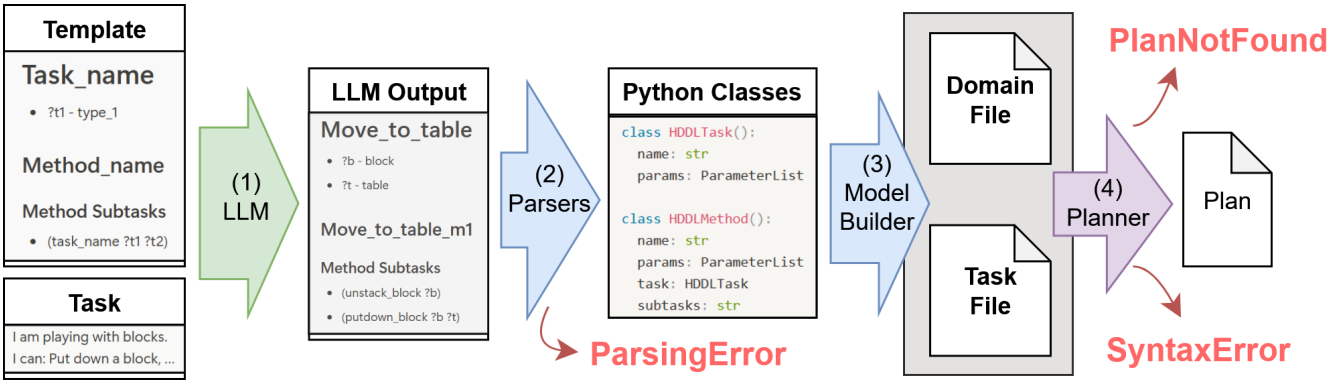

Figure 1: Overview of a typical L2HP workflow: (1) The LLM is asked to model a planning task, following a template written in Markdown. (2) The LLM output is parsed to extract the planning elements into structured Python Classes. (3) Using the structured data, the planning model is generated – i.e., both the domain and task files are created. (4) A symbolic planner is then invoked to generate a plan that solves the given problem.

tial. Consequently, HP often results in more interpretable and computationally efficient planning than classical AP.

The most prominent formalism of HP is the Hierarchical Task Networks (HTN) framework (Nau et al. 2003). In HTN planning, abstract tasks are iteratively decomposed into simpler subtasks through predefined methods, until a sequence of primitive actions is obtained. To support consistent domain modeling across various planners, several languages have been proposed, highlighting HPDL (Fdez-Olivares et al. 2006) and HDDL (Höller et al. 2020), both extending the classical PDDL syntax to support a smooth transition to HP structures. HDDL has become the most widely adopted standard due to its broad planner support.

However, despite its advantages in scalability, interpretability, and knowledge reuse, HP presents its own challenges. Chief among these is the need for domain experts to define tasks, methods, and hierarchical rules, increasing the complexity of modeling, especially for new or poorly structured domains (Erol, Hendler, and Nau 1994). Furthermore, since it is a more expressive approach than classical AP, the modeling process, as well as the validation, debugging, and maintenance of HTN models, becomes more complex (Erol, Hendler, and Nau 1994). Moreover, the absence of automated tools to assist in creating or refining hierarchical models further limits adoption of HP (Zhuo, Muñoz-Avila, and Yang 2014). Therefore, although HP aligns well with how humans decompose and reason about complex tasks, its scalability and adoption in new environments are currently limited by the difficulty of its modeling process.

Planning with LLMs

In last years, the use of LLMs in planning is being widely explored. This is a relatively new and rapidly evolving area of research. Early studies investigated how LLMs could assist AP by suggesting action sequences or guiding search processes. Subsequent work extended their use to translating (NL into formal models, inferring symbolic representations from observations, and even acting as approximate planners. Given the diversity and pace of this field, several surveys have emerged to consolidate and clarify the state of the art (Pallagani et al. 2024; Huang et al. 2024; Valmeekam et al. 2022). Within this context, two major lines of research can be identified: LLMs-as-Planners and LLMs-as-Modelers.

The motivation for using LLMs directly as reasoning agents or planners is reasonable. These models, trained on vast textual corpora, can perform approximate reasoning without relying on structured symbolic representations. One of the earliest works in this line is the approach of Silver et al. (2022), but the field has rapidly advanced, with increasingly sophisticated architectures aimed to enhance planning capabilities. These systems incorporate techniques that has been shown to significantly improve LLMs performance, such as Chain of Thought (Wei et al. 2022), Few-shot prompting (Brown et al. 2020), output refinement (Madaan et al. 2024), etc. One of the most works following this approach is the LLM-Modulo (Kambhampati et al. 2024).

On the other hand, the use of LLMs as model generators is also well-motivated. Their capacity to generate structured text, encode commonsense knowledge, and operate in both natural and formal languages positions them as valuable tools for knowledge engineering in planning. LLM+P (Liu et al. 2023) is one of the first works to generate partial AP models with LLMs, by translating a task NL description into a PDDL task file. Several notable works have since extended this idea, integrating LLMs into more sophisticated architectures, as NL2Plan (Gestrin, Kuhlmann, and Seipp 2024) and (Guan et al. 2023). For an in-depth overview, see the survey by (Tantakoun, Zhu, and Muise 2025), which focuses specifically on the LLMs-as-Modelers paradigm.

LLMs in Hierarchical Planning

As shown above, LLMs have received increasing attention for classical AP modeling and are being progressively integrated into the AP life cycle. However, a substantial gap remains between research on the integration of LLMs into AP in general and their specific integration into the HP life cycle, with only a limited number of studies addressing this topic (Puerta-Merino et al. 2025).

Following the LLMs-as-Planners paradigm, some works have explored the use of LLMs to generate hierarchically structured plans. These plans are typically represented in NL and generated iteratively, by prompting the LLM to decompose a task into subtasks until a complete plan is formed (Tse 2024; Tianxing et al. 2025; Zhao et al. 2024; Yang, Zhang, and Hou 2024). As with AP, more sophisticated architectures can be developed to improve the LLM performance. For example, (Kienle et al. 2025) builds on the LLM-Modulo framework to generate hierarchical plans.

In the LLMs-as-Modelers line, one of the first HP approaches is (Luo, Xu, and Liu 2023), which uses an HTN-like NL structure to generate a task representation that is then translated into a Linear Temporal Logic (LTL) model. In the specific context of HTN generation, more recent works have also emerged (Fine-Morris et al. 2024; Sinha 2024; Munoz-Avila, Aha, and Rizzo 2025).

Finally, there are also noteworthy hybrid approaches: (Dai et al. 2024) employs an LLM as heuristic to accelerate an LTL planning process, while (Hsiao et al. 2025) combines a symbolic HTN planner for high-level reasoning and an LLM Agent as low level planner.

L2P Library

Although the integration of LLMs within various stages of the AP life-cycle is being widely explored, the lack of standardized tools and methodologies continues to limit the scalability and reproducibility of LLM-based planning research. At the same time significant efforts are underway to simplify LLM integration through streamlined prompt engineering, API access, and output parsing. Notable contributions include the LangGraph framework[2] and the Model Context Protocol[3].

In the specific context of AP, one of the pioneering efforts is L2P – LLM-driven Planning Model library kit (Tantakoun, Zhu, and Muise 2025). To the best of our knowledge, L2P is the first library designed to facilitate the LLM-driven generation of PDDL models (i.e. domain plus instance files). Given its scope, modularity and extensibility, L2P was selected as the foundational platform for building our HP-focused extension: L2HP.

L2P offers a suite of tools for building custom architectures. Its core functionality centers on a set of *extraction* methods, which allow to easily use an LLM to obtain specific structured data representations of PDDL components (e.g., actions, predicates, parameters). These methods are conformed by pipelines that apply prompt templates, invoke an LLM, and parse the generated responses into structured objects. This general workflow – illustrated in steps (1-2) of Figure 1 – is always similar across use cases, but can be applied at varying levels of complexity, from generating individual actions to producing complete PDDL domain files.[4]

[2]https://github.com/langchain-ai/langgraph

[3]https://github.com/modelcontextprotocol

[4]Figure 1 represents a typical L2HP workflow, and does not exactly reflect L2P behavior (e.g., L2P does not use Markdown). However, since L2HP is inspired by and built on top of L2P, the overall process is conceptually similar and closely reflect the core logic of L2P

This workflow proceeds as follows:

1. **Template Application**. The NL task description is embedded into a prompt template designed to elicit a specific structured response.
2. **LLM Invocation**. The prompt is submitted to an LLM via API or local inference.
3. **Response Parsing**. The LLM output, which is expected to match the template, is parsed into structured elements using custom parsers.

The orange elements in Figure 2 represents the core components of the L2P library. Some of these will be discussed in more detail in the *L2HP Framework* section. Briefly, L2P provides a collection of templates and parsers for various planning constructors. Parsed elements are managed using two builder classes: one for domain-level definitions and another for task-level specifications. These builders not only manage structured data but also implement the mentioned extraction methods, along with additional functionalities such as automatic PDDL generation. L2P also offers seamless integration with LLM APIs and a planner, as well as validation tools, feedback mechanisms, and a prompt construction toolkit. Its versatility is further demonstrated by its integration of several previously decentralized, state-of-the-art LLM-based planning architectures.

L2HP Framework

L2HP is introduced as the first framework designed to support the use of LLMs as model generators for HP. However, its overarching goal extends beyond only reducing entry barrier and encourage the community to research in this emerging area; it also aims to simplify the development and evaluation of LLM-based planning architectures. Therefore, the design of L2HP is guided by two core principles:

1. **Generalization and Extensibility**. L2HP is structured to allow seamless integration of LLMs, planners, templates and architectures, ensuring that the experiments are reproducible and independent of implementation details. Note that, due to the nondeterministic nature of LLMs, the results might variate even when running the exactly same experimentation.
2. **HP Integration**. We provide a suite of tools designed for HTN modeling and planning, including support for HPDL, HDDL, and HTN-compatible planners.

To achieve this, L2HP builds upon L2P by extending its capabilities to support hierarchical structures. It introduces new tools and abstractions to promote experimentation, along with additional integrations specifically designed for HP. All components that conform L2HP are described in the following subsections and illustrated in Figure 2.

Generalization and Extensibility

L2HP has integration with a variety of planners and LLMs with modular compatibility. To promote flexible experimentation, L2HP introduces a general `Planner` class that standardizes the execution of the different planners with a same

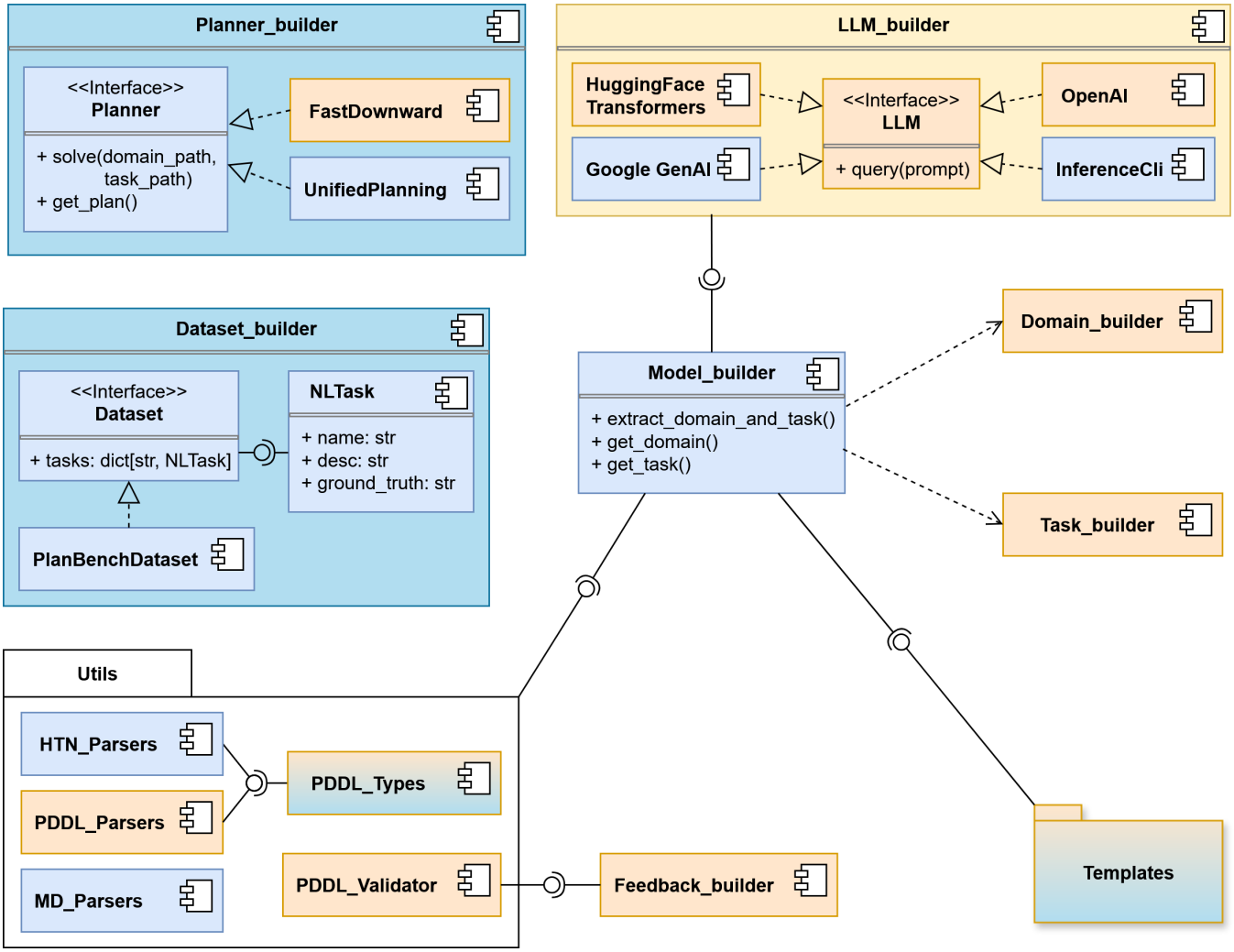

Figure 2: Component Diagram of L2HP. Orange components are native to the L2P library, while blue ones were developed within L2HP. Components with both colors indicate that they were originally part of L2P and have been extended in L2HP.

interface. This interface currently supports the Unified Planning (Micheli et al. 2025) library – highlighting the ARIES planner for HDDL – and maintains compatibility with the Fast Downward planner interface inherited from L2P. The interface is simple, to encourage the future integration of additional planners, especially HTN-specific ones.

Following a similar principle, L2HP extends the LLM interface introduced by L2P, supporting four distinct methods for executing LLMs: the two original ones from L2P: Local inference via HuggingFace Transformers and Remote inference via OpenAI's GPT API; and two new options: The InferenceClient API, for accessing the HuggingFace-hosted LLMs, and Google's GenAI API for the Gemini LLMs.

Datasets integration is also streamlined through a interface conformed by a series of standardized tasks, ensuring consistency and accessibility across future experiments. Figure 4 illustrates an example of an L2HP task. As an initial set of AP tasks described in NL, we have incorporated the PlanBench dataset (Valmeekam et al. 2023). Further details about PlanBench are provided in *Experimentation* section.

L2HP encourages the use of Markdown (MD) as the default output format when building more robust and consistent templates and parsers. Therefore, L2HP provides a built-in parsing toolkit, which includes additional MD utilities to extract sections by title or convert lists into structured data. Markdown is selected for several reasons:

- **Consistency.** LLMs are well-trained on MD and tend to output it more reliably than PDDL, HDDL, or another uncommon or hand-made format.
- **Performance.** MD gives the LLM space to carry out the reasoning or knowledge retrieval process that has already been seen to improve results (Wei et al. 2022).
- **Compatibility.** MD aligns closely with the original L2P representations, facilitating reuse and extension of tools.
- **Extensibility.** MD offers structured formatting that is visually intuitive, making it easier to build new user-defined templates and parsers.

Finally, a new `model_builder` class is provided. It integrates the inherit L2P domain and task builders, preserving all L2P functionalities, but they are also extended to support HTN, as will be illustrated in the next subsection. This builder also provides new functions to export the internal models into multiple formats (PDDL, HPDL, and HDDL).

Hierarchical Planning Integration

To extend L2P to support HTN planning, we introduce the following enhancements:

1. **New Data Types**. In addition to the native L2P structures (Parameters, Predicates, Actions, etc), we provide structures for representing HTN-specific data, such as tasks and methods, both in HPDL and in HDDL syntax.
2. **HTN Parsers**. Custom parsers that extract an store HTN elements from structured LLM output into built-in types.
3. **Prompt Templates**. Designed to guide LLMs in producing output compatible with the built-in parsers.

An integrated extraction method leverages these components to automatically generate complete planning models from NL tasks (steps 1, 2 and 3 of Figure 1). This method is integrated into the new model builder mentioned above. It supports both classical and hierarchical modes, employs all the tools mentioned before, and also serves as a reference for developing new pipelines.

Finally, we present NL2HTN, a simple and illustrative architecture that combine all the provided functionalities to automatically create plans from NL tasks (Figure 1). It supports both PDDL and HDDL modes, and can be instantiated with interchangeable components (e.g., LLMs, planners, templates), aligning with the framework's goal of enabling flexible experimentation.

Experimentation

To highlight the current gap between AP and HP with LLMs, we conducted preliminary experiments using the NL2HTN architecture and the PlanBench dataset. This evaluation has two additional goals: (1) to establish a baseline for LLM-based HP model generation, and (2) to demonstrate a practical use case of L2HP for empirical studies within HP.

PlanBench Dataset

PlanBench (Valmeekam et al. 2023) is a dataset that includes various tasks designed to evaluate planning and reasoning capabilities in NL. We focus on the plan generation subset, which includes 2270 problem instances spanning five different domains. Each instance represents an AP task, and has the following main several components: the **domain** name, a **instance** identifier, a **ground-truth plan** (i.e., a valid plan that solves the problem), and the task **query** itself. The query consists of a prompt with three distinct elements (Figure 3):

1. **NL Domain Definition**. Explicit NL specifications of all available actions, including their constraints.
2. **NL Task Definition**. A description of the initial state and the goal to archive.
3. **One-shot Prompting**. A separate problem instance within the same domain along with a valid solution plan. This approach has been shown to significantly improve LLM performance (Brown et al. 2020).

PlanBench is particularly well-suited for evaluating the modeling capabilities of LLMs, as it provides a diverse set of fully specified planning problems in NL. This setup primarily tests an LLM's ability to accurately interpret detailed task descriptions and convert them into structured representations, with minimal reliance on generating additional information through knowledge retrieval. For this reason, we consider PlanBench the most appropriate benchmark to establish a starting point for research within LLM modeling capabilities. Consequently, it has been integrated into L2HP.

We integrate PlanBench with a focus on the framework's principles of modularity and consistency, aiming to facilitate future extension and integration with other datasets. To this end, we define a standardized dataset structure, into which each PlanBench instance is preprocessed. This structure (Figure 4) is conformed by three key elements: (1) the **task name**, (2) the **ground-truth plan** for evaluation, and (3) the **NL description** of the planning task. Therefore, the domain and problem descriptions are merged into a single task description, while the resolution example is omitted.

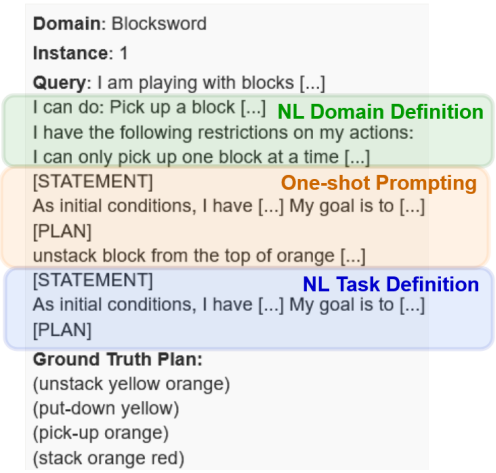

Figure 3: An instance of the PlanBench generation subset.

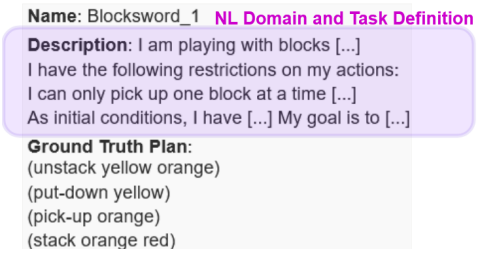

Figure 4: An instance of a standardized task. The task shown is the same as in Figure 3, after being preprocessed to conform to the L2HP standardized dataset structure.

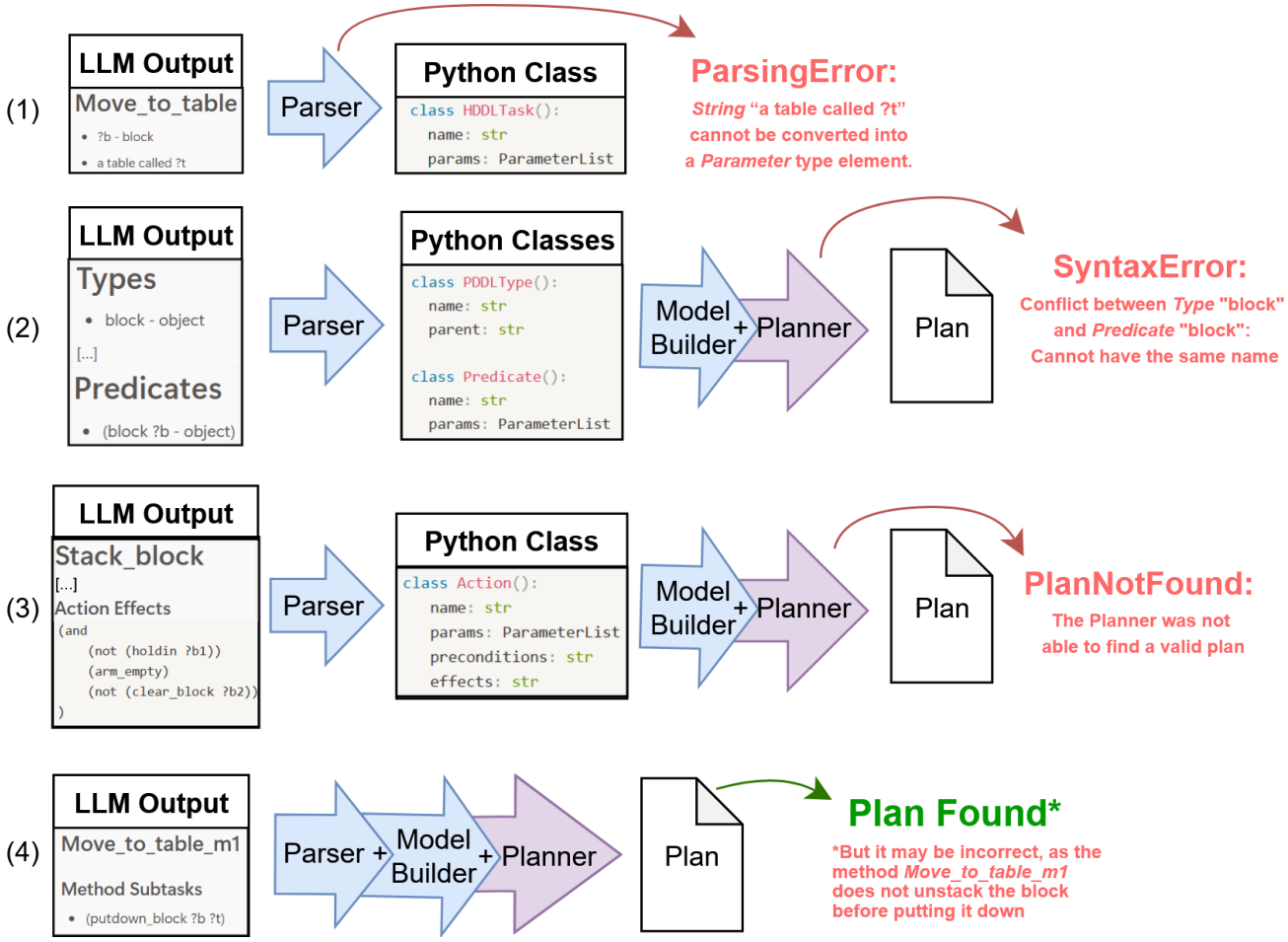

Figure 5: Illustrative examples of different types of invalid LLM outputs. Examples (1) to (3) represents the three core categories of identifiable errors, while (4) shows an incorrect output that does not trigger an error assertion. In (1), the phrase `a table called ?t` lacks a proper structure, so the parser is unable to translate it into a valid parameter. In (2), both the object `block` and the type `block` are parsable, but a conflict arises when invoking the planner. In (3), no plan can be found because the Stack_block action lacks the `(on ?b2 ?b1)` effect. In (4), the planner finds a plan; however, it is likely incomplete, as the method Move_to_table_m1 lacks the `(unstack_block ?b)` subtask. A correct definition is shown in Figure 1.

Experimental Considerations

When executing the NL2HTN pipeline (Figure 1), various types of invalid LLM outputs can arise – each of which should be identify and, ideally, corrected or prevented. Based on the type of error that they generate, we can categorize these invalid outputs into four groups (Figure 5):

1. **Parsing Error.** A parser is unable to build the internal representation of some element. This occurs when output includes noise, incoherent fragments or even valid content but presented in an incorrect structure (Figure 5(1)).
2. **Syntax Error.** The model is parsable, but the planner is unable to load it and initiate the search process. This may result from structural inconsistencies that were not detected by the parsers, or from conflicts between different element definitions (Figure 5(2)).
3. **Plan Not Found.** The planner is able to initiate a search but fails to find a plan. This indicates that the model is syntactically valid but logically incorrect – it does not correspond to the intended task, nor to any solvable task (Figure 5(3)). While this could theoretically occur due to computational limitations, our experiments use feasible, state-of-the-art planning domains, making such failures unlikely in practice.
4. **Incorrect Plan Found.** The planner finds a plan, but it is incorrect. This occurs when the generated model is valid an coherent but does not corresponds to the intended task, or when it contains semantic errors undetectable by either the parser or planner (Figure 5(4)).

While the first three error types are relatively straightforward to detect, the last presents a non-trivial challenge. Generating a plan that differs from the ground-truth does not imply that the model is incorrect, as it may describe an equivalent representation of the same task, with a different action structure. Conversely, reproducing the ground-truth does not

		PDDL			HDDL		
Domain	N	SP	VS	S	SP	VS	S
Blocksworld	600	99	92	88	0	0	0
Mystery	600	169	121	98	423	16	10
Depots	500	237	78	64	343	6	4
Logistics	285	173	102	85	29	1	0
OD_ Logistics	285	79	68	66	108	7	3
Total	2270	757	461	401	903	30	17
Average:		**33.35%**	**20.31%**	**17.67%**	**39.78%**	**1.32%**	**0.75%**

Table 1: Summary of the execution of NL2HTN through the PlanBench plan generation subset. Showing the number of problems (N) per domain and the results of NL2HTN in each mode (PDDL and HDDL). The scores shown are explained in the *Experimentation* section, representing the number of LLM outputs successfully parsed (SP), with a valid syntax (VS), and solvable (S). The final row reports average scores across all instances. Note that each metric is a prerequisite for the next, leading to progressively lower passing rates.

guarantee correctness either, as the model might describe a different task that coincidentally yields the same solution. One might argue that the core problem is the lack of ground-truth models for direct comparison. However, even having such references, it would remain unresolved. A generated model could still represent a valid alternative formulation of the same problem. This highlights a deeper challenge that deserves further investigation: determining whether a generated model faithfully captures the intended problem.

With this limitation in mind, in this preliminary work we focus on the quantifiable aspects of our evaluation. The objective of this experimentation is to establish a baseline, trace the NL2HTN pipeline (Figure 1) and identify its weakness, focusing on the three core errors types that we can find within the execution. Rather than scoring outputs based directly on the error types, we evaluate each execution using three critical checkpoints. While similar, these metrics provide a more informative and interpretable perspective on the execution results:

1. **Successfully Parsed (SP).** The parsers extracted all required elements from the LLM output. This indicates that the LLM output adhered to the expected structure, but it may not represent a valid task.
2. **Valid Syntax (VS).** The LLM generated a valid planning model. However, this model may not correspond to the input task or even be a solvable one.
3. **Solvable (S).** The planner successfully found a plan, indicating that the generated task is logically coherent. However, it may still be semantically incorrect or it may encode a different task than the one provided as input.

We executed NL2HTN across the entire PlanBench subset in both classical (PDDL) and hierarchical (HDDL) planning modes. The specific configuration used was as follows: (LLM) *Gemini 2.0 flash*, via Google GenAI; (planner) *Aries*, through Unified Planning; and (templates) the two hand-made templates available in the `templates/model_templates` directory within the L2HP repository.

Results

Table 1 summarizes the experimental results. Key findings for each of the three checkpoints are highlighted below:

Successfully Parsed (SP). The percentage of SP tasks was similar in both modes – 33% for PDDL and 40% for HDDL. This suggests that the ability of the LLM to follow structural prompts is relatively independent of the underlying planning paradigm. However, parsing errors were the most common failure across both planning modes, affecting more than 60% of all tasks. This highlights the challenges that LLMs face in adhering to strict structure requirements, even when guided by well-constructed templates.

Valid Syntax (VS). A notable disparity exists in the proportion of VS models between classical and hierarchical planning. In PDDL, most SP models were also VS – 20% of all tasks (60% of the 33% SP). By contrast, in HDDL only 1.3% of all tasks (around a 3% of the 40% SP) were syntactically valid. This discrepancy is likely due to the nature of the PlanBench dataset, which provides explicit definitions for classical planning elements but lacks hierarchical annotations. Consequently, the LLM must infer the entire HTN structure from scratch, a task that requires not only syntactic precision but also domain-level reasoning. Although this requirement does not significantly affect structural consistency, as seen in the last paragraph, it has a substantial impact on the capacities of the LLM to generate valid models. Nevertheless, these findings highlight a significant limitation in the current capabilities of LLMs to infer and construct well-formed hierarchical decompositions.

Solvable (S). Most VS models also generate solvable plans. In PDDL, 17% of all tasks were S (85% of the 20% SV). In HDDL, 0.75% of all tasks were S (around 60% of the 1.3% VS). These results highlight the potential of LLMs to generate effective planning models, as long as consistency issues are further addressed. Nevertheless, it is important to remember that finding a valid plan does not necessarily imply that the generated model corresponds to the intended task, as was exposed in the *Experimentation* section.

Discussions and Future Works

The results reveal several challenges in using LLMs to model AP problems – particularly in the context of HP. LLMs struggle with adhering to strict structures and inferring well-formed hierarchical decompositions. Additionally, a core issue in this area remains: the difficulty of automatically verify whether a generated model faithfully represents the intended task. Below, we outline promising directions and strategies to address these limitations.

Inference Capability of LLMs

This is a general limitation that affects many LLM integration areas, not just AP or HP. Therefore, multiple techniques has been developed to enhance LLM reasoning. These includes the already mentioned Chain of Thoughts, few-shot prompting and output refinement, as well as additional strategies such as Tree of Thoughts (Yao et al. 2023), best-of-k sampling, iterative generation, etc. As discussed in the *Background* section, many of these techniques have already been explored in AP. However, there exists still a gap in their application into HP. Therefore, we argue that evaluating and adapting these techniques to HP – alongside their integration into the L2HP framework – is a valuable and necessary direction for future research.

Is Fine-Tuning LLMs Worth It? While fine-tune can be a powerful strategy, we argue it is not the most effective approach for HTN planning. Fine-tuning a LLM requires large task-specific datasets and is both computationally and temporally expensive. Moreover, given the rapid advancement of LLMs development, fine-tuned models risk becoming obsolete quickly. Therefore, leveraging advanced prompting techniques with state-of-the-art models offers a more sustainable and cost-effective direction for this domain.

Structural Consistency of LLMs Output

As with inference, structural consistency is a widespread challenge. It is fundamental to the agentic LLMs area – i.e. LLMs capable of interacting with external tools such as web APIs, databases, or code execution. In this context, some tools such as LangGraph and the Model Context Protocol (mentioned in the *Background* section) have been developed to guide LLMs toward producing consistent executable outputs. In summary, these tools typically combine structured prompting with post-processing pipelines to ensure that the generated outputs are both valid and actionable. We believe that integrating such tools into L2HP, alongside the development of more robust HTN-specific parsing mechanisms, represents a high-valuable direction for future work.

Another promising approach is model repair, where flawed outputs are automatically refined based on feedback. This can be reached by incorporating into the prompt the generated error messages, feedback from some validation tool, or even responses from a secondary LLM. This mechanism is already implemented in the original L2P framework, so extending it to support HTN models would be a natural and impactful improvement.

Correctness of Generated Models

As discussed in the *Experimentation* section, assessing the semantic correctness of generated models remains an open challenge. While a definitive solution is still elusive, two promising directions include: (1) developing HTN-specific benchmarks that pair NL descriptions with ground-truth HTN models, and (2) embedding the architectures within interactive agents that operate in simulated or game environments, enabling a more qualitative evaluation in which correctness is inferred from the agent's behavior.

Is PlanBench a Suitable Choice? PlanBench is one of the few datasets specifically designed for LLM-based planning, and provides a valuable set of AP tasks described in NL. While other datasets exist, most focus on another specific planning-related tasks (e.g. repairing or predicting missing actions) or are derived from PlanBench. Therefore, we consider PlanBench the most suitable choice at the present.

However, we found notable limitations when applying it for HP. The tasks do not provide decompositions information, forcing the LLMs to infer hierarchical structures from flat AP tasks. In addition, the LLM must transform an AP domain into an HTN one, which often results in shallow or flat hierarchies, instead of fully exploiting HP's strengths. Moreover, goals are expressed as sets of conditions, requiring the LLM to reinterpret them into equivalent HTN task formulations rather than directly translating them from NL. These issues highlight the need for new datasets or extensions of PlanBench specifically tailored for HP.

Conclusion

This work explored the applicability gap in the intersection of LLMs and HP. First, we introduced **L2HP**, an extension of the L2P library designed to encourage the LLM integration across the HP life-cycle. Second, we presented a preliminary study of LLM capabilities for model generation in both AP and HP. Beyond benchmarking, our experiments illustrates how L2HP simplifies the development and evaluation of LLM-based planning architectures. We hope that, by offering this modular and extensible framework, L2HP will serve as a valuable tool for the community and encourage further integration between LLMs and HP.

Our results show that LLMs struggle in adhering to strict structural requirements, even when guided by well-designed templates, as parsing errors were the most common failure in both planning modes. Moreover, HP presents further difficulties, being syntactic validity remarkably lower than in AP. Nevertheless, in both settings, most valid models were also solvable, suggesting that – while further research is required – this remains a promising direction.

From these findings, we identify three key research directions: **(1) developing HTN-specific evaluation methods**, such as dedicated benchmarks or embedded evaluation frameworks; **(2) improving output reliability** through advanced prompting techniques, and integrating them into the L2HP framework; **(3) enhance HP modeling capabilities** through more robust, specific parsers and novel techniques that boost LLM performance in constructing HP problems.

References

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; et al. 2020. Language models are few-shot learners. *NeurIPS*, 1877–1901.

Dai, Z.; Asgharivaskasi, A.; Duong, T.; Lin, S.; Tzes, M.-E.; et al. 2024. Optimal scene graph planning with large language model guidance. In *ICRA*, 14062–14069. IEEE.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *AAAI*, 1123–1128.

Fdez-Olivares, J.; Castillo, L.; Garcıa-Pérez, O.; and Palao, F. 2006. Bringing users and planning technology together. Experiences in SIADEX. In *ICAPS*, volume 16, 11–20.

Fine-Morris, M.; Hsiao, V.; Smith, L. N.; Hiatt, L. M.; and Roberts, M. 2024. Leveraging LLMs for Generating Document-Informed Hierarchical Planning Models: A Proposal. In *AAAI 2025 Workshop LM4Plan*.

Geffner, H.; and Bonet, B. 2013. *A concise introduction to models and methods for automated planning*. Morgan & Claypool Publishers.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions. *arXiv preprint arXiv:2405.04215*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *NeurIPS*, 36: 79081–79094.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *AAAI*, 9883–9891.

Hsiao, V.; Fine-Morris, M.; Roberts, M.; Smith, L. N.; and Hiatt, L. M. 2025. A Critical Assessment of LLMs for Solving Multi-step Problems: Preliminary Results. In *AAAI 2025 Workshop LM4Plan*.

Huang, X.; Liu, W.; Chen, X.; Wang, X.; Wang, H.; Lian, D.; Wang, Y.; et al. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716*.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; et al. 2024. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*.

Kienle, C.; Alt, B.; Arenz, O.; and Peters, J. 2025. Hi-TAMP: A Hierarchical LLM-Modulo Planner for Feasible Long-Horizon Task and Motion Planning.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; et al. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Luo, X.; Xu, S.; and Liu, C. 2023. Obtaining hierarchy from human instructions: an llms-based approach. In *CoRL 2023 Workshop LEAP*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; et al. 2024. Self-refine: Iterative refinement with self-feedback. In *the NeurIPS*.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; et al. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Munoz-Avila, H.; Aha, D. W.; and Rizzo, P. 2025. ChatHTN: Interleaving Approximate (LLM) and Symbolic HTN Planning. *arXiv preprint arXiv:2505.11814*.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; et al. 2003. SHOP2: An HTN planning system. *Journal of artificial intelligence research*, 20: 379–404.

Pallagani, V.; Muppasani, B. C.; Roy, K.; Fabiano, F.; Loreggia, A.; Murugesan, K.; et al. 2024. On the prospects of incorporating large language models (llms) in automated planning and scheduling (aps). In *ICAPS*, 432–444.

Puerta-Merino, I.; Núñez-Molina, C.; Mesejo, P.; and Fernández-Olivares, J. 2025. A Roadmap to Guide the Integration of LLMs in Hierarchical Planning. *AAAI 2025 Workshop LM4Plan*.

Silver, T.; Hariprasad, V.; Shuttleworth, R. S.; Kumar, N.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PDDL planning with pretrained large language models. In *NeurIPS*.

Sinha, V. 2024. *Leveraging LLMs for HTN domain model generation via prompt engineering*. Ph.D. thesis, University of Stuttgart.

Tantakoun, M.; Zhu, X.; and Muise, C. 2025. LLMs as Planning Modelers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models. arXiv:2503.18971.

Tianxing, Z.; Zhirui, W.; Haojia, A.; Guangyan, C.; Boyang, X.; Jingwen, C.; et al. 2025. STEP Planner: Constructing cross-hierarchical subgoal tree as an embodied long-horizon task planner. *arXiv preprint arXiv:2506.21030*.

Tse, C. 2024. Improving Human-Robot Communication of Hierarchical Task Planning through LLMs. *Brown Univ.*

Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *NeurIPS*, 38975–38987.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In *NeurIPS*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 24824–24837.

Yang, R.; Zhang, F.; and Hou, M. 2024. Oceanplan: Hierarchical planning and replanning for natural language AUV piloting in large-scale unexplored ocean environments. In *WUWNET*, 1–5.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; et al. 2023. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS*, 11809–11822.

Zhao, Q.; Fu, H.; Sun, C.; and Konidaris, G. 2024. Epo: Hierarchical llm agents with environment preference optimization. *arXiv preprint arXiv:2408.16090*.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artificial intelligence*, 212: 134–157.

PSPACE Planning With Expressivity Beyond STRIPS: Plan Constraints via Unordered HTNs, ILPs, Numerical Goals, and More

Pascal Lauer[1,2], **Yifan Zhang**[1], **Patrik Haslum**[1], **Pascal Bercher**[1]

[1]School of Computing, The Australian National University, Canberra, Australia
[2]Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
firstname.lastname@anu.edu.au

Abstract

To better capture real-world problems, Hierarchical Task Network (HTN) planning and numerical planning provide enhanced modeling capabilities over classical planning. However, the plan existence problem in these formalisms is generally undecidable. We identify restricted fragments that remain PSPACE-complete, matching the complexity of classical planning, while being more expressive. The most important result proves that plan existence in unordered HTN planning, i.e. ignoring all ordering relations, is PSPACE-complete. The result motivates a strong preference for unordered HTN models, a largely ignored fragment that deserves more attention. To bridge the gap between the tractable fragments of numerical and HTN planning, we introduce new formalisms that use Integer Linear Programs, Presburger formulas, and grammatical constraints to express action sequence restrictions within PSPACE, offering practical alternatives when the HTN structure is too complex to model.

1 Introduction

Automated planning is a branch of artificial intelligence focusing on generating operator sequences (plans) from a user-provided problem specification. The specification must be based on a planning formalism. A widely supported formalism is referred to as classical planning (Fikes and Nilsson 1971; Bäckström and Nebel 1995), which is restricted to a set of simple features to make finding a plan more feasible.

While the restriction helps planners run efficiently, users often need more flexibility to model a problem. To address this, there are formalisms extending classical planning. We focus on: (1) Numerical planning (Helmert 2002), allowing numerical variable values instead of propositional ones; and (2) Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996), allowing to constrain plans through a hierarchical task structure. The flexibility comes at a cost. Plan existence in both formalisms is undecidable (Helmert 2002; Erol, Hendler, and Nau 1996), but PSPACE-complete for classical planning (Erol, Nau, and Subrahmanian 1995).

It is clearly desirable to have a formalism that combines both benefits, i.e., remains computationally tractable while offering greater modeling flexibility. A good example, showing that this is possible, is totally ordered HTN planning, which imposes a pre-defined order between all tasks. The restriction still allows modeling a meaningful hierarchical structure, but drops the complexity of plan existence to EXPTIME (Erol, Hendler, and Nau 1996). The drop in complexity is reflected in practice. E.g., the results of the HTN track in the latest International Planning Competition (Taitler et al. 2024) show a clear gap between the ability to solve totally ordered versus partially ordered problems.

In this paper, we show that there exists an alternative ordering constraint, namely unordered, which does not allow any order at all, and reduces the complexity of plan existence to PSPACE. To reach this result, we reconsider a fragment of numerical planning for which Helmert (2002) established decidability. It allows only numerical goal conditions, but no numerical preconditions. We improve the result by showing that plan existence in this fragment is PSPACE-complete. The fragment is closely related to HTN planning as the added constraint, imposed on plans by the hierarchical structure, closely compares to the added constraint from the numeric goal. To formalize this connection, we introduce three new planning formalisms that add constraints on classical planning plans through: (1) Integer Linear Programs, (2) Presburger formulas and (3) Counting constraints over context-free grammars. We prove PSPACE membership by establishing a chain of encodings from unordered HTN planning through the formalisms (3) to (1), ending in the numerical fragment. Hardness follows from the fact that these formalisms extend classical planning.

To objectively show that each of the five formalisms meaningfully extends classical planning, we use a language analysis (Höller et al. 2014; 2016; Lin and Bercher 2022). The set of plans for a planning problem is its language, and the language class of a formalism groups all such languages for problems it can represent. The analysis nicely complements our complexity results, as it favors bigger classes, including more languages, whereas complexity theory favors lower classes. To demonstrate that our formalisms are significantly more expressive, we show they include the classical planning language as well as each a regular, context-free, and context-sensitive language that classical planning can not express.

Our results strongly suggest moving away from partial-order HTN models in favor of unordered ones and point at new alternative formalisms that can be used when other formalisms make modeling too complicated.

2 Background

Numerical (and Classical) Planning In numerical planning, operator preconditions and effects are evaluated through rational functions. Restrictions on these functions impact the complexity of plan existence (Helmert 2002). Therefore, we fix families of rational functions to define distinct numerical planning formalisms.

Definition 2.1 (Numerical Planning Formalism)**.** A function $f : \mathbb{Q}^n \to \mathbb{Q}$ is called an n-ary *rational function*. A *family of rational functions* F is a set of rational functions, i.e., $F \subseteq \bigcup_{n\geq 1}(\mathbb{Q}^n \to \mathbb{Q})$. A *numerical planning formalism* (G, P, E) consists of three families of rational functions G, P, E. Here G contains the *goal condition functions*, P the *precondition functions*, and E the *effect functions*.

As we build on results by Helmert (2002) in Section 4, we also reproduce his numerical planning problem definition.

Definition 2.2 (Numerical Planning Problem)**.** A *numerical planning problem* $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ over numerical planning formalism (G, P, E) consists of: The *propositional variables* V_P and *numerical variables* V_N, which are disjoint finite sets. The *states* S are:

$$S := \{(\alpha, \beta) \mid \alpha : V_P \to \{\bot, \top\},\ \beta : V_N \to \mathbb{Q}\}.$$

For each state $(\alpha, \beta) \in S$, α is called *propositional state*. The set of all propositional states is denoted by S_α. β is called *numerical state*. The set of all numerical states is denoted by S_β. Init is a state, called *initial state*.

A *propositional condition* is a propositional variable $v \in V_P$, also written as $v = \top$. For a family of rational functions $F \in \{G, P\}$, a *numerical condition* over F is given by an n-ary function $f \in F$, numerical variables $v_1, \ldots, v_n \in V_N$, and a relational operator $\mathit{relop} \in \{=, <, \leq, \geq, >, \neq\}$ denoted as $f(v_1, \ldots, v_n)\ \mathit{relop}\ 0$.

A *propositional effect* is given by a variable $v \in V_P$ and a truth value $t \in \{\top, \bot\}$, written as $v \leftarrow t$. A *numerical effect* over E is given by a function $f \in E$ and variables $v_1, \ldots, v_n \in V_N$. It is written as $v_1 \leftarrow f(v_2, \ldots, v_n)$. We say the effect *assigns* v_1. An operator over $o = (\mathit{pre}, \mathit{eff})$ consists of two finite sets pre and eff, where: pre contains propositional conditions and numerical conditions over P and eff contains propositional effects eff_P and numerical effects eff_N over E with pairwise distinct assigned variables. Ops is the finite set of operators over P and E of Π. The *goal condition* Goal is a finite set containing propositional and numerical conditions over G.

$\mathcal{NUM}(G, P, E)$ is the set of all numerical planning problems over numerical planning formalism (G, P, E).

Helmert (2002) defines plans using the full transition graph. For brevity, we only define the progressive successor states using the notation by Hoffmann and Nebel (2001).

Definition 2.3 (Numerical Plans)**.** For a numerical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ over numerical planning formalism $F = (G, P, E)$, a propositional condition $v \in V_p$ is *satisfied* in propositional state $\alpha \in S_\alpha$ iff $\alpha(v) = \top$. A numerical condition $f(v_1, \ldots, v_n)\ \mathit{relop}\ 0$ with $f \in G \cup P$ is satisfied in numerical state $\beta \in S_\beta$ iff $f(\beta(v_1), \ldots, \beta(v_n))\ \mathit{relop}\ 0$.

An operator $o = (\mathit{pre}, \mathit{eff}) \in \mathit{Ops}$ is *applicable* in propositional state $\alpha \in S_\alpha$ iff all propositional conditions in pre are *satisfied* in α. In this case the *propositional successor* state is the state replacing the truth assignments of α by eff:

$$\mathrm{progr}(\alpha, o) := \{v \mapsto t \in \alpha \mid \nexists v \leftarrow t' \in \mathit{eff}_P\} \cup \mathit{eff}_P$$

Otherwise $\mathrm{progr}(\alpha, o)$ is undefined.

Further, o is *applicable* in numerical state $\beta \in S_\alpha$ iff all numerical conditions in pre are satisfied. In this case we set the *numerical successor* as:

$$\mathrm{progr}(\beta, o) := \{v \mapsto n \in \beta \mid \nexists e \in \mathit{eff}_N \text{ assigning } v\}$$
$$\cup\, \{v_1 \mapsto f(\beta(v_2), ..., \beta(v_n)) \mid v_1 \leftarrow f(v_2, ..., v_n) \in \mathit{eff}_N\}$$

Otherwise $\mathrm{progr}(\alpha, o)$ is undefined.

Finally, o is *applicable* in state (α, β) if it is applicable in both α and β. In this case we denote the *progressive successor state* as:

$$\mathrm{progr}((\alpha, \beta), o) := (\mathrm{progr}(\alpha, o), \mathrm{progr}(\beta, o))$$

Otherwise $\mathrm{progr}((\alpha, \beta), o)$ is undefined.

We denote $\mathrm{progr}(\mathrm{progr}(...\,\mathrm{progr}(s, o_1), ...), o_n)$ by $\mathrm{progr}(s, o_1, ..., o_n)$. A plan is an operator sequence $o_1, \ldots, o_n$ so that $\mathrm{progr}(\mathit{Init}, o_1, \ldots, o_n) =: (\alpha', \beta')$ so that all propositional conditions in Goal are satisfied in α' and all numerical conditions in Goal are satisfied in β'. $\mathrm{sol}(\Pi)$ denotes the set of all plans in Π.

We now replicate Helmert's list of conditions, excluding polynomial functions, which are beyond this paper's scope.

Definition 2.4 (Common Numerical Conditions)**.**

1. $C_\varnothing := \varnothing$
2. $C_0 := \{x \mapsto x\}$
3. $C_c := \{x \mapsto x - c \mid c \in \mathbb{Q}\}$
4. $C_= := \{(x_1, x_2) \mapsto x_1 - x_2\}$

$C_\varnothing$ corresponds to no preconditions. C_0 compares a variable with zero. C_c compares a variable with a constant. $C_=$ compares two variables. For section 4, it will be important to restrict preconditions to $C_\varnothing$, so we give this a name.

Definition 2.5 (No Numerical Preconditions)**.** A numerical planning problem over formalism $(G, C_\varnothing, E)$ with arbitrary families of rational function G and E is said to have *no numerical operator preconditions*.

We also focus on all effects Helmert (2002) uses, except polynomial functions. Helmert describes 10 classes. We summarize them by listing the effects that can add/subtract $E_{\pm c}$, assign constants $E^{=c}$, and their combination $E^{=c}_{\pm c}$

Definition 2.6 (List of Numerical Effects)**.**

1. $E_{\pm c} := \{x \mapsto x + c \mid c \in \mathbb{Q}\}$
2. $E^{=c} := \{x \mapsto c \mid c \in \mathbb{Q}\}$
3. $E^{=c}_{\pm c} := E_{=c} \cup E_{\pm c}$

$E^{=c}_{\pm c}$ includes all other non-polynomial effects Helmert lists.

Definition 2.7 (Constant Effects)**.** A numerical planning problem over numerical planning formalism (G, C, E) so that $E \subseteq E^{=c}_{\pm c}$ is said to have *constant effects*.

Finally, observe that classical planning formalisms, like SAS$^+$ (Bäckström and Nebel 1995) or FDR (Helmert 2009), match our formalism if there are no numerical functions.

Definition 2.8 (Classical Planning)**.** The numerical planning formalism $(C_\varnothing, C_\varnothing, C_\varnothing)$ is called classical planning. We will also refer to $\mathcal{NUM}(C_\varnothing, C_\varnothing, C_\varnothing)$ as $\mathcal{CLASSIC}$.

Unordered HTN Planning We introduce HTN planning based on Geier and Bercher (2011), but restrict it to unordered HTN planning. This means that we remove ordering constraints from any task network in the planning problem.

Definition 2.9 (Unordered HTN planning problem)**.** An *unordered HTN planning problem* $\Pi_{\mathcal{H}} = (\Pi, \mathcal{H})$, consists of a classical planning problem $\Pi = (V_P, V_N, Ops, Init, Goal)$ and the hierarchy $\mathcal{H} = (tn_I, \mathcal{M}, \mathcal{C})$ modeling hierarchical restrictions to operator solutions. $\mathcal{C}$ is the set of *compounds tasks*. The set of operators Ops is also called *primitive tasks* in this context. A *task network* $tn = (T, \alpha)$ consists of a set of *task IDs* T and a map $\alpha : T \to Ops \cup \mathcal{C}$.

A *method* $m = (c_m, tn_m)$ allows to replace a compound task $c_m \in \mathcal{C}$ with the tasks from task network $tn_m = (T_m, \alpha_m)$. In particular, m defines a relation $tn_1 \to_m tn_2$ between task networks $tn_1 = (T_1, \alpha_1)$ and tn_2, iff there exists a task identifier $tid \in T_1$ such that $\alpha_1(tid) = c_m$, a bijection $\sigma : T_m \to T'$, where T' is a set of fresh task identifiers not in T_1, so that the resulting task network tn_2 is given by:

$$tn_2 = (T_1 \setminus \{tid\} \cup \sigma(T_m), \alpha_1 \setminus \{tid \mapsto c_m\} \cup \sigma^{-1} \circ \alpha_m)$$

$\mathcal{M}$ denotes the set of all methods in $\Pi_{\mathcal{H}}$. The relation $\to_{\mathcal{M}}$ is defined as the union of all equivalence relations $\to_m$, for all $m \in \mathcal{M}$. The transitive closure is denoted by $\to^*_{\mathcal{M}}$. A task network (T, α) is called primitive iff all of its tasks T are primitive. The *solution set* of the hierarchy $\mathcal{H}$ are the primitive task networks that can be derived by decomposition from the initial task network, i.e.:

$$\mathrm{sol}(\mathcal{H}) := \{tn \mid tn_I \to^*_{\mathcal{M}} tn, tn \text{ is primitive}\}$$

A *linearization* of a primitive task network is an arbitrarily ordered sequence of all elements $\alpha(tid)$ for $tid \in T$. A *solution* to an unordered HTN planning task is a set of primitive task networks obtained by decomposing tn_I and have a linearization that is a classical planning solution, i.e.:

$$\mathrm{sol}(\Pi_{\mathcal{H}}) = \{tn \in \mathrm{sol}(\mathcal{H}) \mid \exists \pi \in \mathrm{sol}(\Pi) : \pi \text{ is a linearization of } tn\}$$

$\mathcal{HTN}_{\mathcal{U}}$ is the set of all unordered HTN planning problems.

The solution definition clearly shows that plans in HTN planning must meet two criteria: (1) follow the hierarchy, and (2) solve a classical planning task correctly. In sections 5, 6, and 7, we introduce new formalisms that keep this solution structure but replace (1) with different constraints.

Context-Free Grammars

Definition 2.10 (Context-Free Grammar)**.** A *context-free grammar* $G = (N, \Sigma, S, R)$ consists of a finite set of *nonterminals* N, a finite set *terminals* Σ, disjoint from N, the *start symbol* $S \in N$ and a finite set of *production rules* R. A production rule is of the form $A \to \alpha_1 | ... | \alpha_n$, where $A \in N$ is a non-terminal and all $\alpha_1, ..., \alpha_n \in (N \cup \Sigma)^*$ are finite sequences of terminals and/or non-terminals.

The language of G are all words derived by repeatedly applying production rules until only terminals remain.

Definition 2.11 (Grammar Language)**.** For a context-free grammar $G = (N, \Sigma, S, R)$, a sequence $s = s_0 s_1 ... s_n \in (N \cup \Sigma)^*$ can *derive* a sequence $t \in (N \cup \Sigma)^*$, denoted $s \Rightarrow t$, by replacing some symbol s_i for $i \in \{0, ..., n\}$ with α_j for some $j \in \{1, ..., m\}$ from a production rule $s_i \to \alpha_1 \mid \cdots \mid \alpha_m$, in R. The relation $\Rightarrow^*$ is the transitive closure of $\Rightarrow$. A *word* (or *string*) is a sequence over Σ^*. The *language* generated by G, denoted $L(G)$, is the set of all words derived from the start symbol:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

Planning Problem Languages A planning problem language represents the set of valid plans. By default terminals are operators whose structure depends on the formalism. To compare languages across different formalisms, we use *exchange functions* that map operators to arbitrary terminals.

Definition 2.12 (Exchange Function)**.** For operators Ops the set $\mathrm{exch}(Ops)$ contains all bijections $\sigma : Ops \to \Sigma$ for any terminals Σ. We call its elements *exchange function*.

A language for non-HTN planning problems is a plan where each operator is replaced using a fixed exchange function.

Definition 2.13 (Non-HTN Planning Language)**.** For a planning problem Π with operator sequences over operators Ops as solutions, the *language* of Π under $\sigma \in \mathrm{exch}(Ops)$ is:

$$L_\sigma(\Pi) := \{\sigma(\pi) \mid \pi \in \mathrm{sol}(\Pi)\}.$$

For HTN planning, we adapt the definition to consider a linearization of the primitive task solution.

Definition 2.14 (HTN Planning Language)**.** Let $\Pi_{\mathcal{H}}$ be an HTN planning problem with operators Ops. The *language* of $\Pi_{\mathcal{H}}$ under $\sigma \in \mathrm{exch}(Ops)$ is:

$$L_\sigma(\Pi_{\mathcal{H}}) = \{\sigma(\pi) \mid \exists tn \in \mathrm{sol}(\mathcal{H}) : \pi \in \mathrm{sol}(\Pi) \text{ is a linearization of } tn\}$$

A language class captures all languages of a set of planning problems under any fixed exchange function.

Definition 2.15 (Language class)**.** For a set $\mathcal{P}$ of planning problems, the *language class* is defined as:

$$L(\mathcal{P}) := \{L_\sigma(\Pi) \mid \Pi \in \mathcal{P}, \sigma \in \mathrm{exch}(Ops)\}$$

Integer Linear Programs (ILPs) We define ILPs following Papadimitriou (1981). In our notation $\mathbb{N}$ includes 0.

Definition 2.16. An integer linear program $\mathbb{L} = (M, v)$ is a combination of a matrix $M \in \mathbb{Z}^{m \times n}$ and a vector $v \in \mathbb{Z}^m$. The *solution set* of $\mathbb{L}$ is $\mathrm{sol}(\mathbb{L}) := \{x \in \mathbb{N}^n \mid Mx = v\}$.

3 Formal Criteria for Increased Expressivity

To capture *better expressivity*, we ensure that: A formalism must (1) subsume classical planning, and (2) admit languages beyond the expressive power of classical planning.

Definition 3.1. Let $\mathcal{P}$ be a set of planning problems. We say that $\mathcal{P}$ is *significantly more expressive* than classical planning if all of the following statements hold:

- $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{P})$
- $\{aa\}, \{a^n b^n \mid n \in \mathbb{N}\}, \{a^n b^n c^n \mid n \in \mathbb{N}\} \in L(\mathcal{P})$

where the languages in the latter bullet points are defined over the terminals $\Sigma = \{a, b, c\}$.

The language class for classical planning $L(\mathcal{CLASSIC})$ is a strict subset of regular languages (Höller et al. 2014; 2016). As $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ is a context-sensitive language, and $\{a^n b^n \mid n \in \mathbb{N}\}$ is a context-free language, they are not in $L(\mathcal{CLASSIC})$. $\{aa\}$ is an example of a regular language that is not in $L(\mathcal{CLASSIC})$. The additionally included languages establish $L(\mathcal{CLASSIC}) \subsetneq L(\mathcal{P})$. The inclusion of a regular, context-free and context-sensitive language, members of three and widely studied language classes, demonstrates a significant increase in expressivity.

The three languages also showcase that increased expressivity significantly improves modeling capabilities: In particular, they showcase that a modeler can require operators to occur equally often, without fixing an upper bound. This is impossible in classical planning and relevant in practice. E.g., in a financial market one may require money-earning operators a to be followed by the same number of money-spending operators b. While we keep our analysis to single operators to fit space constraints, our analysis naturally extends to multiple operators, like the language represented by:

$$(\mathit{earn}_1 \mid ... \mid \mathit{earn}_m)^n (\mathit{spend}_1 \mid ... \mid \mathit{spend}_l)^n$$

In the following sections, we show that each formalism has better modeling capabilities than classical planning, as they are significantly more expressive, while deciding plan existence remains PSPACE-complete.

4 Numerical Planning With No Numerical Preconditions And Constant Effects

In this section, we analyze numerical planning without numerical preconditions and restrict effects to increase, decrease, or assign fixed constants. The goal conditions are comparisons between two variables, or one variable with a constant. This fragment is closely related to (unordered) HTN planning, as it allows counting operator applications and applying conditions on these counts.

We first focus on the complexity of plan existence, which we show to be PSPACE-complete[1]. We build on results of Helmert and so paraphrase the relevant parts of their work in the following, mainly surrounding Helmert's Algorithm 22, which creates plans of a specific format. To explain the format, we introduce properties of operator sequences (paths) based on the propositional states they traverse. In particular, whether states are visited only once or revisited, which would make the path a cycle.

Definition 4.1. For a numerical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ a *path* from s_0 to s_n is a sequence of operators $o_1, ..., o_n \in \mathit{Ops}$ resulting in a state $s_i = \mathit{progr}(o_i, s_{i-1})$ for $i \in \{1, ..., n\}$.

Let $s_i = (\alpha_i, \beta_i)$ for $i \in \{0, ..., n\}$. If all states $\alpha_0, ..., \alpha_n$ are pairwise distinct, the path is called *propositionally acyclic*. If $\alpha_0, ..., \alpha_{n-1}$ are pairwise distinct and $\alpha_0 = \alpha_n$, the path is called a *propositional cycle* for α_0.

A path is *weakly acyclic* if it can be partitioned into subsequences of paths $p_1, ..., p_m$ where p_1 is an acyclic path from s_0 to s_n, and each p_j with $j \in \{2, ..., m\}$ is a propositional cycle for some α_i with $i \in \{0, ..., n\}$.

We now define the plan format as activator path. This is a concatenation of weakly acyclic paths, each ending with an assignment to a numerical variable. After this assignment, the variable is only modified by increments or decrements.

Definition 4.2. An *activator path*[2] $p = p_1, ..., p_n$ for pairwise distinct variables $v_1, ..., v_n \in V_N$ is a sequence of weakly acyclic paths p_i, each ending with an assignment to v_i. No v_i is assigned in any later path p_j with $1 \leq i < j \leq n$.

Helmert (2002) observes that one can always represent plans in this format, as: To satisfy propositional goals it suffices to follow an acyclic path. Since there are no numerical preconditions, it is enough to ensure applicability over propositional states and only consider the final numerical goal values. These values are determined by a single assignment followed by increments or decrements.

Lemma 4.3. *(Helmert 2002) Let Π be a numerical planning problem with no numerical preconditions and only constants effects with $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$.*

If there exists a plan for Π, then there exists a plan for Π that is an activator path for some $v_1, ..., v_n \in V_N$.

Helmert (2002) guess a weakly acyclic path and then determine how many times a cycles need to be repeated to obtain a plan. We capture these repetitions by cycle extensions.

Definition 4.4. For a path $p = p_1, p_c, p_2$, where p_c is a propositional cycle and p_1, p_2 are (possibly empty) paths, p_1, p_c, p_c, p_2 is a *cycle extension*.

To determine the required cycle extensions, Helmert (2002) constructs an ILP. The idea is to introduce a constraint for the goal value of each numerical variable $v \in V_N$:

$$\mathit{goal}_v = \mathit{activation}_v + \sum_{c \in \mathit{Cycles}} (\Delta_v(c) \cdot x_c)$$

Here, $\mathit{activation}_v$ is the value assigned by the activator, $\Delta_v(c)$ is the value change to v after one execution of cycle c, and x_c is the number of times c is repeated. By adding the numerical goal constraints, the required cycle extensions can be bounded by the sum of all x_c.

Lemma 4.5. *(Helmert 2002) Let p be an activator path for variables $v_1, ..., v_n \in V_N$ in a numerical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ with no numerical preconditions and only constant effects.*

There exists an ILP formulation that computes the number of cycle extensions needed to turn p into a plan, if possible, and is unsolvable otherwise. The ILP has a polynomial number of rows in the size of Π and a number of columns polynomial in the size of Π and the number of cycles in p.

[1] Dekker and Behnke (2024) already claim this result in their Table 3, citing Helmert (2002). While Helmert establishes decidability, there is no PSPACE-completeness proof. To the best of our knowledge also not in other literature. We confirmed this with the main author of Dekker and Behnke (2024). As no proof appears in the literature, we provide it in our work.

[2] Helmert (2002) refer to the last assigning the values as activator sequence. For brevety we combine this with the operators leading up to it as activator path.

This summarizes the main results we needed to restate in order to prove that we can exponentially bound plans.

Lemma 4.6. *Let* Π *be a numerical planning problem with no numerical preconditions and only constants effects with* $\Pi = (V_P, V_N, Init, Goal, Ops)$.

If there exists a plan for Π*, then there exists a plan for* Π *which consists of at most exponentially-many operators bounded in the size of* Π.

Proof. We first bound the number of distinct cycles in a weakly acyclic path: As addition is commutative, the order of operators in a cycle does not matter, only the number of times each operator occurs in it. Thus, cycles with the same underlying multiset of operators are equivalent in this context. Each such multiset has $|Ops|$ entries, each bounded by the maximum cycle length $|S_\alpha| \leq 2^{|V_P|}$. Therefore the number of distinct cycles is at most $(2^{|V_P|})^{|Ops|} = 2^{|V_P| \cdot |Ops|}$, which is exponential in the size of Π. As both cycles and acyclic paths are bounded by $\leq 2^{|V_P|}$, any weakly acyclic path, where each cycle occurs at most once, is exponentially length-bounded in the size of Π.

Assume a plan exists. By Lem. 4.3, there is an activator path that is a plan. We can construct it by taking an activator path where each cycle occurs at most once, and then applying cycle extensions. The activator path consists of at most $|V_N|$ weakly acyclic paths and thus is also exponentially length-bounded in the size of Π. It remains to bound the operators added by the cycle extensions.

This follows from Papadimitriou (1981), who shows that if an ILP has polynomially many rows in $n \in \mathbb{N}$, and exponentially many columns in n, then if there is a solution, there is one with values bounded exponentially in n. Combined with Lem. 4.5 this implies that the number of cycle extensions required for an activator-path p is exponentially bounded in Π. As the size of each cycle is exponentially bounded ($\leq 2^{|V_P|}$), the cycle extensions leading to a plan add at most an exponential operators in the size of Π. □

Theorem 4.7. *Plan Existence for numerical planning problems over* $(G, C_\varnothing, E)$ *is PSPACE-complete for all:*

$$G \in \{C_\varnothing, C_0, C_c, C_=\} \text{ and } E \subseteq E^{=c}_{\pm c}$$

Proof. PSPACE-hardness follows directly from classical planning, which is PSPACE-complete (Erol, Nau, and Subrahmanian 1995) and captured as a syntactic fragment of the formalisms considered.

For membership we can determine plan existence by a guess-and-check procedure in NPSPACE = PSPACE, since all plans are exponentially length-bounded. □

We now turn to expressivity. We show that numerical planning with additive effects and equality tests can define languages beyond those expressible by classical planning. To ease the upcoming proofs, we introduce a simple classical planning task as a gadget. It has three operators a, b, and c, which must be applied in this order but may be repeated multiple times, before switching to the next operator type. We reuse this gadget throughout the expressivity proofs in the following sections.

Definition 4.8 (Sequential Operator Gadget)**.** The classical planning problem $\Pi^{\text{Seq}} = (V_P, V_N, Init, Goal, Ops)$ encodes sequence constraints among operators a, b, and c using: The set of propositional variables $V_P = \{apply\_a, apply\_b, apply\_c\}$. The empty set of numerical variables $V_N = \varnothing$. An empty goal condition $Goal = \varnothing$. The initial state $Init = (\alpha, \beta)$, setting the propositional parts to true, i.e., $\alpha(apply\_a) = \top, \alpha(apply\_b) = \top, \alpha(apply\_c) = \top$ and has an empty numerical part $\beta = \varnothing$. Finally, the operator set Ops contains the following operators:

- Operator $a = (pre_a, eff_a)$:
  $pre_a = \{apply\_a\}$, $eff_a = \varnothing$
- Operator $b = (pre_b, eff_b)$:
  $pre_b = \{apply\_b\}$, $eff_b = \{apply\_a \leftarrow \bot\}$
- Operator $c = (pre_c, eff_c)$:
  $pre_c = \{apply\_c\}$, $eff_c = \{apply\_a \leftarrow \bot,\ apply\_b \leftarrow \bot\}$

We can now prove the expressivity result for the numerical planning fragment.

Theorem 4.9. $\mathcal{NUM}(C_=, C_\varnothing, E)$ *is is significantly more expressive than classical planning for* $E \subseteq E^{=c}_{\pm c}$.

Proof. By definition, $\mathcal{CLASSIC} \subseteq \mathcal{NUM}(C_=, C_\varnothing, E)$, so it also holds that $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{NUM}(C_=, C_\varnothing, E))$.

To show that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are in $L(\mathcal{NUM}(C_=, C_\varnothing, E))$, we extend the construction $\Pi^{\text{Seq}} = (V_P, V_N, Init, Goal, Ops)$ from Def. 4.8 to a numerical planning task $\Pi_N = (V_P, V'_N, Ops', Init', Goal')$. The numeric variables $V'_N = \{v_a, v_b, v_c, v_0, v_2\}$ track how often an operator was applied and simulate constants in the goal condition. We initially set the counters to zero, i.e., $Init' = (\alpha, \beta)$ where α is the propositional state of $Init$ in Π^{Seq} and $\beta = \{v_a \mapsto 0, v_b \mapsto 0, v_c \mapsto 0, v_0 \mapsto 0, v_2 \mapsto 2\}$. Finally, we consider operators $Ops' = \{a', b', c'\}$, extending the original operators $Ops = \{a, b, c\}$ as:

- $a' = (pre_a, eff'_a)$, $eff'_a = eff_a \cup \{v_a \leftarrow v_a + 1\}$
- $b' = (pre_b, eff'_b)$, $eff'_b = eff_b \cup \{v_b \leftarrow v_b + 1\}$
- $c' = (pre_c, eff'_c)$, $eff'_c = eff_c \cup \{v_c \leftarrow v_c + 1\}$

Each operator now increases its count to track how often the operator was applied. We define the goals separately for the two languages.

(1) For Language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ we define the goal:

$$Goal' := (v_a - v_b = 0) \wedge (v_b - v_c = 0)$$

(2) For Language $\{a^n b^n \mid n \in \mathbb{N}\}$ we define the goal:

$$Goal' := (v_a - v_b = 0) \wedge (v_c - v_0 = 0)$$

(3) For Language $\{aa\}$ we define the goal:

$$Goal' := (v_a - v_2 = 0) \wedge (v_b - v_0 = 0) \wedge (v_c - v_0 = 0)$$

Goal (1) requires $v_a = v_b \wedge v_b = v_c$ and so only accepts sequences with equal counts of a, b, and c. Goal (2) requires $v_a = v_b \wedge v_b = 0$ and so only accepts sequences with equal counts of a, b, but no occurrence of c. Goal (3) requires $v_a = 2 \wedge v_b = 0 \wedge v_c = 0$ and so only accepts the sequence aa. □

5 Classical Planning with Linear Constraints

In this and the next two sections, we introduce new planning formalisms that extend classical planning by imposing different solution constraints. This section focuses on Integer Linear Programs (ILPs). We use the numerical fragment from the last section for membership proofs here, creating the first link in a chain connecting all presented formalisms.

To restrict solutions using an ILP, we count the number of occurrences of each operator in a sequence. In language theory, this corresponds to the well-known concept of the Parikh vector, which we then require to satisfy the ILP.

Definition 5.1. Let Σ is a finite set of terminals. The *Parikh vector* $\Psi(w)$ of a sequence $w = e_1 e_2 \dots e_{|w|}$ over the terminals Σ is a vector in $\mathbb{N}^{|\Sigma|}$, where each entry counts the occurrences of the corresponding terminal in w. I.e., for position $i \in \{1, ..., |\Sigma|\}$:

$$(\Psi(w))_i = \sum_{j=1}^{|w|} \begin{cases} 1, \text{ if } \mathrm{id}(e_j) = i \\ 0, \text{ otherwise} \end{cases}$$

For some unique bijective mapping $\mathrm{id} : \Sigma \to \{1, ..., |\Sigma|\}$. For $e \in \Sigma$ we represent $(\Psi(w))_{\mathrm{id}(e)}$ with shorthand $(\Psi(w))_e$.

In the context of planning problems the terminals $\Sigma = Ops$ are simply the operators. We now define the new planning formalism where the solutions are restricted by an ILP.

Definition 5.2 (Planning with Linear Constraints)**.** A *planning problem with linear constraints* $\Pi_L = (\Pi, \mathbb{L})$ consists of a planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$ and an ILP $\mathbb{L}$ with $|Ops|$ columns. The plans for Π_L are:

$$\mathrm{sol}(\Pi_L) := \{\pi \in \mathrm{sol}(\Pi) \mid \Psi(\pi) \in \mathrm{sol}(\mathbb{L})\}$$

The set of all planning problems with linear constraints is denoted by $\mathcal{ILP}$.

Linear constraints over action counts, even in a very simple form, are useful for modeling (Lauer 2025). E.g., different money-earning and money-spending operators may earn or spend different amounts of money. A linear constraint can express the exact amounts $w_i, w'_i \in \mathbb{Q}$ to ensure a profit:

$$w_1 \cdot spend_1 + ... + w_l \cdot spend_l \;\leq\; w'_1 \cdot earn_1 + ... + w'_m \cdot earn_m$$

We now show that plan existence for this formalism is PSPACE-complete by encoding it into numerical planning.

Theorem 5.3. *Deciding whether there exists a plan for a classical planning problem with linear constraints is PSPACE-complete.*

Proof. PSPACE-hardness follows by reduction from classical planning, using the linear constraint $0x = 0 \equiv$ true.

For membership, we encode classical planning problem with linear constraints $\Pi_L = (\Pi, (M, v))$ with $M \in \mathbb{Z}^{m \times n}$ into numeric planning formalism $(C_=, C_\varnothing, E_{\pm c})$, which is PSPACE-complete by Thm. 4.7. If more expressive goal conditions (e.g., supporting multiplication) were allowed, one could simply count operator applications and verify $Mx = v$ line by line in the goal, as in Thm. 4.9. Without such expressiveness, we instead simulate the constraint via auxiliary variables and operators. Specifically, for each constraint line $i \in \{1, \dots, m\}$, we introduce one variable to track each term $M_{i,j} \cdot x_j$ using constant effects, and n auxiliary variables $c_{i,j}$ such that their sum must equal v_i, i.e., $\sum_{j=1}^{n} c_{i,j} = v_i$, enforced by the goal. The goal further requires that each $c_{i,j}$ matches the corresponding value tracking $M_{i,j} \cdot x_j$. The goal conditions imply the original linear constraint $Mx = v$. To allow reaching a valid assignment of values to the $c_{i,j}$ that satisfies the condition, we add distribution actions that transfer units between these variables. This ensures that the constraint can be satisfied if and only if the original system $Mx = v$ holds. □

Theorem 5.4. *$\mathcal{ILP}$ is is significantly more expressive than classical planning.*

Proof. The hardness encoding from classical planning in the proof of Thm. 5.3 shows that $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{ILP})$ by allowing arbitrary sequences without constraints. To prove that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are included we extend the sequential planning task Π^{Seq} from Def. 4.8 to a planning problem with linear constraints $\Pi_L = (\Pi, \mathbb{L})$ with $\mathbb{L} = (M, v)$ and:

(1) $M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$, $v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ for $\{a^n b^n c^n \mid n \in \mathbb{N}\}$

(2) $M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ for $\{a^n b^n \mid n \in \mathbb{N}\}$

(3) $M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $v = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$ for $\{aa\}$

(1) enforces $x_a = x_b \wedge x_b = x_c$, so only plans with equal counts of a, b, c are accepted. (2) enforces $x_a = x_b \wedge x_c = 0$, so only plans with equal counts of a, b but no c are accepted. (3) admits plans where a occurs twice, and b, c not at all. □

6 Classical Planning with Presburger Constraints

We now introduce the next new planning formalism. We extend classical planning by allowing to constrain plans by existentially quantified Presburger formulas. Presburger formulas can be seen as an extension of ILPs, as they additionally allow disjunctions among constraints, but come with the drawback that constants are represented in unary. E.g., 3 is encoded as 1 + 1 + 1. The connection to ILPs forms the next link in the chain connecting all planning formalisms presented in this paper. Accordingly, we will encode this formalism into ILP constraints for the membership proof. To make use of the work of Verma, Seidl, and Schwentick (2005) in the next section, we reproduce their definition of existentially quantified Presburger formulas, which in turn originates from Seidl et al. (2004).

Definition 6.1 (Presburger Formula)**.** An *existential Presburger formula* ϕ over a set of variables $X = \{x_1, ..., x_n\}$ is a string derived from the context-free grammar with start symbol ϕ_N and production rules:

$$x \to x_1 \mid ... \mid x_n$$
$$t \to 0 \mid 1 \mid x \mid (t + t)$$
$$\phi_N \to (t = t) \mid (t < t) \mid (\phi_N \wedge \phi_N) \mid (\phi_N \vee \phi_N) \mid (\exists x : \phi_N)$$

Here, x, t, and ϕ_N are non-terminals, while all other symbols are terminals. The derived string, i.e. the Presburger formula ϕ, expresses a first-order logic formula with constants 0, 1 and variables x_1, ..., x_n that can be interpreted over the structure $(\mathbb{N}, \leq, +)$. Formally, variables are assigned values via a substitution function $\sigma: X \to \mathbb{N}$. If the resulting ground formula is satisfied, we write $\sigma \vDash \phi$. The solution set $\mathrm{sol}(\phi) := \{\sigma: X \to \mathbb{N} \mid \sigma \vDash \phi\}$ contains all substitutions satisfying ϕ.

Like with ILPs, these formulas simply restrict the amount of operators that occur in the solutions, via the Parikh vector.

Definition 6.2. A classical planning problem with Presburger constraints $\Pi_\phi = (\Pi, \phi)$ consists of classical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ and Presburger formula ϕ. The set of all plans for Π_ϕ is:

$$\mathrm{sol}(\Pi_L) := \{\pi \in \mathrm{sol}(\Pi) \mid \exists \sigma \in \mathrm{sol}(\phi) \\ \forall o \in \mathit{Ops} : \Psi(\pi)_o = \sigma(o)\}$$

The set of all planning problems with Presburger constraints is denoted by $\mathcal{PRESBURGER}$.

We now prove PSPACE-completeness by converting the Presburger formula non-deterministically into an ILP.

Theorem 6.3. *Deciding whether there exists a plan for a classical planning problem with Presburger constraints is PSPACE-complete.*

Proof. Hardness is shown by reduction from Classical planning, which is PSPACE-complete (Erol, Nau, and Subrahmanian 1995). We reduce a classical planning problem Π to a classical planning problem with Presburger constraints $\Pi_L = (\Pi, \phi)$, with trivially true formula $\phi = (0 < 1)$,

For membership, we follow Verma, Seidl, and Schwentick (2005), to encode a Presburger formula ϕ into an ILP. We use a non-deterministic transformation that removes all disjunctions $\phi_1 \vee \phi_2$ by non-deterministically choosing either ϕ_1 or ϕ_2. Let ϕ' be the resulting formula after all such choices. Then ϕ' contains only conjunctions, atomic comparisons, and existential quantifiers. We move all existential quantifiers to the front, yielding:

$$\phi' \equiv \exists x_1 : ... \exists x_n : \bigwedge_{i=1}^{m} \psi_i$$

where all ψ_i are quantifier-free and atomic formulas of shape $t_1 = t_2$ or $t_1 < t_2$. The structures can be encoded in an ILP (M, v). M has n columns so that each row can match constants $c_1, ..., c_n \in \mathbb{Q}$ to the values variables $x_1, ..., x_n$ would take. Each such row, together with a value c_0 of v, forms a constraints of shape $\sum_{i=1}^{n} c_i \cdot x_i \leq c_0$, which can be generated from the Presburger formulas: Given that Presburger formulas are interpreted over $(\mathbb{N}, \leq, +)$ we can use standard algebraic rewriting to bring $t_1 < t_2$ to shape $\sum_{i=1}^{n} c_i^* \cdot x_i < c_0^*$. Here $c_i^* \cdot x_i$ represents $c_i^* \in \mathbb{N}$ additions of x_i and c_0^* is a sum of constants $0, 1$. We can represent this as linear constraint $\sum_{i=1}^{n} c_i^* \cdot x_i \leq c_0^* - 1$. For $t_1 = t_2$ we use the same rewriting, but generate $\sum_{i=1}^{n} c_i \cdot x_i \leq c_0 \wedge \sum_{i=1}^{n} c_i \cdot x_i \geq c_0$ where $\geq$ can be represented by negating the according constants in M.

If ϕ is satisfiable by some plan, then for every disjunction in the formula, at least one part must be satisfied. Any other transformation we do preserves equivalence. Therefore, if there exists a plan satisfying ϕ, there is at least one such transformation that results in a satisfiable ILP instance. Moreover, if the ILP instance is satisfiable, this guarantees that for each disjunction in the original formula, at least one part is satisfiable. This concludes correctness. □

We now turn to expressivity, showing $\mathcal{PRESBURGER}$ is more expressive than classical planning.

Theorem 6.4. *$\mathcal{PRESBURGER}$ is significantly more expressive than classical planning.*

Proof. The hardness encoding from classical planning in the proof of Thm. 6.3 shows that $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{PRESBURGER})$ by allowing arbitrary sequences without constraints. To prove that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are included we use the sequential operator gadget Π^{Seq} from Def. 4.8 and extend it with the following Presburger formula ϕ over variables x_a, x_b, x_c to planning problem with Presburger constraints $\Pi_\phi = (\Pi^{\text{Seq}}, \phi)$:

(1) $(x_a = x_b) \wedge (x_b = x_c)$ for $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
(2) $(x_a = x_b) \wedge (x_c = 0)$ for $\{a^n b^n \mid n \in \mathbb{N}\}$
(3) $(x_a = (1{+}1)) \wedge (x_b = 0) \wedge (x_c = 0)$ for $\{aa\}$

(1) accepts exactly the Parikh images where a, b, c occur equally often. (2) accepts the Parikh images where a, b occur equally often, but no c. And (3) accepts only plans where a occurs twice and b, c not at all. □

7 Classical Planning with Grammar Amount Constraints

We now introduce the final new formalism before closing the gap to (unordered) HTN planning. It again constrains the number of operator occurrences in a solution, this time by requiring that a context-free grammar can derive a word with the same Parikh vector, i.e., the same number of terminal occurrences matching an operator. As before, we will encode this formalism into the previously introduced one, continuing the next link in the chain connecting all presented planning formalisms.

Definition 7.1. A *classical planning problem with grammar amount constraints* $\Pi_G = (\Pi, G)$ consists of classical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ and context-free grammar G with terminals Ops. The plans for Π_G are:

$$\mathrm{sol}(\Pi_G) := \{\pi \in \mathrm{sol}(\Pi) \mid \exists L \in L(G) : \Psi(\pi) = \Psi(L)\}$$

$\mathcal{GAMOUNT}$ is the set of all planning problems with grammar amount constraints.

Again, we show that plan existence under this formalism matches the complexity of classical planning.

Theorem 7.2. *Deciding whether there exists a plan for a classical planning problem with grammar amount restrictions is PSPACE-complete.*

Proof. Hardness is shown by reduction from Classical planning, which is known to be PSPACE-complete (Erol, Nau, and Subrahmanian 1995). This is, we can simply reduce a planning problem Π to classical planning problem with

grammar amounts, where the grammar amounts are arbitrary. This can be encoded in a context-free grammar with start symbol S and production rules:

$$S \to S\,o_1 \mid ... \mid S\,o_n \mid \varepsilon$$

Where the terminal $Ops = \{o_1, ..., o_n\}$ represent the operators in the problem and S is the only non-terminal.

For membership, we encode classical planning problem with grammar amount constraints $\Pi_G = (\Pi, G)$ into a classical planning problem with Presburger constraints $\Pi_\phi = (\Pi, \phi)$, by using the result from Verma, Seidl, and Schwentick (2005), stating that there is a linear time construction that encodes a predicate matching the Parikh vector of a CFG G into existential Presburger Logic ϕ. So, by Thm. 6.3 we can use the PSPACE verifier for Π_ϕ. □

It is easy to prove that $\mathcal{GAMOUNT}$ is signficantly more expressive than classical planning. We omit the proof due to space constraints. It is akin to the construction in Thm. 8.2.

8 Unordered HTN Planning

We now analyze unordered HTN planning and show that plan existence is PSPACE-complete. So far, we knew that deciding plan existence is EXPTIME for total-ordering (Erol, Hendler, and Nau 1996), and Ackermann-complete for some restricted partial orders (Dekker and Behnke 2024). PSPACE fragments had only been identified by restricting the hierarchy (Alford, Bercher, and Aha 2015), which is less desirable for modeling. Unordered HTN was studied only for lifted plan verification (Lauer, Lin, and Bercher 2025), but its plan existence complexity was unknown. We close this gap by encoding the hierarchy into a context-free grammar, completing the chain of encodings.

Theorem 8.1. *Deciding whether there exists a plan for an unordered HTN planning problem is PSPACE-complete.*

Proof. For hardness we use a reduction from Classical planning, which is PSPACE-complete. To simulate a classical planning problem with HTN planning we use the construction from Erol, Hendler, and Nau (1996, Sec. 3.3). This is to have one compound task that can be decomposed into an arbitrary operator or twice this compound task. On this construction one can impose arbitrary ordering constraints, without restricting the operator refinements, as any arbitrary sequence remains to be a refinement.

From the HTN structure we create a context-free grammar with start symbol S and the following production rules: There is one production rule for the initial task network $tn_I = (T, \alpha)$ with task IDs $T = \{tid_1, ..., tid_n\}$:

$$S \to \alpha(tid_1)\ ...\ \alpha(tid_n)$$

And one production rule per method $m \in \mathcal{M}$ to replace the head of $m = (c, tn)$ with $tn = (T, \alpha)$, $T = \{tid_1, ..., tid_n\}$:

$$c \to \alpha(tid_1)\ ...\ \alpha(tid_n)$$

The terminals are Ops and non-terminals $\mathcal{C}$.

First observe that by our definition, of terminals and non-terminals each derived string only contains operators. Now, we can inductively observe (starting with initial task network, to $n \in \mathbb{N}$ derivation steps) that each application of a production rule mirrors the replacement by a method in a way, so that the amount of non-terminals, representing compound tasks, and terminals, representing operators, are the same as by the method replacement. This concludes that the amounts and so the Parikh image of the HTN structure and constructed CFG are the same, allowing us to construct a planning problem with grammar amounts and exploit the PSPACE-verifier from Thm. 7.2. □

At first, it may seem that unordered HTN is restrictive, since orders between operators are very natural. However, these orders do not necessarily need to be enforced through task network constraints. We implicitly show this in the following by encoding languages such as $\{a^n b^n \mid n \in \mathbb{N}\}$, where a must precede b, into unordered HTN.

Theorem 8.2. *$\mathcal{HTN}_\mathcal{U}$ is significantly more expressive than classical planning.*

Proof. $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{HTN}_\mathcal{U})$ follows from the encoding to classical planning in the proof of Thm. 5.3 where arbitrary classical planning problems can be encoded in HTN planning with matching plans.

To show that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are in $L(\mathcal{HTN}_\mathcal{U})$, we construct unordered HTN planning tasks $\Pi_\mathcal{H} = (\Pi^{\text{Seq}}, \mathcal{H})$. The hierarchy $\mathcal{H} = (tn_I, \mathcal{M}, \mathcal{C})$ contains one compound task, i.e., $\mathcal{C} = \{S\}$ and the initial task network as $tn_I = (\{t_0\}, \{t_0 \mapsto S\})$. We define methods:

(1) For $\{a^n b^n c^n \mid n \in \mathbb{N}\}$:

$m_1 = (S, (\{\}, \{\}))$
$m_2 = (S, (\{t_1, t_2, t_3, t_4\}, \{t_1 \mapsto a, t_2 \mapsto b, t_3 \mapsto c, t_4 \mapsto S\}))$

(2) For $\{a^n b^n \mid n \in \mathbb{N}\}$:

$m_1 = (S, (\{\}, \{\}))$
$m_2 = (S, (\{t_1, t_2, t_3\}, \{t_1 \mapsto a, t_2 \mapsto b, t_3 \mapsto S\}))$

(3) For $\{aa\}$:

$$m = (S, (\{t_1, t_2\}, \{t_1 \mapsto a, t_2 \mapsto a\}))$$

In (1) every decomposition of S adds the same number of a, b, c; making it $\{a^n b^n c^n\}$. In (2) every decomposition of S adds the same number of a, b; making it $\{a^n b^n\}$. (3) yields the task network with two operators a, matching $\{aa\}$. □

Together, the theorems underline why unordered HTN can combine the hierarchy guidance with state-of-the-art planning techniques. E.g., consider the language represented by:

$$(drive\_to_1 \mid \ldots \mid drive\_to_m)^n\ load$$
$$(drive\_back_1 \mid \ldots \mid drive\_back_m)^n\ unload$$

Here, a truck drives, loads a package, and returns. Encoding this similar to $a^n b^n$ allows counting the remaining $drive\_back$ (and $unload$) operators in the task network after $load$, to get the exact remaining plan length. But, there is no additional guidance to reach the load step. To provide that, one can likely use techniques from classical planning (Hoffmann and Nebel 2001; Richter and Westphal 2010; Seipp

and Helmert 2018), as the plan existence is also PSPACE-complete. From a practical perspective, this is a major advantage, as there are many ongoing lines of research to improve planners for classical planning (Corrêa et al. 2020; 2021; 2022; Lauer et al. 2020; 2021; 2025a; 2025b; Chen et al. 2024a; 2024b; Tollund et al. 2025).

Similar benefits likely exist for the other formalisms. But HTNs are already well-studied, making it easier to reuse existing knowledge and to improve existing applications.

9 Conclusion

We have analyzed five planning formalisms that are significantly more expressive than classical planning, without increasing the complexity of plan existence. Our results open new directions for modeling, both through the introduction of novel formalisms and by highlighting the low complexity of unordered HTN planning. The latter findings strongly advocate a shift towards unordered HTN planning problems.

Acknowledgment

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – ProjectID 232722074 – SFB 1102. Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight Bounds for HTN Planning. In *Proc. of the 25th ICAPS*, 7–15.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Comp. Intell.*, 11: 625–656.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning domain-independent heuristics for grounded and lifted planning. In *Proc. of the 38th AAAI*, 20078–20086.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proc. of the 34th ICAPS*, 68–76.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In *Proc. of the 31st ICAPS*, 94–102.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *Proc. of the 30th ICAPS*, 80–89.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In *Proc. of the 36th AAAI*, 9716–9723.

Dekker, M.; and Behnke, G. 2024. Barely Decidable Fragments of Planning. In *Proc. of the 27th ECAI*, 4198–4206.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Ann. Math. Artif. Intell.*, 18(1): 69–93.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *AIJ*, 76(1-2): 75–88.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proc. of the 2nd IJCAI*, 608–620.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proc. of the 22nd IJCAI*, 1955–1961.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proc. of the 6th AIPS*, 44–53.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AIJ*, 173: 503–535.

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

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proc. of the 21st ECAI*, 447–452.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages. In *Proc. of the 26th ICAPS*, 158–165.

Lauer, P. 2025. Arguments in Favor of Allowing a Modeler to Constrain Action Repetitions. In *Proc. of KEPS at ICAPS*.

Lauer, P.; and Fickert, M. 2020. Beating LM-cut with LM-cut: Quick Cutting and Practical Tie Breaking for the Precondition Choice Function. In *Proc. of the 12th HSDIP at ICAPS*.

Lauer, P.; and Fišer, D. 2025. Potential Heuristics: Weakening Consistency Constraints. In *Proc. of the 35th ICAPS*.

Lauer, P.; Lin, S.; and Bercher, P. 2025. Tight Bounds for Lifted HTN Plan Verification and Bounded Plan Existence. In *Proc. of the 35th ICAPS*.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proc. of the 30th IJCAI*, 4119–4126.

Lauer, P.; Torralba, Á.; Höller, D.; and Hoffmann, J. 2025. Continuing the Quest for Polynomial Time Heuristics in PDDL Input Size: Tractable Cases for Lifted hAdd. In *Proc. of the 35th ICAPS*.

Lin, S.; and Bercher, P. 2022. On the Expressive Power of Planning Formalisms in Conjunction with LTL. In *Proc. of the 32nd ICAPS*, 231–240.

Papadimitriou, C. H. 1981. On the complexity of integer programming. *JACM*, 28: 765–768.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39: 127–177.

Seidl, H.; Schwentick, T.; Muscholl, A.; and Habermehl, P. 2004. Counting in Trees for Free. In *Proc. of the 31st ICALP*, 1136–1149.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fiser, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine*, 45: 280–296.

Tollund, R. G.; Larsen, K. G.; and Torralba, A. 2025. What Makes You Special? Contrastive Heuristics Based on Qualified Dominance. In *Proc. of the 34th IJCAI*, 8652–8660.

Verma, K. N.; Seidl, H.; and Schwentick, T. 2005. On the Complexity of Equational Horn Clauses. In *Proc. of the 20th CADE*, 337–352.