Probabilistic HTN Planning: Formalization and Computational Complexity Analysis

Mohammad Yousefi , Johannes Schmalz , Patrik Haslum , Pascal Bercher
School of Computing, The Australian National University
{mohammad.yousefi, johannes.schmalz, patrik.haslum, pascal.bercher}@anu.edu.au

Abstract

Hierarchical Task Network (HTN) planning is an approach to sequential decision making that allows expressing complex grammar-like path constraints. In this paper, we first introduce an extension to HTN planning that takes probabilistic outcomes into account, and then study the computational complexity of deciding such problems either by finding a fixed sequence of actions (i.e., a conformant solution) or an outcome-dependent policy. This formalization extends factored Markov Decision Processes (MDPs) to have a hierarchical structure. In all studied cases, the conformant solutions are harder to obtain than their non-deterministic analogues, whereas policies are not always harder. Surprisingly, unlike their deterministic counterparts, severely restricted cases of probabilistic HTN problems are proven to be undecidable. The result holds even if all of the transition probabilities are bounded to be 0, 0.5, or 1.

1 Introduction

Markov Decision Processes (MDPs) are one of the fundamental frameworks for decision making under uncertainty, and there has been extensive research on the computational complexity of solving them under various conditions (Littman, Goldsmith, and Mundhenk 1998; Littman 1997; Papadimitriou and Tsitsiklis 1987). When faced with hierarchical problems, the traditional approach (especially in the reinforcement learning literature) has been to start with MDPs as the probabilistic foundation and shape a hierarchy through various extensions like HAMs (Parr and Russell 1997), Options (Sutton, Precup, and Singh 1999), and MAXQ (Dietterich 2000). In this paper, we take the opposite approach – we begin with Hierarchical Task Networks (HTNs), a framework designed for expressing complex hierarchical structures, and extend it to handle probabilistic dynamics. HTN planning is a powerful approach to sequential decision making that enables expressing multiple layers of abstraction by decomposing tasks into refined, smaller subtasks (Bercher, Alford, and Höller 2019; Erol, Hendler, and Nau 1996). While HTN planning has been studied in the non-deterministic setting (Chen and Bercher 2022; 2021; Patra et al. 2021; 2019; Ghallab, Nau, and Traverso 2016; Kuter and Nau 2004), less attention has been paid to its potential in scenarios involving probabilistic outcomes. In this paper, we start with formalizing the notion of a probabilistic hierarchical planning problem (and the solution criteria under different observability conditions), and then investigate the computational complexity of finding a solution under various observability and hierarchical restrictions. In order to do this, we build upon one of the formalisms for HTN planning with non-deterministic action effects (Chen and Bercher 2021), and attach a probability distribution to the outcomes. This results in a new formalism where policies are structured hierarchically, extending beyond the traditional flat mapping from states to actions typically seen in MDPs and classical planning, and can natively express complex constraints on the action sequences. Our contributions can be summarized as follows.

- providing a unified framework for reasoning about task hierarchies and probabilistic action outcomes that:
  - guarantees every solution terminates in a finite number of steps since unbounded loops are prohibited,
  - provides means of expressing and enforcing procedural knowledge (i.e., path constraints on sequences of actions),
- analyzing the computational complexity of solution existence in our framework under various constraints.

2 Formalization

In this section, we formally define our extension of non-deterministic HTN planning formalism (Chen and Bercher 2021) by assigning probabilities to action outcomes. Before presenting the formalism, we begin with a simple example in our framework.

2.1 Motivating Example

To motivate the discussion, consider the following delivery problem where we want to deliver a package P to someone at location 9 in a grid-world setting. It is known that the recipient may or may not be at home with an equal probability. In case they are not at home, the package must be delivered to the nearest post office. The problem is depicted in Fig. 1 a. We know that any solution can be roughly divided into three abstract steps: picking up the package, attempting to deliver it, and compensate if the recipient is not at home (Fig. 1 b). Such *compound* tasks need to be refined using *decomposition methods*. The methods exemplify a divide-and-conquer paradigm that systematically decomposes abstract tasks into manageable subtasks. They are the quintessential characteristic of HTN planning. While classical planning formalisms are limited to specifying outcome objectives (the *what*), HTN planning methods extend this capability by explicitly encoding procedural knowledge (the *how*). A few decomposition steps for our example are depicted in Fig. 1 c. As a first step, $dispatch[p, l_3, l_9]$ is broken down into two subtasks (by decomposition), to obtain:

$$\mathit{pickup}[p, l_3] \prec \mathit{drive}[l_3, l_9] \prec \mathit{deliver}[p, l_9] \prec \mathit{confirm}[p, l_9]$$

where $\mathit{drive}[l_3, l_9]$ is then recursively broken down. Now suppose that the first two initial abstract tasks are refined and the *primitive actions* are *executed* successfully. This means that there is one $\mathit{confirm}[p, l_9]$ task remaining, and we can be in two possible states with equal probability, depending on whether the delivery attempt was successful or not. One possible refinement of this last task is to delete it, resulting in a linear solution with a success rate of 50%. The feasibility of a linear solution with 100% success rate depends on whether the actions have conditional effects or not. If delivering a package to a person removes it from the truck, then logically it cannot also be delivered to the post office. However, this conflict can be resolved if unloading the package at the post office is implemented as a conditional effect (triggered by the package being in the truck) rather than as a precondition. A more flexible solution is to have a dynamic *policy* that observes the current state and changes the order of task executions (while adhering to the hierarchy). A notable difference with classical policies is that now we can enforce behaviors such as: "return to location l_2 following precisely the same path that was originally used to reach that destination". In this example, as we will prove later, due to the total ordering of the initial task network and methods, any policy collapses to a linear solution.

2.2 Problem Formalization

Tasks and their execution dependencies are captured in a partially ordered multi-set of tasks, which we refer to as a task network.

Definition 1 (Task Network). *A task network is defined as a tuple $tn = \langle T, \prec, \alpha \rangle$ where:*

- *T is a finite set of task IDs,*
- *$\prec \subseteq T \times T$ is a partial ordering of T,*
- *$\alpha \colon T \to N$ is a mapping from task IDs to a set of task names, N (see Def. 2).*

A hierarchical planning domain encodes the world-states with propositional facts, describes how primitive tasks affect the world, and how compound tasks can be decomposed:

Definition 2 (Planning Domain). *An HTN planning domain is a tuple $D = \langle F, N_p, N_c, \delta, A, M \rangle$ where:*

- *F is a finite set of propositional facts,*
- *N_p and N_c are disjoint finite sets of <u>p</u>rimitive and <u>c</u>ompound task names, and their union gives N, the set of all task names,*
- *$\delta \colon N_p \to A$ is a mapping from primitive task names to their corresponding action,*
- *A is a finite set of probabilistic actions (see Def. 3),*
- *$M \subseteq N_c \times TN$ the finite set of decomposition methods, where TN is the infinite set of all possible task networks.*

So far, the definitions of a task network and planning domain follow the standard HTN formalism (Bercher, Alford, and Höller 2019). In the next step, we extend non-deterministic action effects (Chen and Bercher 2021; Cimatti et al. 2003) with a probability distribution and conditional effects. The conditional effect is crucial in settings where plans are linear since many problems do not have a solution unless some runtime "decision-making" is possible (Brafman and Hoffmann 2004). For example, assume in a 3×3 grid, the first action non-deterministically places a robot in one of the cells and the task is to reach the upper right cell. A linear solution that only works when conditional effects are present is to move 2 cells to right followed by moving up 2 cells. In case the robot is next to a wall, the conditional effect prevents it from moving in that direction. Throughout the paper we use $\wp(X)$ to denote the power set of X, and $S = \wp(F)$ to denote the set of all states.

Definition 3 (Probabilistic Action). *A (primitive) probabilistic action is a tuple $a = \langle \mathit{pre}(a), \mathit{eff}(a), \mathcal{P}_a \rangle$ where:*

- *$\mathit{pre}(a) \subseteq F$ is a set of propositions that must be true in order for a to be applicable,*
- *$\mathit{eff}(a) \subseteq \wp(\wp(F) \times \wp(F) \times \wp(F))$ is a set of effects. Each effect, $E \in \mathit{eff}(a)$, is a set of triples $\langle \mathit{con}, \mathit{add}, \mathit{del} \rangle$ where con denotes the conditions under which the propositions are added or deleted, respectively. An effect is called unconditional iff $\mathit{con} = \emptyset$,*
- *$\mathcal{P}_a \colon \mathit{eff}(a) \to [0, 1]$ is a probability mass function over the action effects (i.e., $\forall E \in \mathit{eff}(a) \colon 0 \le \mathcal{P}_a(E) \le 1$ and $\sum_E \mathcal{P}_a(E) = 1$).*

A primitive action is *applicable* in a state $s \in S$, iff its preconditions are true in s. We define the applicability function $\tau : A \times S \to \{\top, \bot\}$ such that $\tau(a, s) = \top \iff \mathit{pre}(a) \subseteq s$. If the primitive action a is applicable in s, then executing it results in a set of successor states defined as $\gamma(s, a) = \{\gamma_E(s, a) \mid E \in \mathit{eff}(a)\}$ where $\gamma_E(s, a) = \{(s \setminus \mathit{del}) \cup \mathit{add} \mid \mathit{con} \subseteq s, \langle \mathit{con}, \mathit{add}, \mathit{del} \rangle \in E\}$. We define the probability of reaching a particular state after executing an applicable action as follows:

$$\mathcal{P}(s' \mid s, a) := \sum_{E \in \mathit{eff}(a)} \mathcal{P}_a(E) \cdot I(s' \mid s, E)$$

where the function indicating whether s' is the result of effect E on state s is given by:

$$I(s' \mid s, E) := \begin{cases} 1 & \text{if } s' = \gamma_E(s, a) \\ 0 & \text{otherwise.} \end{cases}$$

There can be $E \neq E'$ with $I(s' \mid s, E) = 1$ and $I(s' \mid s, E') = 1$, i.e., there can be multiple effects that lead to the same state. As an example, suppose $s = \emptyset$ and we have two unconditional effects E and E' with a uniform distribution. Suppose E only adds a proposition f, whereas E' adds f and removes f'. Clearly, the probability of $s' = \{f\}$ is 100%. So, when evaluating the probability of reaching s', we must add up the probabilities of all effects that lead to s'.

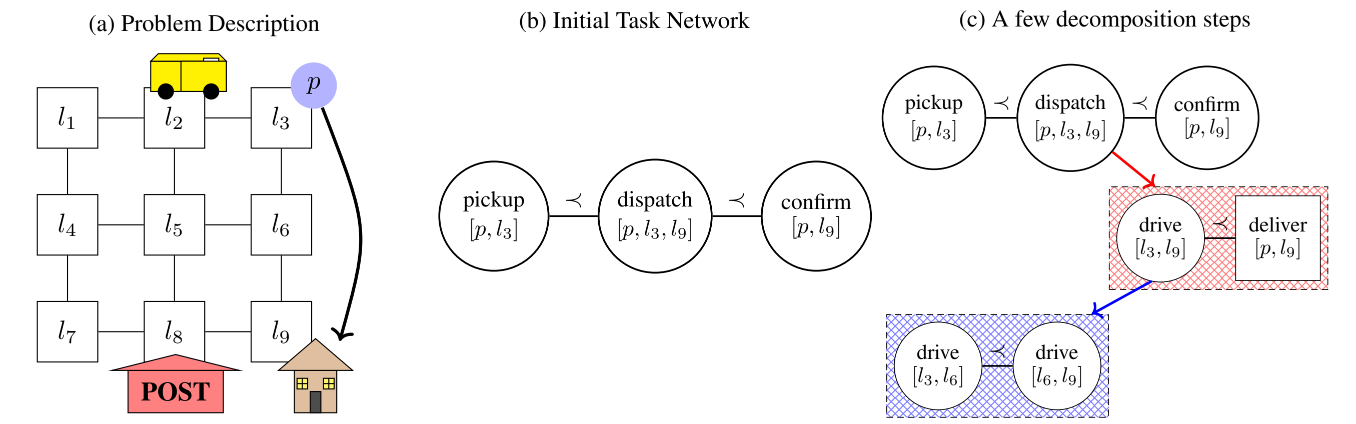

Figure 1: (a) Deliver package P to someone at location 9 who may or may not be at home with an equal probability. In case they are not at home, the package must be delivered to the nearest post office. (b) The initial task network consisting of 3 compound tasks. (c) A few steps of decomposing the initial task network into a more refined one. By convention, primitive tasks are square and compound ones are circular.

A planning problem is given by a domain, the initial state of the world, and a task network. To solve the problem, one must achieve all the tasks in the task network, subject to its partial ordering and the dynamics of the domain initialised at the start state. Note that we do not have goal descriptions because the "goal" is to achieve all tasks. Explicit goal descriptions would not add any power, since they can be compiled into the precondition of an artificial primitive task that is appended as a last task to the initial task network (Geier and Bercher 2011).

Definition 4 (Probabilistic HTN Planning Problem)**.** *A Probabilistic HTN planning problem is a tuple* $\Omega = \langle D, tn_I, s_I \rangle$ *where:*

- *D is an HTN domain,*
- *$s_I \in S$ is an initial state,*
- *$tn_I \in TN$ is an initial task network.*

In order to solve an HTN planning problem, we need to first define the equivalence test between two task networks.

Definition 5 (Isomorphism of Task Networks)**.** *The task networks* $tn_1 = \langle T_1, \prec_1, \alpha_1 \rangle$ *and* $tn_2 = \langle T_2, \prec_2, \alpha_2 \rangle$ *are isomorphic, written* $tn_1 \cong tn_2$*, iff there exists a bijection* $\sigma : T_1 \rightarrow T_2$ *such that* $(t \prec_1 t') \iff (\sigma(t) \prec_2 \sigma(t'))$ *and* $\alpha_1(t) = \alpha_2(\sigma(t'))$ *for all* $t, t' \in T_1$*.*

The main machinery of HTN planning is the decomposition methods which refine compound tasks by substituting them with their corresponding network of subtasks.

Definition 6 (Task Decomposition)**.** *A method* $\langle c, tn \rangle \in M$ *decomposes a task network* $tn_1 = \langle T_1, \prec_1, \alpha_1 \rangle$ *into* $tn_2 = \langle T_2, \prec_2, \alpha_2 \rangle$ *if and only if there exists a* $t \in T_1$ *such that* $\alpha_1(t) = c$ *and there is a task network* $tn' = \langle T', \prec', \alpha' \rangle$ *with* $tn' \cong tn$ *where* $T_1 \cap T' = \emptyset$*. The task network* tn_2 *is defined as* $tn_2 = \langle (T_1 \setminus \{t\}) \cup T', \prec' \cup \prec_D, (\alpha_1 \setminus \{t \mapsto c\} \cup \alpha') \rangle$ *where* $\prec_D$ *is defined as follows.*

$$\begin{aligned}\prec_D = &\{(t_1, t_2) \mid (t_1, t) \in \prec_1, t_2 \in T'\} \cup \\ &\{(t_1, t_2) \mid (t, t_2) \in \prec_1, t_1 \in T'\} \cup \\ &\{(t_1, t_2) \mid (t_1, t_2) \in \prec_1, t_1 \neq t \wedge t_2 \neq t\}\end{aligned}$$

Decompositions can be chained together to *achieve* another task network. This process can go on until we find a *primitive* one. Primitive task networks cannot be decomposed any further.

Definition 7 (Primitive and Achievable Task Networks)**.** *A task network* tn *is achievable from* tn' *iff there is a finite (potentially empty) sequence of decompositions that transform* tn' *into* tn'' *such that* $tn'' \cong tn$*. A task network* $tn'' = \langle T, \prec, \alpha \rangle$ *is primitive iff* $\forall t \in T\colon \alpha(t) \in N_p$*.*

Given the operations to decompose task networks and execute primitive actions, we are now equipped to formally define the solution criteria to such problems. Notably, a simple sequence of actions is no longer the only form of solution as we might need to alter the course of actions based on the observed outcome of an action.

2.3 Conformant Solution

Conformant solutions, also called linearization-dependent solutions (Chen and Bercher 2021), are a fixed sequence of primitive actions (i.e., a linearization) that is guaranteed to be executable regardless of the uncertainty. Thus, the setup is similar to conformant probabilistic planning, also called probabilistic planning with no observability and conditional probabilistic planning (Hyafil and Bacchus 2004; Majercik and Littman 2003; Kushmerick, Hanks, and Weld 1995), but in the HTN setting. In order to formally define the probabilistic conformant solution criteria, we first need to consider the state trajectories that can arise from executing a sequence of probabilistic actions.

Definition 8 (Conformant Trajectory)**.** *Let* $tn = \langle T, \prec, \alpha \rangle$ *be a primitive task network, and* $\sigma = t_1, t_2, ..., t_k$ *be some linearization of* tn *(i.e., a total ordering of* T *that is consistent with* $\prec$*). We write* $\mathcal{X}(\sigma, s)$ *to denote the set of state sequences that are induced from executing* σ*. Formally, for all* $x = \langle s_1, \ldots, s_{k+1} \rangle \in \mathcal{X}(\sigma, s)$*, it holds that* $s_1 = s$ *and* $\forall i \in \{1, \ldots, k\}$*, we have:*

- $\tau(\alpha(t_i), s_i) = \top$

- $\mathcal{P}(s_{i+1} \mid s_i, \alpha(t_i)) > 0$

The probability of state sequence x is defined as

$$\mathcal{P}(x) \coloneqq \prod_{i=1}^{k} \mathcal{P}(s_{i+1} \mid s_i, \alpha(t_i))$$

A probabilistic conformant solution is a sequence of primitive tasks, derived from the hierarchy, where the probability of successful execution exceeds a threshold value ρ.

Definition 9 (ρ-Linearizable)**.** *Consider the Probabilistic HTN problem $\Omega = \langle D, tn_I, s_I\rangle$. For any probability threshold $\rho \in (0, 1]$, we say that Ω is ρ-linearizable iff there exists:*

- *a primitive task network tn achievable from tn_I, and*
- *a linearization, σ, of tn such that $\sum_{x \in \mathcal{X}(\sigma, s_I)} \mathcal{P}(x) \geq \rho$*

2.4 Outcome-dependent Solution

In contrast to the conformant solution, outcome-dependent solutions are able to change course based on the observed effects of actions, making them more powerful (Chen and Bercher 2021). In particular, after executing each action, the realized non-deterministic effect of that action is observed. This allows the subsequent action to be *selected* based on the state. As such, the solution is not a linear sequence of actions, but a Directed Acyclic Graph (DAG) that branches based on the action outcomes. This DAG is represented as a partial function, that is referred to as a *policy*.

Definition 10 (Policy)**.** *Let $tn = \langle T, \prec, \alpha\rangle$ be a primitive task network. A policy is a partial function $\pi : \wp(T) \times S \to T$ such that for all $T' \subseteq T$ and $s \in S$ where π is defined (i.e., $\pi(T', s) = t$), we have:*

- *$t \notin T'$,*
- *$\forall \tilde{t} \in T \setminus T' \colon\ \tilde{t} \not\prec t$, and*
- *$\tau(\alpha(t), s) = \top$.*

A key distinction between this hierarchical policy formulation and classical state-action policies in MDPs is the guarantee of termination. Unlike classical policies that may permit infinite loops (i.e., "do X until Y"), our approach – rooted in primitive task networks with a bounded number of tasks – ensures that every policy solution terminates in a finite number of steps.

The underlying DAG is known as the *execution structure* of the policy (Chen and Bercher 2021). The success probability of a policy is computed based on this representation.

Definition 11 (Execution Structure)**.** *Let π be a policy for primitive task network $tn = \langle T, \prec, \alpha\rangle$. We define the execution structure of π as a directed graph $K = \langle Q, R\rangle$ where $Q \subseteq \wp(T) \times S$ and $R \subseteq \langle \wp(T) \times S\rangle \times T \times \langle \wp(T) \times S\rangle$ are minimal sets satisfying:*

- *$\langle \emptyset, s_I\rangle \in Q$*
- *if $\langle T', s\rangle \in Q$ and $\pi(T', s) = t$ then, for all $s' \in \gamma(s, \alpha(t))$:*
  - *$\langle T' \cup \{t\}, s'\rangle \in Q$*
  - *$\Big\langle \langle T', s\rangle, t, \langle T' \cup \{t\}, s'\rangle \Big\rangle \in R$*

Contrary to Def. 8, a trajectory that is induced by a policy interleaves observation and execution, allowing a more flexible approach.

Definition 12 (Observable Trajectory)**.** *Let π be a policy for the primitive network $tn = \langle T, \prec, \alpha\rangle$, and $K = \langle Q, R\rangle$ be the execution structure of π. We define the sequence $\phi = \langle T_1, s_1\rangle, \langle T_2, s_2\rangle, ..., \langle T_k, s_k\rangle$ as an observable trajectory of π iff it is a path in the execution structure from the initial node to a terminal node, i.e.,*

- *$\forall i \in \{1..k\} \colon \langle T_i, s_i\rangle \in Q$,*
- *$\forall \langle T_i, s_i\rangle, \langle T_{i+1}, s_{i+1}\rangle$ where $1 \leq i < k$, we have $\pi(T_i, s_i) = t$ such that $\big\langle \langle T_i, s_i\rangle, t, \langle T_{i+1}, s_{i+1}\rangle \big\rangle \in R$,*
- *$\langle T_1, s_1\rangle = \langle \emptyset, s_I\rangle$,*
- *$\langle T_k, s_k\rangle$ does not have any outgoing edges in K.*

A trajectory is defined to be successful iff $T_k = T$. The probability of occurrence is defined as:

$$\mathcal{P}(\phi \mid \pi) = \prod_{i=1}^{k-1} \mathcal{P}(s_{i+1} \mid s_i, \alpha(\pi(T_i, s_i)))$$

Recall that the product over the empty set is one, so $\mathcal{P}(\phi \mid \pi) = 1$ if the trajectory consists of zero or one step. This makes sense because applying zero or one actions will always succeed, and is unaffected by any probabilistic effects. Our definition of ρ-policies extends fixed-method policies (i.e., policies that cannot do decomposition during plan execution) in non-deterministic HTN planning (Chen and Bercher 2021), which contains the hierarchical information, with specifications on success probability.

Definition 13 (ρ-Policy)**.** *Let $\Omega = \langle D, tn_I, s_I\rangle$ be a probabilistic HTN planning problem, and $tn = \langle T, \prec, \alpha\rangle$ be a primitive task network. Let π be a policy for tn, and Φ be the finite set of all successful trajectories induced by this policy. For any probability threshold $\rho \in (0, 1]$, we say that π is a ρ-Policy for Ω iff:*

- *tn is achievable from tn_I, and*
- *$\sum_{\phi \in \Phi} \mathcal{P}(\phi \mid \pi) \geq \rho$.*

3 Background

We assume the reader is familiar with complexity classes and decidability as presented in textbooks (Sipser 2020; Hopcroft, Motwani, and Ullman 2006). However, since some concepts like Probabilistic Turing Machines and their associated complexity classes might be unfamiliar to some readers, we provide a brief explanation of the terminology used in the paper. In the latter part of this section, we present several well-known hierarchical restrictions from the HTN planning literature, which serve as the foundation for the subsequent analysis of their impact on computational complexity within our formalism.

3.1 Complexity Classes

We use Turing Machines as our model of computation.

Definition 14 (Turing Machine)**.** *A (deterministic) Turing Machine is a tuple $TM = \langle Q, \Gamma, \Sigma, \delta, q_0, B, F\rangle$ where:*

- *Q is a finite set of controller states,*
- *Γ is a finite set of tape symbols,*
- *$\Sigma \subseteq \Gamma$ is a finite set of input symbols,*
- *$\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a partial function that maps the current state and the symbol under the machine head to a triple of outputs: the next state, the symbol to be written, and the direction for the head movement,*
- *$q_0 \in Q$ is the initial state of the controller,*
- *$B \in \Gamma \setminus \Sigma$ is the blank symbol,*
- *$F \subseteq Q$ is a finite set of accepting states.*

In addition to the mentioned deterministic Turing Machine, we also use its non-deterministic variant where the output of δ is altered to be a set. Formally, we have $\delta\colon Q\times\Gamma \to \wp(Q\times\Gamma\times\{L,R\})$. Such machines, effectively "guess" the correct transition. To capture a complete snapshot of any Turing Machine variant, we present the concept of a *configuration*.

Definition 15 (Configuration)**.** *A (Turing Machine) configuration is a tuple $C = \langle \omega, q_c, n\rangle \in \Sigma^* \times Q \times \mathbb{N}$ where:*

- *ω is the content of the tape,*
- *q_c is the current state of the Turing Machine,*
- *n is the head position on the tape.*

We write $C_i \vdash C_{i+1}$ to indicate that configuration C_{i+1} is immediately derivable from C_i via a single transition in δ. For non-deterministic Turing Machines, we have $C_i \vdash C_{i+1}$ iff any of the transitions in the set transform C_i to C_{i+1}.

Definition 16 (Language of a Turing Machine)**.** *Let $TM = \langle Q, \Gamma, \Sigma, \delta, q_0, B, F\rangle$ be a Turing Machine. The language recognized by TM, denoted $\mathcal{L}(TM)$, is defined as the set of strings, $\omega \in \Sigma^*$, such that there exists a sequence of configurations $C_0, C_1, \ldots, C_n$ where:*

- *C_0 is the initial configuration with input ω, the machine in state q_0, and the head is on the first position of the tape,*
- *$C_i \vdash C_{i+1}$ for all $0 \leq i < n$, and*
- *C_n is a configuration where the machine is in some accepting state $q_f \in F$.*

Definition 17 (Decidability)**.** *A language L is decidable iff there exists a Turing Machine with $\mathcal{L}(TM) = L$ that halts for every ω with either $\omega \in \mathcal{L}$ or $\omega \notin \mathcal{L}$.*

For Probabilistic Turing Machines, we adopt the definition proposed by Gill (1974). These Turing Machines are a special kind of non-deterministic ones where the transition function δ is constrained to have either one or two outputs. On inputs where two outputs are available, the machine flips an unbiased coin to select one of them as the transition to be applied. Unlike non-deterministic machines, the language recognized by a Probabilistic Turing Machine is defined as the set of input strings for which *more than half* of all computation paths–represented by sequences of configurations–lead to an accepting state.

Complexity classes are defined by imposing a time or space constraint on the Turing Machines. Each class represents a set of problems that can be solved within the computational resource boundaries. We begin by introducing:

- $\mathbf{DTIME}(t(n))$ is the set of languages that are decidable by a deterministic TM in $O(t(n))$ steps.
- $\mathbf{NTIME}(t(n))$ is the set of languages that are decidable by a non-deterministic TM in $O(t(n))$ steps.
- $\mathbf{\Pi TIME}(t(n))$ is the set of languages that are decidable by a probabilistic TM in $O(t(n))$ steps.
- $\mathbf{DSPACE}(t(n))$ is the set of languages that are decidable by a deterministic TM using $O(t(n))$ tape cells.
- $\mathbf{NSPACE}(t(n))$ is the set of languages that are decidable by a non-deterministic TM using $O(t(n))$ tape cells.

Then, for a string $\omega \in \Sigma^*$ as input with $|\omega| = n$, we have:

- $\mathbb{PTIME} = \bigcup_{k\in\mathbb{N}} \mathbf{DTIME}(n^k)$
- $\mathbb{NP} = \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(n^k)$
- $\mathbb{PP} = \bigcup_{k\in\mathbb{N}} \mathbf{\Pi TIME}(n^k)$
- $\mathbb{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{DSPACE}(n^k)$
- $\mathbb{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{NSPACE}(n^k)$

Similarly, we use $\mathbb{EXPTIME}$, $\mathbb{NEXPTIME}$, $\mathbb{PEXP}$, and $\mathbb{EXPSPACE}$, when the bound is exponential (i.e., $O(2^{n^k})$). Additionally, we denote a language L as $\mathbb{UNDECIDABLE}$ if it is *not* decidable. Finally, we introduce the oracle machine–a theoretical extension of the Turing Machine that has access to a "black box" capable of instantly solving specific problems.

Definition 18 (Oracle Machine)**.** *An oracle machine X^Y is a Turing Machine X augmented with an additional tape that can query an oracle Y. The machine may write any string l on the oracle tape and determine whether $l \in \mathcal{L}(Y)$ in unit time.*

3.2 Hierarchical Restrictions

The undecidability of deterministic HTN planning (Erol, Hendler, and Nau 1996) has motivated research into decidable subclasses. In this section we list some of the most commonly used subclasses (Alford, Bercher, and Aha 2015; Erol, Hendler, and Nau 1996).

Definition 19 (Primitive Problem)**.** *An HTN planning problem, $\Omega = \langle D, tn_I, s_I\rangle$, is primitive if tn_I is a primitive task network.*

Another well-studied class emerged by restricting the interplay between task executions by imposing a total-ordering.

Definition 20 (Totally-Ordered Problem)**.** *An HTN planning problem is totally-ordered if the initial task network and the task network of every method is totally-ordered.*

To define the other subclasses, we use the notation of set stratification (Alford et al. 2012).

Definition 21 (Set Stratification)**.** *A set $R \subseteq N_c \times N_c$ is a stratification if it is a total preorder (i.e., reflexive, transitive, and total). A stratum is an inclusion-maximal subset $\mathcal{S} \subseteq N_c$ such that for all $x, y \in \mathcal{S}$, we have both $(x, y) \in R$ and $(y, x) \in R$.*

The acyclic class does not allow recursive tasks (i.e., a compound task that can reach itself via decomposition).

Definition 22 (Acyclic Problem)**.** *An HTN planning problem is acyclic if there exists a stratification R with the property that for all methods, $\langle c, \langle T, \prec, \alpha\rangle\rangle \in M$, and for all compound task, $t \in T$, we have both $(\alpha(t), c) \in R$ and $(c, \alpha(t)) \notin R$*

Tail-recursive problems allow special forms of recursion. In particular, it only allows a last task to be recursive. This restricts the growth of task networks by forcing every task before the recursive one to be on an easier stratum.

Definition 23 (Tail-recursive Problem)**.** *An HTN planning problem is tail-recursive if there exists a stratification R with the property that for all methods $\langle c, \langle T, \prec, \alpha\rangle\rangle \in M$, we have:*

1. *if there is a compound last task $t \in T$ (formally, $\alpha(t) \in N_c$ and for all $t' \neq t$ it holds that $(t', t) \in \prec$), then $(\alpha(t), c) \in R$,*
2. *for all compound non-last task $t \in T$, we have both $(\alpha(t), c) \in R$ and $(c, \alpha(t)) \notin R$*

4 Complexity Results

Before discussing the computational complexity, we show that if a probabilistic HTN problem is totally-ordered, it cannot form a branching policy. Thus, every ρ-policy in this setting can be reduced to a ρ-linearization. In this paper, we use this result to collapse the complexity of decision problems associated with these problems into one since knowing one answer immediately leads to the other one.

Theorem 1. *Let Ω be a totally-ordered HTN planning problem, and $\rho \in (0, 1]$. Ω has a ρ-policy if and only if it is ρ-linearizable.*

Proof. $\Rightarrow$) Assume Ω is ρ-linearizable, and let $\sigma = t_1, t_2, ..., t_k$ be one such solution. To construct a ρ-policy, we have to artificially augment σ with state observations. For this, we recursively construct a policy as follows:

- $\pi(\emptyset, s_I) = t_1$,
- $\pi(T, s) = t_i \implies \pi(T \cup \{t_i\}, s') = t_{i+1}$ for all $s' \in \gamma(s, \alpha(t_i))$ and $1 \leq i < k$.

$\Leftarrow$) Assume Ω has a ρ-policy. By definition, this policy is defined for a primitive task network, tn. Since the problem is totally-ordered, tn is also totally-ordered. It follows that the policy must execute each task one after another regardless of the state. Thus, the sequence of actions in tn is a linearization of the task network, which succeeds with probability exactly equal to that of the policy. □

Next, we provide the complexity results for probabilistic HTN planning across three key axes: the hierarchical restrictions (primitive, acyclic, tail-recursive, and arbitrary), ordering constraints (total or partial), and solution type (linear or branching). The results are summarized in Table 1.

4.1 Complexity Results for ρ-Linearization Existence

We start by introducing a variant of the model counting problem that is known to be $\mathbb{PP}$-Complete (Cozman and Mauá 2018, Prop. 1).

Definition 24 ($\#3SAT(>)$)**.** *Let ζ be a boolean formula in conjunctive normal form with exactly three literals per clause over variables $V = \{v_1, \ldots, v_n\}$, and k be a non-negative integer. The $\#3SAT(>)$ problem asks whether ζ has more than k satisfying truth assignments.*

Given a sequence of primitive tasks, deciding whether its probability of success exceeds a threshold is $\mathbb{PP}$-Complete.

Theorem 2. *Let $\Omega = \langle D, tn_I, s_I\rangle$ be a totally-ordered primitive HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization exists for Ω is $\mathbb{PP}$-Complete.*

Proof. (Membership) Our membership proof is similar to probabilistic plan evaluation in the classical setting (Littman, Goldsmith, and Mundhenk 1998). Given $tn_I = t_1, t_2, \ldots, t_n$, we construct the computation tree in which every root-to-leaf path represents a trajectory and its associated probability of success. For each action $a_k = \alpha(t_k)$, where $1 \leq k \leq n$, we require m coin flips, with m representing the number of bits needed to encode a common denominator of the outcome probabilities. In the worst case, we need to multiply every denominator together, leading to an exponential growth for the value of this denominator. However, given the binary encoding of the numbers, logarithmic number of bits are required for the encoding. Thus, the encoding is still polynomial. To represent the probabilities, we flip m coins to create a binary subtree with 2^m leaves, where the numerator determines the minimum number of accepting and rejecting leaves. In particular, we accept if at least $\rho = \frac{u}{v}$ of leaves are in an accepting state, and reject otherwise. When restricted to polynomial time bounds, probabilistic languages (i.e., an input is accepted if more than half of the computation leaves are accepting) and threshold languages (i.e., an input is accepted if at least a defined ratio of computation leaves are accepting) are identical (Simon 1975, Thm. 4.4). Thus, the problem is in $\mathbb{PP}$.

(Hardness) We reduce from the $\#3SAT(>)$ problem. Given a boolean formula ζ as a set of clauses $C = \{c_1, \ldots, c_m\}$ over variables $V = \{v_1, \ldots v_n\}$ and an integer k, we construct a planning domain $D = \langle F, N_p, \emptyset, \delta, A, \emptyset\rangle$ and a planning problem $\Omega = \langle D, tn_I, \emptyset\rangle$ such that:

- for each $c_i \in C$, we have a proposition $f_{c_i} \in F$ which represents whether that clause is satisfied or not,
- for each $v_j \in V$, we have:
  - two propositions $f_{v_j}, f_{\neg v_j} \in F$ which represent the variable assignments, and
  - one probabilistic action $p_j \in N_p$ with no precondition or delete effect which either unconditionally adds $f_{v_j} \cup \{f_{c_i} \mid v_j \textit{ satisfies clause } c_i\}$ or $f_{\neg v_j} \cup \{f_{c_i} \mid \neg v_j \textit{ satisfies clause } c_i\}$ with each effect having $\frac{1}{2}$ chance of occurring.

Hierarchy	Ordering	Solution Type			
		ρ-Linearization	Proof	ρ-Policy	Proof
Primitive	Total	$\mathbb{PP}$-Complete	Thm. 2	$\mathbb{PP}$-Complete	Thm. 1 and Thm. 2
	Partial	$\mathbb{NP}^{\mathbb{PP}}$-Complete	Thm. 3	$\mathbb{PSPACE}$-Complete	Thm. 8
Acyclic	Total	in $\mathbb{NEXPTIME}^{\mathbb{PP}}$	Thm. 4	in $\mathbb{NEXPTIME}^{\mathbb{PP}}$	Thm. 1 and Thm. 4
	Partial	in $\mathbb{NEXPTIME}^{\mathbb{PP}}$	Thm. 4	in $\mathbb{NEXPTIME}^{\mathbb{PSPACE}}$	Thm. 9
Tail-Recursive	Total	$\mathbb{UNDECIDABLE}$	Thm. 5	$\mathbb{UNDECIDABLE}$	Thm. 1 and Thm. 5
	Partial	$\mathbb{UNDECIDABLE}$	Cor. 6	$\mathbb{UNDECIDABLE}$	Cor. 10
Arbitrary	Total	$\mathbb{UNDECIDABLE}$	Cor. 7	$\mathbb{UNDECIDABLE}$	Thm. 1 and Cor. 7
	Partial	$\mathbb{UNDECIDABLE}$	Cor. 11	$\mathbb{UNDECIDABLE}$	Cor. 11

Table 1: Complexity results for probabilistic HTN planning. For decision problems where only a membership (i.e., an upper bound) is provided, the hardness (i.e., a lower bound) follows from the corresponding class in non-deterministic HTN planning (Chen and Bercher 2021). In particular, deciding whether a partially ordered acyclic problem is ρ-linearizable is $\mathbb{NEXPTIME}$-Hard, and the totally ordered case is $\mathbb{PSPACE}$-Hard. To the best of our knowledge no lower bound is known for deciding ρ-policies in acyclic problems.

- we have an action $is\_sat \in N_p$ with no effect and $\{f_{c_1}, \dots f_{c_m}\}$ as its precondition which checks whether all clauses are satisfied or not.
- the initial task network, tn_I has $n + 1$ totally ordered tasks which correspond to the action sequence $p_1, \dots, p_n, is\_sat$.

The next step is to construct the probability threshold, ρ. We know that there are 2^n possible assignment to the variables. Thus, $\rho = \frac{k+1}{2^n}$ ensures that the number of satisfying assignments exceed k. Even though, the denominator is exponential, we only need $\lfloor \log_2 2^n \rfloor + 1$ bits to represent it in a binary notation. Thus, ρ is constructible in $\mathbb{PTIME}$. As the last step we show that there are more than k satisfying assignments to ζ if and only if Ω is ρ-linearizable.

$\Rightarrow$) Assume ζ has exactly k' satisfying assignments ($k' \geq k + 1$), then by construction the precondition of is_sat is satisfied in k' trajectories out of 2^n possible ones. The rest of the actions do not have any preconditions. Thus, Ω is $(\frac{k+1}{2^n})$-linearizable.

$\Leftarrow$) Assume Ω is $(\frac{k+1}{2^n})$-linearizable. Since the effects of each action p_j are uniformly distributed between two effects and there are n actions, we have 2^n conformant trajectories (each with a probability of $\frac{1}{2^n}$). The precondition of is_sat is satisfied in at least $k + 1$ of these trajectories (exceeding k). Given that this precondition requires all clauses in ζ to be simultaneously satisfied, and considering our space of 2^n possible variable assignment, we can conclude that ζ must have at least $k + 1$ unique satisfying assignments. □

Next, we show that allowing partial ordering of the tasks increases the complexity. Our reduction is from E-MAJSAT, which is $\mathbb{NP}^{\mathbb{PP}}$-Complete (Littman, Goldsmith, and Mundhenk 1998).

Definition 25 (E-MAJSAT). *Let ζ be a boolean formula in conjunctive normal form over variables $V = \{v_1, \dots, v_n\}$, and k be an integer such that $1 \leq k \leq n$. The E-MAJSAT problem asks whether there is an initial partial assignment to variables $v_1, \dots, v_k$ so that the majority of assignments that extend that partial assignment satisfy ζ.*

Theorem 3. *Let $\Omega = \langle D, tn_I, s_I \rangle$ be a primitive HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization exists for Ω is $\mathbb{NP}^{\mathbb{PP}}$-Complete.*

Proof. (Membership)

1. guess a linearization of tn_I,
2. verify using the $\mathbb{PP}$ membership procedure of Thm. 2's proof.

(Hardness) We reduce from the E-MAJSAT problem. Given a boolean formula ζ as a set of clauses $C = \{c_1, \dots, c_m\}$ over variables $V = \{v_1, \dots v_n\}$, and an integer k, we construct a planning domain $D = \langle F, N_p, \emptyset, \delta, A, \emptyset \rangle$, similar to the one in the hardness proof of Thm. 2, such that:

- $F = C' \cup V' \cup O$ where:
  - $C' = \{f_{c_i} \mid c_i \in C\}$ tracks which clauses are satisfied,
  - $V' = \{f_{v_j}, f_{\neg v_j} \mid v_j \in V\}$ tracks variable assignments, and
  - $O = \{o_{v_j} \mid v_j \in V\}$ ensures each variable is assigned once.
- for each $v_j \in v_1 \dots v_k$, there are two deterministic actions $p_j, \neg p_j \in N_p$ where:
  - *pre*$(p_j) =$ *pre*$(\neg p_j) = \emptyset$,
  - *eff*$(p_j) = \{\langle \{o_{v_j}\}, \{f_{v_j}\} \cup \{f_{c_i} \mid v_j$ satisfies clause $c_i\}, \{o_{v_j}\} \rangle\}$, and
  - *eff*$(\neg p_j) = \{\langle \{o_{v_j}\}, \{f_{\neg v_j}\} \cup \{f_{c_i} \mid \neg v_j$ satisfies clause $c_i\}, \{o_{v_j}\} \rangle\}$.
- for each $v_j \in v_{k+1} \dots v_n$, there is one probabilistic action $p'_j \in N_p$ with no precondition or delete effect which either unconditionally adds $f_{v_j} \cup \{f_{c_i} \mid v_j$ *satisfies clause* $c_i\}$ or $f_{\neg v_j} \cup \{f_{c_i} \mid \neg v_j$ *satisfies clause* $c_i\}$ with each effect having $\frac{1}{2}$ chance of occurring.

- there is a deterministic action is_sat with *eff*$(is\_sat) = \emptyset$, and *pre*$(is\_sat) = C'$.

Next, we construct a planning problem $\Omega = \langle D, tn_I, s_I \rangle$ with $s_I = O$, and $tn_I = \langle N_p, \prec, \{x \mapsto x \mid x \in N_p\}\rangle$ where:

$$\begin{aligned}\prec = &\{(x, y) \mid x \in \{p_i, \neg p_i\}, y \in \{p_{i+1}, \neg p_{i+1}\}, 1 \leq i < k\}\\ &\cup \{(p_k, p'_{k+1}), (\neg p_k, p'_{k+1})\}\\ &\cup \{(p'_i, p'_{i+1}) \mid k < i < n\}\\ &\cup \{(x, is\_sat) \mid x \in (N_p \setminus \{is\_sat\})\}\end{aligned}$$

Ω is $\frac{1}{2}$-linearizable if and only if the E-MAJSAT problem has a solution.

$\Rightarrow$) Assume $\mathcal{A} = \{\mathcal{A}_{v_1}, \mathcal{A}_{v_2}, \ldots, \mathcal{A}_{v_k}\}$ is a partial assignment to V such that the majority of assignments to the remaining variables satisfy ζ. By construction, there exists a partial plan (which forms a prefix) consisting of $2k$ steps comprising unordered pairs of p_i and $\neg p_i$ for each $1 \leq i \leq k$. The order of each pair denotes which assignment is chosen for variable v_i. Thus, an action sequence corresponding to $\mathcal{A}_{v_1}, \neg\mathcal{A}_{v_1}, \ldots, \mathcal{A}_{v_k}, \neg\mathcal{A}_{v_k}$ sets the state to $\mathcal{A}$. After this, we have $p'_{k+1}, \ldots, p'_n$ where each corresponding action randomly assigns a value to one of the remaining variables. Since none of these actions have any precondition, they are always executable. Lastly, is_sat action is executable if and only if all clauses are satisfied. By assumption, the majority of the remaining variables satisfy ζ, which means more than half of the conformant trajectories satisfy the precondition of is_sat. Hence, Ω is $\frac{1}{2}$-linearizable.

$\Leftarrow$) Assume Ω is $\frac{1}{2}$-linearizable. By construction, there exists a plan whose prefix is a sequence of p_i and $\neg p_i$ pairs in some order, with $1 \leq i \leq k$. The state of variable f_{v_i} and $f_{\neg v_i}$ after executing this prefix corresponds to the variable assignments caused by the first action of each pair. In other words, f_{v_i} and its negation $\neg f_{v_i}$ are uniquely determined for all $i \leq k$ after executing the prefix. Since these actions do not have any non-deterministic effect, they are fixed for every conformant trajectory. The actions following this prefix branch over each variable assignment to create a new trajectory. By construction, is_sat is executable if and only if every clause is satisfied. By assumption, the majority of trajectories succeed. Thus, the prefix gives us the assignment for the first k variables where the majority (i.e., $\geq \frac{1}{2}$) of the assignments to the remaining variables satisfy ζ. □

The next theorem shows an upper bound on the complexity of deciding whether a ρ-linearization exists under the assumption that compound tasks exist in the initial task network, but none of the methods are recursive.

Theorem 4. *Let $\Omega = \langle D, tn_I, s_I \rangle$ be an acyclic HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization exists for Ω is in $\mathbb{NEXPTIME}^{\mathbb{PP}}$.*

Proof. (Membership) Since recursion is not possible in acyclic problems, each compound task in the domain can be refined to primitive tasks, at worst case, using exponentially many decompositions (Alford, Bercher, and Aha 2015). Thus, there is a decomposition sequence $\Lambda = d_1, \ldots, d_n$ where $n \leq |T| \cdot 2^k$ (for some positive integer k) that refines tn_I to a primitive task network tn_p, that is at most exponentially larger than tn_I.

1. guess and apply Λ to obtain a primitive task network $tn_p = \langle T_p, \prec_p, \alpha_p, \rangle$.
2. guess a total order $\prec'_p$ that is consistent with $\prec_p$, and construct $tn'_p = \langle T_p, \prec'_p, \alpha_p, \rangle$
3. use the $\mathbb{PP}$ membership proof of Thm. 2 to check whether $\langle D, tn'_p, s_I \rangle$ is ρ-linearizable.

□

For the undecidability results, we reduce from the emptiness problem of a simple probabilistic automaton (Gimbert and Oualhadj 2010).

Definition 26 (Emptiness of Simple Probabilistic Automaton)**.** *Given a $\lambda \in [0, 1]$ and a probabilistic automaton $\mathcal{A} = \langle Q, \Gamma, \Delta, q_0, \mathcal{F}\rangle$ where:*

- *Q is a finite set of states,*
- *Γ is a finite set of input alphabet,*
- *Δ is a finite set of $|Q| \times |Q|$ probabilistic transition matrices where for each $l \in \Gamma$ there exists a Δ_l where each of its rows defines the probability of next state given that the current character is l and current state is the one corresponding to the row,*
- *$q_0 \in Q$ is the initial state, and*
- *$\mathcal{F} \subseteq Q$ is the set of final states.*

The emptiness problem asks whether there exists a word ω such that $\mathcal{A}$ accepts ω with probability greater than or equal to λ, where acceptance means the automaton transitions from its initial state q_0 to some accepting state $f \in \mathcal{F}$ (denoted as $\mathcal{P}_{\mathcal{A}}(\omega) \geq \lambda$). A probabilistic automaton is called simple if every transition probability in Δ is in $\{0, 0.5, 1\}$.

It has been shown that the emptiness problem is undecidable for rational thresholds, too (Rote 2024). Based on this, the next theorem shows that severely restricted cases of probabilistic tail-recursive problems are undecidable.

Theorem 5. *Let $\Omega = \langle D, tn_I, s_I \rangle$ be a totally-ordered tail-recursive HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization exists for Ω is $\mathbb{UNDECIDABLE}$. The result holds even if for every action a and effect E, we have $\mathcal{P}_a(E) \in \{0, 0.5, 1\}$.*

Proof. We reduce from the $\mathbb{UNDECIDABLE}$ emptiness problem of a simple probabilistic automaton (Gimbert and Oualhadj 2010). Formally, given $\mathcal{A} = \langle Q, \Gamma, \Delta, q_0, \mathcal{F}\rangle$ and $\lambda \in [0, 1]$, we construct a totally-ordered tail-recursive HTN domain $D = \langle F, N_p, N_c, \delta, A, M\rangle$ such that:

- $F = Q \cup \Gamma \cup \{f\}$ where fact f indicates reaching a final state,
- $N_p = N_\Gamma \cup N_\Delta \cup \{$*is_final*$\}$ where:
  - $N_\Gamma = \{l \mid l \in \Gamma\}$,
  - $N_\Delta = \{n^l_q \mid n^l_q \in Q \times \Gamma\}$,

- $N_c = \{C_{guess}, C_{simulate}\}$,
- A is the minimal set satisfying:
  1. for each $n_l \in N_\Gamma$, there exists a deterministic action a such that $pre(a) = \emptyset$ and $eff(a) = \{\langle \emptyset, \{l\}, \Gamma \setminus \{l\}\rangle\}$,
  2. for each $n^l_q \in N_\Delta$, there exists an action a with $pre(a) = \emptyset$ and the following effects:
     - If the row corresponding to state q in Δ_l has two non-zero cells (leading to either state q' or q''), then we have two conditional effects (each with probability equal to 0.5). The condition for both of them is $\{q, l\}$ (i.e., we are at state q, and the token l has not been consumed). If a conditional effect is fired, state q and token l are consumed (i.e., deleted from the state). The added propositions differ in whether q' is added or q''. Additionally, f is added if the new state is a final one. Formally:
$$\text{eff}_1(a) = \{\langle \{q,l\}, \begin{Bmatrix} q', f & \textit{if } q' \in \mathcal{F} \\ q' & \textit{if } q' \notin \mathcal{F} \end{Bmatrix}, \{q,l\}\rangle\}$$
$$\text{eff}_2(a) = \{\langle \{q,l\}, \begin{Bmatrix} q'', f & \textit{if } q'' \in \mathcal{F} \\ q'' & \textit{if } q'' \notin \mathcal{F} \end{Bmatrix}, \{q,l\}\rangle\}$$
       with $\mathcal{P}_a(\text{eff}_1(a)) = \mathcal{P}_a(\text{eff}_2(a)) = 0.5$.
     - If the transition is deterministic (i.e., there is only one non-zero cell transitioning to q'), then $eff(a) = \{\langle \{q,l\}, \{q'\}, \{q,l\}\rangle\}$ iff $q' \notin \mathcal{F}$. Otherwise, $eff(a) = \{\langle \{q,l\}, \{q', f\}, \{q,l\}\rangle\}$.
  3. there exists a deterministic action *is_final* with $pre(\textit{is\_final}) = \{f\}$, and $eff(\textit{is\_final}) = \emptyset$.
- δ is a canonical mapping,
- $M = M_{guess} \cup M_{simulate}$ where:
  - $M_{guess} = \{(C_{guess}, \langle \{1\}, \emptyset, \{1 \mapsto a\}\rangle) \mid a \in N_\Gamma\}$
  - $M_{simulate}$ consists of two methods. The first one is a totally-ordered tail recursive method that decomposes $C_{simulate}$ to a task network corresponding to the sequence $C_{guess}, n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}, C_{simulate}$ where $\{n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}\} = N_\Delta$. The other one is used to terminate the loop by decomposing $C_{simulate}$ to a single instance of the primitive task, *is_final*.

The probabilistic planning problem is $\Omega = \langle D, \langle \{1\}, \emptyset, \{1 \mapsto C_{simulate}\}\rangle, \{q_0\}\rangle$. Ω is λ-linearizable if and only if there exists a word ω such that $\mathcal{P}_\mathcal{A}(\omega) \geq \lambda$.

$\Rightarrow$) Assume there exists a word ω such that $\mathcal{P}_\mathcal{A}(\omega) \geq \lambda$. Given an initial unit vector of the starting state in the automaton, each transition triggers a matrix multiplication. To simulate this, we need to simulate both multiplication and summation of rational numbers. The multiplication is achieved by executing probabilistic actions, which multiply the probability of a particular trajectory with the rational number associated with the probability of the effect. The summation occurs over the trajectories that are induced by executing those actions. The combination of these two allows us to do matrix multiplication, which results in simulating the transitions. Formally, On a single letter l of ω, the state distribution vector $[q_0\ q_1\ \ldots\ q_m]$ changes based on Δ_l. And, each action in $\sigma = n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}$, corresponds to one row of the transition matrices. The condition $\{q, l\}$ prevents them from firing the outcome if the letter is not l. Thus, for any single letter produced by C_{guess}, only Δ_l is simulated. As such, each trajectory induced by σ corresponds to one single transition of the automaton. The reason is that l (and its removal) forces only one of the automaton state transitions to fire in each iteration. Lastly, C_{guess} makes an arbitrary guess of which letter to produce next. Hence, it can iteratively guess a letter to obtain the following primitive task network:

$$n_{l_1}, n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}, n_{l_2}, n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}, \ldots, n_{l_k}, \ldots, \textit{is\_final}$$

where the probability of successful execution is greater than or equal to λ and $n_{l_1} n_{l_2} \ldots n_{l_k} = \omega$. As such, Ω is λ-linearizable.

$\Leftarrow$) Assume Ω is λ-linearizable. By definition, there exists a primitive task network with the following structure.

$$n_{l_1}, n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}, n_{l_2}, n^{l_0}_{q_0}, \ldots, n^{l_n}_{q_m}, \ldots, n_{l_k}, \ldots, \textit{is\_final}$$

where the probability of successful execution is greater than or equal to λ. Thus, $\mathcal{P}_\mathcal{A}(l_1 l_2 \ldots l_k) \geq \lambda$ where $l_1 l_2 \ldots l_k$ is the *guessed* word. □

Since totally-ordered tail-recursive problems are restricted cases of more general problems, the general ones (i.e., partially-ordered tail-recursive and totally-ordered with no restriction on recursion) are also undecidable.

Corollary 6. *Let $\Omega = \langle D, tn_I, s_I\rangle$ be a tail-recursive HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization exists for Ω is* $\mathbb{UNDECIDABLE}$.

Corollary 7. *Let $\Omega = \langle D, tn_I, s_I\rangle$ be a totally-ordered HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization exists for Ω is* $\mathbb{UNDECIDABLE}$.

4.2 Complexity Results for ρ-Policy Existence

Unlike linear solutions where probabilistic versions are more complex to decide than non-deterministic ones, probabilistic and non-deterministic hierarchical policies have more or less the same computational complexity (Chen and Bercher 2021). We demonstrate this by reducing the non-deterministic policy existence problem to its probabilistic counterpart for primitive planning problems.

Theorem 8. *Let Ω be a primitive HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-policy exists for Ω is* $\mathbb{PSPACE}$*-Complete.*

Proof. (Membership) The main idea is that it is possible to systematically enumerate every possible policy (to be more specific, their execution structures) and run a Depth-First search with a polynomial bound on space to check whether this policy succeeds with a probability greater than or equal to ρ. Assume tn_I is the initial task network with n tasks. The total number of possible policies for tn_I in the worst case is equal to the set of all permutations of n tasks, which can be enumerated in finite amount of time. In all of them, the diameter of the execution graph (i.e., the longest distance between two vertices of a graph) is $n + 2$ where one extra is used for the initial node with no executed task and the other one for the extra node after executing all n tasks. Thus, the time (and space) required to calculate the probability of each branch is clearly polynomially bounded. The branches can be recursively generated based on: (1) the sequence of actions that have been executed, (2) the state under which they were executed in order to backtrack and select the next outcome, and (3) the accumulated probability of the current branch. Notably, the backtracking in step 2 forgets everything after the backtracking point and jumps to the next outcome of the "current" action. This approach prevents us from storing the entire execution structure (which is exponentially larger) and ensures that we don't revisit the same branch. As a final detail, we need to keep track of the accumulated probability over all branches for evaluating the entire policy. All of the mentioned information can be stored in $O(|T| + b)$ space where b is the number of bits required to represent the probabilities. Thus, the overall procedure is in $\mathbb{PSPACE}$.

(Hardness) A strong policy in non-deterministic setting is a special case of a ρ-policy where $\rho = 1$. Thus, the $\mathbb{PSPACE}$ hardness follows from the fact that finding a strong policy for a primitive task network is $\mathbb{PSPACE}$-Complete (Chen and Bercher 2021). $\square$

The complexity of policy existence in the acyclic case is similar to the linear solution existence in the same setting.

Theorem 9. *Let $\Omega = \langle D, tn_I, s_I \rangle$ be a partially-ordered acyclic HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-policy exists for Ω is in $\mathbb{NEXPTIME}^{\mathbb{PSPACE}}$.*

Proof. As mentioned in the membership proof of Thm. 4, there is a decomposition sequence $\Lambda = d_1, \ldots d_n$ where $n \leq |T| \cdot 2^k$ (for some positive integer k) that refines tn_I to a primitive task network tn_p, where tn_p is at most exponentially larger than tn_I.

1. guess and apply Λ to obtain a primitive task network tn_p.
2. use the $\mathbb{PSPACE}$ membership proof of Thm. 8 to check whether $\langle D, tn_p, s_I \rangle$ has a ρ-policy.

$\square$

From Theorems 1 and 5, it follows that finding a ρ-policy for a totally-ordered probabilistic HTN planning problem is undecidable. Thus, the general case for partially-ordered problems are also undecidable.

Corollary 10. *Let $\Omega = \langle D, tn_I, s_I \rangle$ be a partially-ordered tail-recursive probabilistic HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-policy exists for Ω is $\mathbb{UNDECIDABLE}$.*

It is known that partially-ordered HTN planning with no restriction on hierarchy is $\mathbb{UNDECIDABLE}$ (Erol, Hendler, and Nau 1996; Geier and Bercher 2011). Since probabilistic planning is a generalization of that, it is also $\mathbb{UNDECIDABLE}$.

Corollary 11. *Let $\Omega = \langle D, tn_I, s_I \rangle$ be a partially-ordered HTN planning problem. Let ρ be a rational number, represented by two binary numbers, where $0 < \rho \leq 1$. Deciding whether a ρ-linearization or a ρ-policy exists for Ω is $\mathbb{UNDECIDABLE}$.*

5 Conclusion

We present a formal framework for incorporating probabilistic outcomes into Hierarchical Task Network planning, a key step towards real-world applicability by moving beyond rigid non-deterministic assumptions. Our formalism allows integrating procedural constraints and hierarchical abstraction into Markov Decision Processes and their associated solutions, providing a unified approach for reasoning about task decomposition hierarchies and probabilistic action outcomes. The paper includes an analysis of the framework's computational complexity under different constraints and solution types. We proved that probabilistic conformant solutions (fixed action sequences) are consistently harder than their non-deterministic counterparts. Even with severe restriction, the solution existence problem is undecidable. Branching solutions (i.e., policies) are proven to be even harder in some of the subclasses. This work establishes a solid theoretical foundation for future research. Potential directions include devising algorithms and heuristics, as well as analyzing the complexity of more flexible solution criteria, such as allowing decomposition during execution.

Acknowledgements

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. We extend our appreciation to the anonymous reviewers whose insightful comments helped us identify and correct some errors from the first submission.

References

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight bounds for HTN planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 7–15. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN problem spaces: Structure, algorithms, termination. In *Proceedings of the 5th Symposium on Combinatorial Search (SoCS)*, 2–9. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A survey on hierarchical planning-one abstract idea, many concrete realizations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 6267–6275. IJCAI.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence Journal (AIJ)* 170:355–364.

Chen, D., and Bercher, P. 2021. Fully observable nondeterministic HTN planning – formalisation and complexity results. In *Proceedings of the 31st International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, 74–84. AAAI Press.

Chen, D., and Bercher, P. 2022. Flexible FOND HTN planning: A complexity analysis. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*, 26–34. AAAI Press.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence Journal (AIJ)* 147(1):35–84.

Cozman, F. G., and Mauá, D. D. 2018. The complexity of bayesian networks specified by propositional and relational languages. *Artificial Intelligence Journal (AIJ)* 262:96–141.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)* 13(1):227–303.

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

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 1955–1961. AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Gill, J. T. 1974. Computational complexity of probabilistic turing machines. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, 91–95. Association for Computing Machinery.

Gimbert, H., and Oualhadj, Y. 2010. Probabilistic automata on finite words: decidable and undecidable problems. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming: Part II*, 527–538. Springer-Verlag.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc.

Hyafil, N., and Bacchus, F. 2004. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 1033–1034. IOS Press.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence Journal (AIJ)* 76(1–2):239–286.

Kuter, U., and Nau, D. 2004. Forward-chaining planning in nondeterministic domains. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, 513–518. AAAI Press.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research (JAIR)* 9:1–36.

Littman, M. L. 1997. Probabilistic propositional planning: representations and complexity. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, 748–754. AAAI Press.

Majercik, S. M., and Littman, M. L. 2003. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence Journal (AIJ)* 147(1-2):119–162.

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of markov decision processes. *Mathematics of Operations Research* 12(3):441–450.

Parr, R., and Russell, S. 1997. Reinforcement learning with hierarchies of machines. In *Proceedings of the 11th International Conference on Neural Information Processing Systems (NeurIPS)*, 1043–1049. MIT Press.

Patra, S.; Ghallab, M.; Nau, D.; and Traverso, P. 2019. Acting and planning using operational models. In *Proceedings of the 19th AAAI Conference on Artificial Intelligence (AAAI)*, 7691–7698. AAAI Press.

Patra, S.; Mason, J.; Ghallab, M.; Nau, D.; and Traverso, P. 2021. Deliberative acting, planning and learning with hierarchical operational models. *Artificial Intelligence Journal (AIJ)* 299:103523.

Rote, G. 2024. Probabilistic finite automaton emptiness is undecidable. *arXiv Preprint 2405.03035*.

Simon, J. 1975. *On some central problems in computational complexity.* Ph.D. Dissertation, USA. AAI7518004.

Sipser, M. 2020. *Introduction to the Theory of Computation*. Course Technology, 3rd edition.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence Journal (AIJ)* 112(1):181–211.