ABSTRACT

Title of Dissertation:	GOAL DECOMPOSITION FOR PROBABILISTIC PLANNING
	David H. Chan Doctor of Philosophy, 2026
Dissertation Directed by:	Professor Emeritus Dana S. Nau Department of Computer Science

Automated planning is essential for many applications, such as autonomous systems, robotics, and intelligent decision support, where agents must make sequential decisions and execute actions to achieve defined goals. However, as planning domains grow in scale and complexity, classical planning techniques sometimes struggle to generate effective plans. Goal decomposition is an approach to classical planning that applies divide-and-conquer principles to break down planning problems into smaller, more manageable subproblems, allowing planning systems to reason about each subproblem incrementally. Even so, the classical planning paradigm assumes complete, deterministic information about the environment and the effects of actions—an assumption that often does not hold in dynamic, real-world scenarios.

This research builds upon two fundamental goal decomposition approaches from classical planning and extends them to probabilistic planning: (1) landmarks, which are intermediate conditions satisfied at some point along every solution plan, and (2) hierarchical goal networks,

which provide methods for decomposing goals into structured sequences of subgoals. For each approach, we develop extensions to Monte Carlo tree search that leverage goal decomposition information to guide planning under uncertainty. A comparative analysis of these methods leads to a unified framework for goal-network-directed probabilistic planning, in which both planners emerge as specific instantiations of a more general algorithm. By bridging goal decomposition and probabilistic planning, this research develops more scalable planning techniques for complex, uncertain domains, improving the applicability of planning systems to real-world sequential-decision-making problems.

GOAL DECOMPOSITION FOR PROBABILISTIC PLANNING

by

David H. Chan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:
Professor Emeritus Dana S. Nau, *Chair*
Associate Professor Michael W. Otte, *Dean's Representative*
Associate Professor Furong Huang
Professor David M. Mount
Dr. Mark Roberts

© Copyright by
David H. Chan
2026

Dedication

To my parents.

Acknowledgments

Many thanks are due to everyone who has made this dissertation possible. First and foremost, I am deeply thankful to my advisor, Prof. Dana Nau. He has fostered a supportive and intellectually rigorous research environment, and has always been approachable and generous with his time, offering thoughtful feedback while giving me the space to explore ideas and chart my own direction. Even after his retirement and transition to professor emeritus, he was committed to graduating me and his remaining students. I feel very fortunate for the privilege of working with and learning from such a widely respected and dedicated advisor.

I am also deeply grateful to Dr. "Mak" Roberts, who has served as my co-advisor for the past three years following the start of my internship at the U.S. Naval Research Laboratory. He has always taken exceptional care in mentoring his students; his ability to articulate and contextualize his thinking and research process, highlighting common challenges and key lessons, has been invaluable to my own development as a researcher. After recently leaving the U.S. Naval Research Laboratory, he was also committed to seeing me through to graduation, for which I am very thankful.

I would also like to thank my advisory committee for their valuable feedback on my dissertation manuscript: Prof. Michael Otte, Prof. David Mount, and Prof. Furong Huang. I am especially grateful to Prof. Huang, who was my first advisor at the University of Maryland and with whom I

worked during the first year of my PhD program. She was welcoming and supportive, providing guidance that helped me transition into graduate research and set the foundation for my work.

I am also thankful to many colleagues and collaborators, including Dr. Laura Hiatt and the Adaptive Systems Section at the U.S. Naval Research Laboratory, Prof. V.S. Subrahmanian and his lab at Northwestern University, and Dr. Li Ling at Bloomberg L.P. Their expertise and guidance helped shape my academic career. This work was supported in part by the U.S. Naval Research Laboratory.

Table of Contents

List of Algorithms

List of Figures

Chapter 1: Introduction

In decision-making for artificial intelligence, automated planning is the task of deciding what actions an agent needs to perform to achieve a given goal [37], [38]. However, as the scale and complexity of automated planning domains increase, standard planning approaches are often overwhelmed by large problems. In computer science, divide-and-conquer techniques are a fundamental approach for tackling complex problems by breaking them into smaller, more manageable components, solving these components individually, and combining their solutions. Within classical planning, goal decomposition applies this principle by systematically decomposing planning problems into smaller subproblems, allowing planning systems to reason about each subproblem incrementally.

The core aspect of goal decomposition is the use of subgoals, which are often encoded in structures such as *landmark graphs* and *goal networks*, to guide the planning process. Landmarks—conditions that must be satisfied at some point in any planning solution—serve as guideposts by highlighting intermediate states that must be reached on the way to achieving the goal [43]. Goal networks, on the other hand, represent the hierarchical structure of planning problems by using human-written methods to recursively break down goals into sequences of subgoals [71]. These structures enable classical planning systems to improve scalability and efficiency by guiding the

planner toward promising regions of the state space.

However, a major limitation of classical planning is its reliance on the assumption that actions have fully deterministic effects, limiting its applicability to real-world planning problems where uncertainty about the environment is often inherent [56], [57]. Probabilistic planning addresses this by modeling uncertainty in the outcomes of actions. Yet, despite significant advances in probabilistic planning algorithms, most approaches reason directly over the stochastic state space without explicitly leveraging structured goal decompositions that have proven effective in classical, deterministic planning.

This dissertation investigates how goal decomposition techniques from classical planning can be extended to probabilistic planning in order to improve scalability and performance in nondeterministic domains. Specifically, we generalize landmark-based and hierarchical goal-network-based approaches to probabilistic settings and study how these goal structures can guide Monte Carlo tree search (MCTS), a widely used algorithm for online decision-making [14]. By incorporating structured subgoals into MCTS, the planner can focus exploration toward key intermediate states rather than relying solely on simulation to discover effective strategies.

1.1 Contributions

This dissertation makes several contributions to the integration of goal decomposition with probabilistic planning. First, we extend the definition of classical landmarks to probabilistic planning domains. In formalizing *probabilistic landmarks*, we enable their use as intermediate objectives in probabilistic planning. Building on this formalism, we develop the LAMP algorithm, a domain-independent probabilistic planner that incorporates probabilistic landmarks into MCTS

to guide exploration toward these key subgoals. Empirical evaluation across several benchmark domains demonstrates that landmark-guided planning can significantly improve performance over standard MCTS, particularly in large problems or time-constrained probabilistic planning settings.

Second, we introduce a formalism for *probabilistic hierarchical goal network* (HGN) planning, extending classical goal hierarchies to probabilistic domains. Within this framework, we develop two MCTS-based planning algorithms: a baseline approach that directly searches over an augmented state space as a stochastic shortest path problem, and a compressed variant that leverages the structure of the goal network to guide search. Experimental results across several probabilistic planning benchmarks show that explicitly reasoning about goal hierarchies improves both scalability and solution quality.

Third, we conduct a comparative analysis to investigate the fundamental trade-offs between our two approaches. This analysis motivates the final contribution of this work: a unified framework for goal-network-directed probabilistic planning. This framework reveals both the landmark- and HGN-guided planning algorithms as specific instantiations of a more general goal-network-directed probabilistic planning algorithm, clarifying their relationship and highlighting their shared structural principles. Together, these contributions demonstrate that structured goal decompositions can significantly improve the efficiency of probabilistic planning, enabling planning systems to reason more effectively about complex sequential decision-making problems in uncertain environments.

1.2 Dissertation Overview[1]

The remainder of this dissertation is organized as follows:

- Chapter 2 reviews some fundamentals related to classical planning, probabilistic planning, and Monte Carlo–based planning algorithms.
- Chapter 3 formalizes probabilistic landmarks and landmark orderings using knowledge transfer from classical planning, and develops the LAMP algorithm for landmark-guided Monte Carlo planning.
- Chapter 4 formalizes probabilistic HGN planning and introduces $\text{UCT}^{comp}_{\text{HGN}}$, a Monte Carlo planning algorithm that leverages hierarchical goal decompositions to guide rollouts.
- Chapter 5 presents a comparative analysis of the landmark-based and HGN-based planning algorithms and introduces a unified framework for goal-network-directed probabilistic planning using MCTS.
- Chapter 6 concludes with an overview and discussion of our contributions.

[1] Parts of this dissertation are based on previously published work. Chapter 3 is based on:

- [18] D. H. Chan et al., "Landmark-assisted Monte Carlo planning," in ECAI 2025.

Chapter 4 is based on:

- [22] D. H. Chan et al., "Probabilistic hierarchical goal network planning with UCT," in AAAI 2026.
- [21] D. H. Chan et al., "Probabilistic hierarchical goal network planning: Preliminary results," in HPlan 2025.

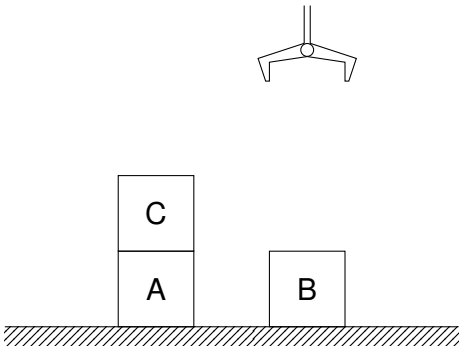
Other publications not directly related to this work are listed in Appendix C.

Chapter 2: Background

The development of automated planning techniques is driven, in large part, by the need for sequential decision-making in real-world problems. This chapter reviews the foundational concepts that form the basis of our work. We begin with classical planning, establishing the core formalisms for goal-directed sequential decision-making in deterministic settings. We then transition to probabilistic planning, which extends these formalisms to account for uncertainty in the effects of actions. Finally, we discuss Monte Carlo tree search (MCTS), a powerful algorithmic framework that has proven highly effective for decision-making in the large, uncertain state spaces characteristic of probabilistic domains. Throughout this chapter, we distinguish between the semantic models used to define planning problems (e.g., state transition systems and Markov decision processes) and the compact syntactic representations used in practice (e.g., STRIPS [32] and PDDL [40]). This distinction clarifies how planning domains are represented symbolically while their underlying state spaces may be exponentially large.

2.1 Classical Planning

Classical planning is a foundational area of artificial intelligence concerned with finding a sequence of actions to achieve a predefined goal. It provides a formal basis for sequential decision-



$\mathit{Blocks} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$

$\mathsf{on} : \mathit{Blocks} \times \mathit{Blocks} \cup \{\mathsf{table}\} \rightarrow \{\mathsf{true}, \mathsf{false}\}$
$\mathsf{holding} : \mathit{Blocks} \cup \{\mathsf{nil}\} \rightarrow \{\mathsf{true}, \mathsf{false}\}$
$\mathsf{clear} : \mathit{Blocks} \rightarrow \{\mathsf{true}, \mathsf{false}\}$

$g = \mathsf{on}(\mathsf{A}, \mathsf{B}) \wedge \mathsf{on}(\mathsf{B}, \mathsf{C})$

Figure 2.1: A depiction of a `blocksworld` problem with goal condition $g = \mathsf{on}(\mathsf{A}, \mathsf{B}) \wedge \mathsf{on}(\mathsf{B}, \mathsf{C})$. State variable $\mathsf{on}(b_1, b_2)$ indicates whether block b_1 is on top of block b_2, $\mathsf{holding}(b)$ indicates whether the gripper is holding block b, and $\mathsf{clear}(b)$ indicates whether the top of block b is clear. Actions in this domain include $\mathsf{stack}(b_1, b_2)$, $\mathsf{unstack}(b_1, b_2)$, $\mathsf{pickup}(b)$, and $\mathsf{putdown}(b)$.

making in a simplified, deterministic world model where the environment is fully observable and changes occur only as a result of the agent's actions. The framework rests upon a descriptive model of a planning domain, which specifies the properties of the environment using a collection of logical atoms over state variables. A *state* is modeled as a set of logical atoms describing the facts that are currently true. Any atoms not contained in the state are assumed to be false; this adopts the closed-world assumption typical of STRIPS-style planning [32]. For example, the `blocksworld` domain in Figure 2.1 shows a state

$$s = \{\mathsf{on}(\mathsf{A}, \mathsf{table}),\ \mathsf{on}(\mathsf{B}, \mathsf{table}),\ \mathsf{on}(\mathsf{C}, \mathsf{A}),\ \mathsf{holding}(\mathsf{nil}),\ \mathsf{clear}(\mathsf{C}),\ \mathsf{clear}(\mathsf{B})\}$$

using state variables on, holding, and clear.

Deterministic actions consist of preconditions, which define the conditions that must hold for an action to be applicable, and effects, which describe how the action transforms the state by adding or deleting specific atoms. In an action's effects, positive atoms are known as *add effects* and are added to the state, while negated atoms are known as *delete effects* and are removed from the state.

$\mathsf{stack}(b_1, b_2)$
pre: $\mathsf{holding}(b_1) \wedge \mathsf{clear}(b_2)$
eff: $\mathsf{on}(b_1, b_2), \neg\mathsf{holding}(b_1), \neg\mathsf{clear}(b_2)$

Figure 2.2: An action schema for stacking one block on another in the `blocksworld` domain. The effects contain positive atoms specifying the add effects of the action and negative atoms specifying the delete effects of the action.

Positive and negative atoms are known collectively as *literals*. Figure 2.2 shows an example of the stack action in the `blocksworld` domain in Figure 2.1.

To formalize this, let $\mathcal{L} = (\mathcal{X}, \mathcal{F})$ be a propositional language with atoms $\mathcal{X}$ and propositional formulae $\mathcal{F}$. A *classical planning domain* is a tuple $\mathcal{D}_C = (\mathcal{L}, A_C, \gamma_C)$, where

- $S = 2^{\mathcal{X}}$ is a finite set of states;
- $A_C \subseteq \mathcal{F} \times (2^{\mathcal{X}} \times 2^{\mathcal{X}})$ is a finite set of deterministic actions $a = (\text{pre}(a), \text{eff}(a)) \in A$ where $\text{pre}(a) \in \mathcal{F}$ is the precondition, and $\text{eff}(a) \in 2^{\mathcal{X}} \times 2^{\mathcal{X}}$ is a pair $(\text{add}(a), \text{del}(a))$ of add effects $\text{add}(a) \subseteq \mathcal{X}$ and delete effects $\text{del}(a) \subseteq \mathcal{X}$; and
- $\gamma_C : S \times A_C \to S$ is a transition function, defined on (s, a) iff s satisfies $\text{pre}(a)$, denoted $s \models \text{pre}(a)$, and $\gamma_C(s, a) = (s \backslash \text{del}(a)) \cup \text{add}(a)$.

An action $a \in A_C$ is *applicable* in state $s \in S$ if $s \models \text{pre}(a)$, denoted $a \in \text{Applicable}(s)$. For simplicity, we assume all actions have unit cost. We use the subscript C to indicate this is a classical domain.

In STRIPS-style planning, $\text{pre}(a)$ is typically a conjunction of positive atoms, though more expressive fragments allow arbitrary propositional formulae. Some richer action languages (e.g., ADL [58]) permit conditional and quantified effects, but many theoretical analyses assume the STRIPS subset. Note that while we define the state space as $S = 2^{\mathcal{X}}$, planning domains are typically represented compactly using schemas and operator templates, as in STRIPS and PDDL. The

explicit, exponential-sized state space is therefore not enumerated in practice; rather, planners reason symbolically over compact domain descriptions.

Given this domain model, a *classical planning problem* is defined as a tuple $\mathcal{P}_C = (\mathcal{D}_C, s_0, g)$. Here, $\mathcal{D}_C$ is the classical domain model, $s_0 \in S$ is the fully specified initial state, and $g \in \mathcal{F}$ is the goal condition. Any state $s \in S$ that satisfies the goal, i.e., $s \models g$, is considered a *goal state*. The objective of a classical planner is to find a *plan*, π_C, which is a sequence of actions $\pi_C = \langle a_1, a_2, \ldots, a_n \rangle$. A plan is considered a valid solution if it is executable (i.e., the preconditions of each action a_{i+1} are satisfied in the state s_i resulting from the execution of the preceding actions $\langle a_1, \ldots, a_i \rangle$) and if the final state reached after executing the entire sequence, s_n, satisfies the goal condition, i.e., $s_n \models g$.

Classical planning relies on several simplifying assumptions. It assumes a finite, static world, where changes occur only as a result of the agent executing an action. Actions are deterministic, with outcomes fully predictable from the current state. It also assumes full observability, with complete and accurate knowledge of the world at all times. Additionally, there is no explicit notion of time or concurrency; the world is modeled as a discrete sequence of states and actions, without accounting for action durations or simultaneous execution.

This symbolic framing of the planning problem lends itself well to formal analysis, enabling the development of algorithms with provable guarantees on soundness and completeness [37], [38]. The classical planning assumptions are adopted to improve computational tractability. Even so, the complexity of deciding plan existence in this simplified model is generally PSPACE-complete [16], highlighting the difficulty of planning even in this highly restricted setting. In practice, classical planners mitigate this complexity using heuristic search [12]. For example, many successful planners employ heuristics such as those derived from ignoring delete effects to estimate goal distance

efficiently [13], [42]. One influential technique in classical planning is the use of landmarks: conditions that must become true at some point along every valid plan [43]. Landmark graphs encode ordering constraints among such conditions and can be used to decompose a planning task. These decomposition structures will play a central role in the extensions developed in later chapters.

However, this tractability comes at the cost of model fidelity to the real world. In practice, environments are rarely static, observability is often partial due to sensor noise or occlusions, and concurrent actions are critical in many applications. A significant limitation for a vast range of problems is the assumption of determinism. Actions frequently have uncertain outcomes—a robot's gripper may fail to grasp an object, or a manufacturing process may have a random chance of failure. The inability of the classical framework to handle such possibilities is the primary motivation for the paradigm that will be central to our work: probabilistic planning, which relaxes the assumption of deterministic actions to formally model and reason about uncertainty.

2.2 Probabilistic Planning

Probabilistic planning extends the classical planning paradigm by relaxing the assumption of deterministic actions to address scenarios where action outcomes are uncertain. In a probabilistic domain, executing an action in a given state may lead to one of several possible successor states, each with a specific probability. For example, a probabilistic `stack` action for the `blocksworld` domain, shown in Figure 2.3, might succeed most of the time but have a small chance of failing and returning to the previous state. This uncertainty fundamentally changes the nature of a solution: instead of a sequence of actions that guarantees success, the objective becomes finding a course of action that maximizes the likelihood of success while minimizing expected costs.

stack-prob(b_1, b_2)
pre: holding(b_1) $\wedge$ clear(b_2)
eff: ($\neg$holding(b_1), on(b_1, b_2), $\neg$clear(b_2)) with probability 0.8
($\neg$holding(b_1), on(b_1, table)) with probability 0.2

Figure 2.3: A probabilistic action schema for stacking one block on another in the `blocksworld` domain, where there is a 20% chance the action fails and the block topples to the table.

Probabilistic planning is often viewed as a Markov decision process (MDP) or stochastic shortest path (SSP) problem, where a state transition function specifies the probabilities of reaching different states given a particular action and current state. To formalize our model of probabilistic planning, let $\mathcal{L} = (\mathcal{X}, \mathcal{F})$ be a propositional language with a set of atoms $\mathcal{X}$ and a set of propositional formulae $\mathcal{F}$. A *probabilistic planning domain* $\mathcal{D}$ is a tuple $(\mathcal{L}, A, \gamma, \Pr)$, where

- $S = 2^{\mathcal{X}}$ is a finite set of states;
- $A \subseteq \mathcal{F} \times 2^{2^{\mathcal{X}} \times 2^{\mathcal{X}}}$ is a finite set of nondeterministic actions $a = (\text{pre}(a), \text{eff}(a)) \in A$ where $\text{pre}(a) \in \mathcal{F}$ is the precondition, and $\text{eff}(a) \in 2^{2^{\mathcal{X}} \times 2^{\mathcal{X}}}$ is a set of pairs $(\text{add}(a), \text{del}(a))$ of add effects $\text{add}(a) \subseteq \mathcal{X}$ and delete effects $\text{del}(a) \subseteq \mathcal{X}$;
- $\gamma : S \times A \to 2^{2^{\mathcal{X}}}$ is a transition function where $\gamma(s, a)$ is defined iff $s \models \text{pre}(a)$, and $\gamma(s, a) = \{(s \backslash \text{del}(a)) \cup \text{add}(a) \mid (\text{add}(a), \text{del}(a)) \in \text{eff}(a)\}$; 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 $a \in A$ is *applicable* in state $s \in S$ if $s \models \text{pre}(a)$, or equivalently, if $\gamma(s, a) \neq \varnothing$; we denote this by $a \in \text{Applicable}(s)$. For simplicity, we assume all actions have unit cost. Probabilistic planning domains of this form are commonly specified in PPDDL, which extends PDDL with probabilistic effects [85].

Algorithm 2.1 (RUN-POLICY) A procedure to execute a policy π [38]. The method APPLY(s, a) performs action a in state s and returns the observed resulting state.

1: **procedure** RUN-POLICY$(\mathcal{D}, s_0, g, \pi)$
2: $\quad s \leftarrow s_0$
3: $\quad$ **while** $s \not\models g$ and $s \in \text{Dom}(\pi)$ **do**
4: $\quad\quad s \leftarrow \text{APPLY}(s, \pi(s))$

As with classical planning, a *probabilistic planning problem* is a tuple $\mathcal{P} = (\mathcal{D}, s_0, g)$, where s_0 is the fully specified initial state, and $g \in \mathcal{F}$ is the goal condition. However, unlike classical planning, for which a solution is a sequence of actions that guarantees reaching a goal, a solution for probabilistic planning is a policy—a partial function $\pi : S \rightarrow A$ that maps states to actions. For a given state s, $\pi(s)$ is an action applicable in s, and executing a policy means performing the actions specified by the policy until reaching a terminal state, as outlined in Algorithm 2.1.

While this framework is more robust for modeling real-world complexities like execution errors, this flexibility comes at a significant computational cost for finding solutions. Planners must reason about branching possibilities and their probabilities. Additionally, the presence of dead ends complicates optimization: minimizing expected cost alone may be ill-defined if some trajectories fail to reach the goal. As a result, optimization criteria may incorporate both the probability of goal achievement and the expected cost of trajectories, yielding utility-based formulations that trade off reliability and efficiency [33], [48], [80].

A variety of algorithmic strategies have been developed for solving probabilistic planning problems. Exact dynamic programming methods such as value iteration and policy iteration compute optimal value functions over the entire state space, but are often infeasible in large domains. In such cases, heuristic search methods, including algorithms such as LAO* [39] and RTDP [6], can

improve efficiency. Another common strategy in probabilistic planning is determinization, where a probabilistic problem is approximated by a deterministic version in order to guide search. For example, FF-Replan [84] repeatedly solves a determinized version of the problem using a classical planner, then replans online as action outcomes are observed. Finally, sampling-based approaches such as MCTS approximate optimal policies through simulation using a generative model of the domain.

2.3 Monte Carlo Tree Search

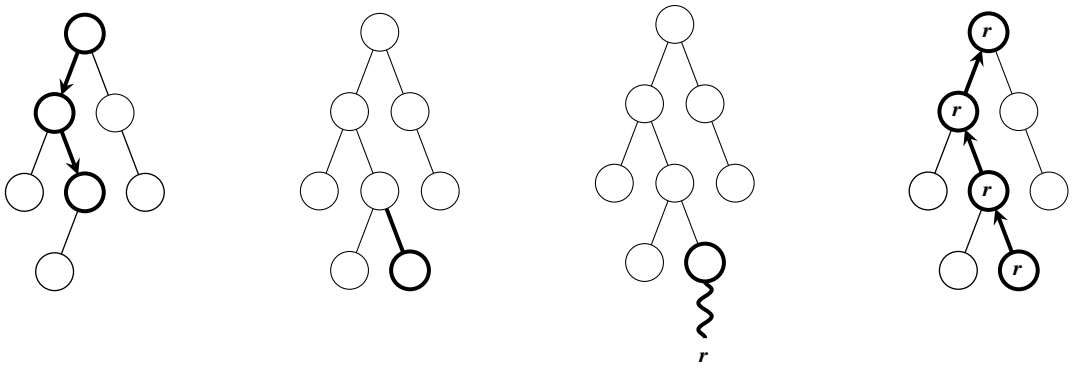
Monte Carlo tree search (MCTS) is a general approach to probabilistic planning that has garnered considerable attention after its noteworthy success in computer Go [72], and has demonstrated strong performance in planning competitions [46]. It is often applied in domains with a large branching factor where it may be difficult or impossible to explore the entire search tree. MCTS combines standard tree search with Monte Carlo rollouts, and uses the outcome of these rollouts to evaluate states. The core idea of MCTS is to asymmetrically grow the search tree by focusing on the most promising regions of the search space.

MCTS iteratively builds a search tree by cycling through four phases: selection, expansion, simulation, and backpropagation, depicted in Figure 2.4 [14].

1. Selection: Starting from the root node, a selection policy is applied to recursively select successor nodes, descending through the search tree until a node with unvisited children is reached.

2. Expansion: An unvisited node is added to the search tree.

3. Simulation: From the newly added node, a simulation policy is applied to simulate a trajectory



(a) Selection (b) Expansion (c) Simulation (d) Backpropagation

Figure 2.4: Phases of a Monte Carlo tree search rollout

of actions until a terminal node or the maximum rollout depth is reached. The outcome of this trajectory yields a return value (a cost or reward).

4. Backpropagation: The cost or reward is backpropagated through the search tree to update node statistics.

Of the many variants of MCTS (e.g., [31], [35], [55]), Upper Confidence Bound applied to Trees (UCT) is the most widely used in practice [47]. It uses a solution to the multi-armed bandit problem [4] that provides a principled balance between exploration and exploitation when selecting nodes to expand. During the selection phase, the action selected is the one which maximizes the UCB1 score

$$\text{UCB1}(Q, N, s, a) = Q(s, a) + C\sqrt{\frac{\log(N(s))}{N(s, a)}}. \tag{2.1}$$

Here, $Q(s, a)$ is the current expected utility for taking action a in state s to achieve the planning goal g. This represents the exploitation term, which favors actions that have historically led to high rewards. $N(s)$ is the number of times the state s has been visited, while $N(s, a)$ is the number of times a has been executed from state s. The second term of Equation 2.1 is the exploration

bonus, which encourages trying actions that have been selected less frequently. In practice, actions that have not yet been tried (i.e., $N(s,a) = 0$) are selected at least once before applying the UCB1 formula, ensuring that all actions are explored. The exploration constant $C > 0$ controls the exploration-exploitation trade-off. By always selecting the action that maximizes the UCB1 score, the algorithm naturally balances exploiting known good paths through the tree with exploring other less-certain but potentially more rewarding ones.

In planning applications, MCTS operates on a generative model of the probabilistic planning domain: given a state and action, a simulator samples a successor state according to $\Pr(s' \mid s,a)$. A rollout generates a trajectory whose cumulative return—defined according to the planning criterion (e.g., negative cumulative cost or utility)—is used as a sample to estimate the action's utility.

Note that this sophisticated UCB1 policy is applied only during the selection phase. The policy used to select actions during the simulation phase is typically much simpler and faster—often uniform random action selection—as its priority is to quickly generate an outcome from the leaves of the tree.

The values obtained from these simulations are backpropagated to update Q and N. As the number of rollouts increases, the $Q(s,a)$ values become more accurate approximations of the true expected utility. Kocsis and Szepesvári [47] showed that with $C = \sqrt{2}$ and utility values bounded in $[0,1]$, the probability of selecting a suboptimal action at the root of the tree converges to 0 as the number of rollouts grows to infinity. Therefore, with an unlimited rollout budget, the action selected at the root converges to an optimal action. Despite this strong theoretical guarantee, the main practical advantage of MCTS lies in its strong anytime performance: it can be stopped at any point to return the currently estimated best action, with the quality of that choice generally

improving with more computation time.

Despite its strong theoretical guarantees and anytime behavior, standard MCTS treats planning as a flat search problem and does not explicitly leverage structured goal decompositions. Classical planning research, however, has shown that intermediate subgoal structure can significantly accelerate search [43]. Incorporating such structure into probabilistic Monte Carlo planning is a central theme of the subsequent chapters.

Chapter 3: Landmarks for Probabilistic Planning[1]

Landmarks have proven to be a powerful tool in classical planning, serving as intermediate conditions that must be satisfied in any solution plan. In classical planning, landmarks have been used to focus search [43] and for heuristic guidance [65]. Ongoing work has developed landmarks for more sophisticated heuristics (see Section 3.1).

Despite their success in classical deterministic planning, landmarks have seldom been applied in probabilistic domains. To address the challenges of probabilistic planning, we introduce a novel formalism for probabilistic landmarks as a natural extension of the classical landmark first defined by Porteous et al. [60]. Building on this formalism, we adapt the UCT algorithm to utilize probabilistic landmarks as subgoals during rollouts, introducing a weighting parameter that balances between greedily achieving the next landmark and optimizing for the final goal. This adaptation forms the foundation of the Landmark-Assisted Monte Carlo Planning (LAMP) algorithm, which employs landmark-sensitive rollouts to learn a Q-function that prioritizes the achievement of key intermediate states. Through extensive evaluation, we demonstrate that LAMP significantly outperforms standard UCT in five of six probabilistic benchmark domains, highlighting the utility of

[1] This chapter is based on the following publication:
[18] D. H. Chan, M. Roberts, D. S. Nau, "Landmark-Assisted Monte Carlo Planning," in *Proceedings of the Twenty-Eighth European Conference on Artificial Intelligence (ECAI 2025)*, Bologna, Italy, October 20–25, 2025, pp. 4710–4717.

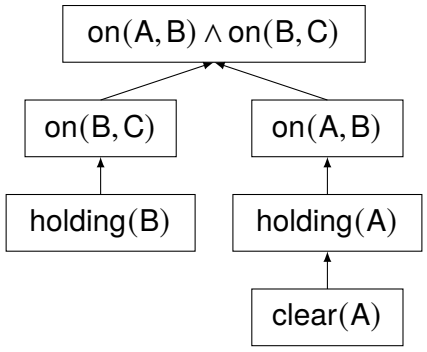


Figure 3.1: Landmark graph for the `blocksworld` problem in Figure 2.1. Each node of the landmark graph contains a condition which must be true at some point in every solution. The directed edges in the graph indicate the order in which each landmark is satisfied.

probabilistic landmarks. Moreover, our analysis reveals that the optimal balance between landmark and final goal achievement depends on the domain and problem.

3.1 Related Works

Landmarks are conditions that must be satisfied at some point in every solution plan. For example, in the `blocksworld` problem shown in Figure 2.1, all the landmarks in Figure 3.1 must be satisfied at some point in every solution. Traditionally, they have been used to reduce the search space of a planning task by providing intermediate states that guide the planning algorithm [43]. By treating landmarks as subgoals, planners can decompose complex planning tasks into smaller, more manageable subproblems. These subproblems can then be solved independently, and their solutions can be combined to obtain a complete solution to the original planning problem.

Porteous et al. [60] first introduced landmarks for propositional planning. They showed that given a planning problem, some landmarks could be computed efficiently, even though deciding whether an arbitrary propositional formula is a landmark is PSPACE-complete in the worst case, equivalent to the complexity of deciding plan existence. They proposed an algorithm, LM^{RPG}, that

could efficiently extract landmarks and orderings using a delete-relaxed planning graph (RPG)—a simplified representation of the planning problem where the negative (or "delete") effects of actions are ignored. Using the resulting landmark graph, their proposed method restricted the action space by disallowing actions that would achieve landmarks out of order. However, their approach saw limited performance improvements over the base planner. Hoffmann et al. [43] later developed an alternative approach that used landmarks to decompose the planning problem by iteratively searching for a plan to the nearest landmark with no predecessors, rather than searching for a plan to the goal. This approach demonstrated a more substantial speedup, but sometimes resulted in longer plans; it could even potentially fail on solvable tasks, even when using a complete base planning algorithm. Vernhes et al. [81] later revisited this problem-splitting approach with a search framework based on landmark orderings for a given landmark graph.

Another major application of landmarks is in the creation of heuristics to guide search-based planners. The core idea is that the number of landmarks yet to be achieved can serve as an estimate of the remaining distance to the goal. Building on early work by Zhu and Givan [88], a key advancement came from Richter et al. [64], who proposed the LM^{RHW} landmark extraction algorithm. This technique uses domain transition graphs (DTGs)—which model how the values of individual state variables change—to generate landmarks. This technique was later used within the highly successful LAMA planner [65].

This landmark heuristic framework has proven highly versatile. Others have tailored it for different planning objectives, developing admissible heuristics for optimal planning where finding the shortest path is required [41], [59], as well as stronger inadmissible heuristics for satisficing planning, where finding a good solution quickly is the priority [15]. The core concept has also

been extended beyond classical planning into more expressive formalisms, including lifted planning, which operates directly on symbolic, first-order representations [82], and numeric planning, which incorporates numeric state variables [66].

However, in nondeterministic planning, landmarks have only been used in limited settings. Speck et al. [75] used a type of landmark analysis to compute a set of necessary observations used to minimize the number of sensors on an agent in a partially-observable nondeterministic domain. Sreedharan et al. [77] define and generate policy landmarks for stochastic shortest path problems (SSPs) to identify subgoals for a given policy. These subgoals then provide high-level explanations of state-action policies, and can be used by end users for explanatory queries. Building on this, Sreedharan et al. [76] extend causal links and action justification concepts to fully observable nondeterministic (FOND) planning, allowing agents to explain why specific actions are required in stochastic policies.

Within reinforcement learning, the options framework [79] is an example of a hierarchical approach that models temporally extended actions or skills, allowing an agent to solve smaller subproblems and compose the resulting policies. Several works aim to learn options that navigate to "bottleneck" states that occur frequently on solution trajectories [51, 52, 73, 74, 78], or prototypical states of well-connected regions of the state space [63]. Then, these options may be used alongside primitive actions in learning methods such as Q-learning. Landmarks have also been used to guide reinforcement learning agents through POMDPs by providing guiding rewards based on how valuable it is to visit each landmark in the task [28].

3.2 Preliminaries

To formalize our contributions, we begin by establishing some notation. Recall from Section 2.1 that, given a propositional language $\mathcal{L} = (\mathcal{X}, \mathcal{F})$, a classical planning domain $\mathcal{D}_C$ is a tuple $(\mathcal{L}, A_C, \gamma_C)$, where $S = 2^{\mathcal{X}}$ and A_C are finite sets of states and actions, respectively; and $\gamma_C : S \times A_C \to S$ is a deterministic state-transition function. An action $a \in A_C$ is applicable in state $s \in S$ if $\gamma_C(s, a)$ is defined; we write Applicable(s) to denote the set of all applicable actions in s. A *plan* is a sequence of actions $\pi_C = \langle a_1, \ldots, a_n \rangle$. We write $\gamma_C(s, \pi_C)$ to denote the state produced by starting at s and applying the actions in π_C in order, if all of them are applicable. Let $\mathcal{P}_C = (\mathcal{D}_C, s_0, g)$ be a classical planning problem, where $s_0 \in S$ is the initial state and g is the goal condition. A solution to $\mathcal{P}_C$ is a plan π_C such that $\gamma_C(s_0, \pi_C) \models g$.

Adopting definitions of landmarks from Richter and Westphal [65], let $\mathcal{P}_C = (\mathcal{D}_C, s_0, g)$ be a classical planning problem and let $\pi_C = \langle a_1, \ldots, a_n \rangle$ be a plan. A condition φ is:

- *true at time i* in π_C if $\gamma_C(s_0, \langle a_1, \ldots, a_i \rangle) \models \varphi$.
- *added at time i* in π_C if φ is true at time i but not at time $i - 1$.
- *first added at time i* in π_C if φ is true at time i, but not at any time $j < i$.

Then, a condition φ is a *landmark* for $\mathcal{P}_C$ if for all plans π_C such that $\gamma_C(s_0, \pi_C) \models g$, φ is true at some time in π_C. Let φ_1 and φ_2 be conditions. In problem $\mathcal{P}_C$, there is a:

- *natural ordering* $\varphi_1 \to \varphi_2$ if for all i, in every plan where φ_2 is true at time i, φ_1 is true at some time $j < i$.

- *necessary ordering* $\varphi_1 \rightarrow_{\mathrm{n}} \varphi_2$ if for all i, in every plan where φ_2 is added at time i, φ_1 is true at time $i-1$.

- *greedy-necessary ordering* $\varphi_1 \rightarrow_{\mathrm{gn}} \varphi_2$ if for all i, in every plan where φ_2 is first added at time i, φ_1 is true at time $i-1$.

They also defined reasonable orderings [65], but such orderings are not mandatory. These are not used in LAMP, so we omit them here.

A probabilistic planning domain $\mathcal{D}$ is a tuple $(\mathcal{L}, A, \gamma, \Pr)$, where $S = 2^{\mathcal{X}}$ and A are finite sets of states and actions, respectively; $\gamma : S \times A \rightarrow 2^S$ is a nondeterministic state-transition function; and $\Pr(s' \mid s, a)$ is a distribution over $\gamma(s, a)$ giving the probability of reaching state s' when executing action a in state s. An action $a \in A$ is applicable in state $s \in S$ if $\gamma(s, a) \neq \varnothing$; we write $\mathrm{Applicable}(s)$ to denote the set of all applicable actions. A *policy* is a partial function $\pi : S \rightarrow A$ with domain $\mathrm{Dom}(\pi) \subseteq S$ such that for all $s \in \mathrm{Dom}(\pi)$, $\pi(s) \in \mathrm{Applicable}(s)$. The policy π is total if $\mathrm{Dom}(\pi) = S$. We will assume all actions have unit cost.

Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem, where s_0 is the initial state and g is the goal condition. Solutions to $\mathcal{P}$ take the form of a policy. For a policy π and initial state s_0, a *history* σ is a finite sequence of states $\langle s_0, s_1, \ldots, s_h \rangle$, starting in s_0, such that for all $i \in \{1, \ldots, h\}$, $s_i \in \gamma(s_{i-1}, \pi(s_{i-1}))$ and for all i, $((s_i \models g) \vee (s_i \notin \mathrm{Dom}(\pi))) \iff i = h$. Let $|\sigma|$ denote the length of σ, and σ_{-1} denote the final state in σ. The probability of a history σ of a policy π from state s_0 is defined to be $\Pr(\sigma \mid \pi, s_0) = \prod_{i=1}^{|\sigma|} \Pr(s_i \mid s_{i-1}, \pi(s_{i-1}))$. We write $H(s_0, \pi, g)$ to denote the subset of all histories of π from s_0 that end in a state that satisfies g. A policy π is *safe* for $\mathcal{P}$ if $\sum_{\sigma \in H(s_0, \pi, g)} \Pr(\sigma \mid \pi, s_0) = 1$.

3.3 Extension to Probabilistic Planning

3.3.1 Formalism

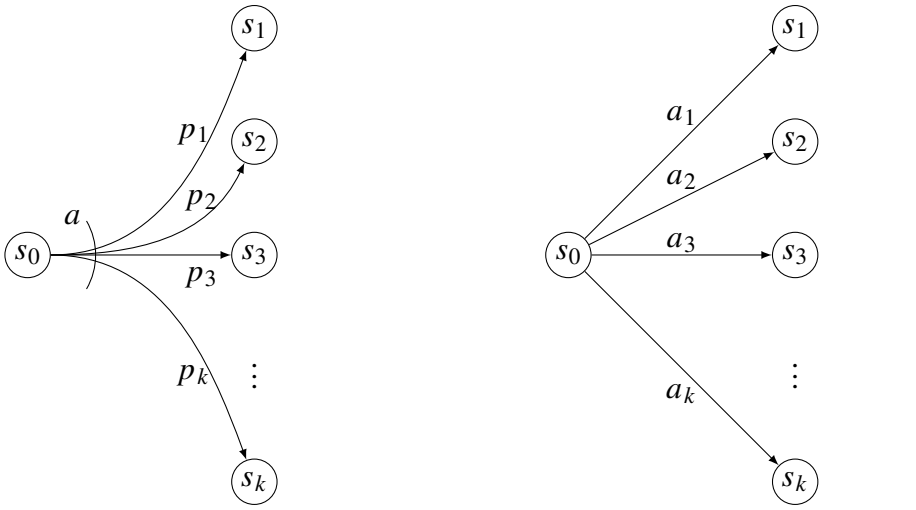
We introduce a new definition of landmarks and landmark orderings for probabilistic planning domains. Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem. For a policy π and a history $\sigma = \langle s_0, \ldots, s_h \rangle \in H(s_0, \pi, g)$, a condition φ is:

- *true at time i* in σ if $s_i \models \varphi$.
- *added at time i* in σ if $s_i \models \varphi$ and $s_{i-1} \not\models \varphi$.
- *first added at time i* in σ if $s_i \models \varphi$ and $s_j \not\models \varphi$ for all $j < i$.

Then, a condition φ is a *landmark* for $\mathcal{P}$ if for all policies π and all histories $\sigma \in H(s_0, \pi, g)$, φ is true at some time in σ. Let φ_1 and φ_2 be conditions. In problem $\mathcal{P}$, there is a:

- *natural ordering* $\varphi_1 \rightarrow \varphi_2$ if for all i, for every policy π and every history $\sigma \in H(s_0, \pi, g)$ where φ_2 is true at time i, φ_1 is true at some time $j < i$.
- *necessary ordering* $\varphi_1 \rightarrow_{\mathrm{n}} \varphi_2$ if for all i, for every policy π and every history $\sigma \in H(s_0, \pi, g)$ where φ_2 is added at time i, φ_1 is true at time $i - 1$.
- *greedy-necessary ordering* $\varphi_1 \rightarrow_{\mathrm{gn}} \varphi_2$ if for all i, for every policy π and every history $\sigma \in H(s_0, \pi, g)$ where φ_2 is first added at time i, φ_1 is true at time $i - 1$.

We will construct probabilistic landmarks for $\mathcal{P} = (\mathcal{D}, s_0, g)$ using the *all-outcomes determinization* of $\mathcal{D} = (\mathcal{L}, A, \gamma, \Pr)$, which is a classical domain, denoted $\mathcal{D}_D = (S, A_D, \gamma_D)$. In this determinization, each action in $\mathcal{D}$ is replaced by a set of deterministic actions, one for each possible outcome of the original action (see Figure 3.2). Formally, let $s \in S$ and $a \in \text{Applicable}(s)$, and suppose



(a) Probabilistic action (b) Determinized actions

Figure 3.2: The all-outcomes determinization transforms a probabilistic action with multiple possible outcomes into multiple deterministic actions, one for each possible outcome of the probabilistic action.

$\gamma(s,a) = \{o_1, \ldots, o_k\}$. We transform a into a set of deterministic actions $\det(s,a) = \{d_1, \ldots, d_k\}$ corresponding to each outcome in $\gamma(s,a)$, where $o_i = \gamma_D(s,d_i)$ for all $i \in \{1,\ldots,k\}$. Then $A_D = \bigcup_{s\in S, a\in \text{Applicable}(s)} \det(s,a)$. Landmarks in $\mathcal{D}_D$ carry over to $\mathcal{D}$ as follows:

Lemma 1. Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem, where $\mathcal{D} = (\mathcal{L}, A, \gamma, \Pr)$. Let $\mathcal{P}_D = (\mathcal{D}_D, s_0, g)$ be the classical planning problem where $\mathcal{D}_D = (S, A_D, \gamma_D)$ is the all-outcomes determinization of $\mathcal{D}$. Let π be a policy for $\mathcal{P}$. For every history $\sigma = \langle s_0, \ldots, s_h\rangle \in H(s_0, \pi, g)$, there exists a plan $\pi_D = \langle a_1, \ldots, a_h\rangle$ for $\mathcal{P}_D$ such that for all $i \in \{1,\ldots,h\}$, $\gamma_D(s_0, \langle a_1, \ldots, a_i\rangle) = s_i$; that is, π_D reaches the same sequence of states as σ.

Proof. Let $\sigma = \langle s_0, \ldots, s_h\rangle \in H(s_0, \pi, g)$ be a history. For all $i \in \{1,\ldots,h\}$, $s_i \in \gamma(s_{i-1}, \pi(s_{i-1}))$, thus there exists $d_i \in \det(s_{i-1}, \pi(s_{i-1})) \subseteq A_D$ where $s_i = \gamma_D(s_{i-1}, d_i)$. By induction, it follows that for all $i \in \{1,\ldots,h\}$, $\gamma_D(s_0, \langle d_1, \ldots, d_i\rangle) = s_i$, so $\pi_D = \langle d_1, \ldots, d_h\rangle$ reaches the same sequence of states as σ. □

Theorem 2. *Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem, where $\mathcal{D} = (\mathcal{L}, A, \gamma, \Pr)$. Let $\mathcal{P}_D = (\mathcal{D}_D, s_0, g)$ be the classical planning problem where $\mathcal{D}_D$ is the all-outcomes determinization of $\mathcal{D}$. If φ is a landmark for $\mathcal{P}_D$, then φ is a landmark for $\mathcal{P}$.*

Proof. Let π be a policy for $\mathcal{P}$, and let $\sigma \in H(s_0, \pi, g)$. By Lemma 1, there exists a plan π_D that reaches the same sequence of states as σ. Suppose φ is a landmark for $\mathcal{P}_D$. Then φ must be true at some time in π_D, and so it follows that φ is also true at some time in σ. □

Moreover, landmark orderings also carry over from the all-outcomes determinization to the probabilistic domain.

Theorem 3. *Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem, where $\mathcal{D} = (\mathcal{L}, A, \gamma, \Pr)$. Let $\mathcal{P}_D = (\mathcal{D}_D, s_0, g)$ be the classical planning problem where $\mathcal{D}_D = (S, A_D, \gamma_D)$ is the all-outcomes determinization of $\mathcal{D}$. Let φ_1 and φ_2 be conditions.*

(1) If $\varphi_1 \rightarrow \varphi_2$ in $\mathcal{P}_D$, then $\varphi_1 \rightarrow \varphi_2$ in $\mathcal{P}$.

(2) If $\varphi_1 \rightarrow_{\mathrm{n}} \varphi_2$ in $\mathcal{P}_D$, then $\varphi_1 \rightarrow_{\mathrm{n}} \varphi_2$ in $\mathcal{P}$.

(3) If $\varphi_1 \rightarrow_{\mathrm{gn}} \varphi_2$ in $\mathcal{P}_D$, then $\varphi_1 \rightarrow_{\mathrm{gn}} \varphi_2$ in $\mathcal{P}$.

Proof. Proofs for (1), (2), and (3) are similar. To prove (1) (resp. (2); (3)), suppose φ_1 and φ_2 are landmarks in $\mathcal{P}_D$, with $\varphi_1 \rightarrow \varphi_2$ (resp. $\varphi_1 \rightarrow_{\mathrm{n}} \varphi_2$; $\varphi_1 \rightarrow_{\mathrm{gn}} \varphi_2$). Let π be a policy for $\mathcal{P}$, let $\sigma \in H(s_0, \pi, g)$, and suppose φ_2 is true (resp. added; first added) at time i in σ. By Lemma 1, there exists a plan π_D that reaches the same sequence of states as σ, so φ_2 is also true (resp. added; first added) at time i in π_D. Then, φ_1 is true at some time $j < i$ (resp. time $i-1$; time $i-1$) in π_D. Thus, φ_1 is also true at time j (resp. $i-1$; $i-1$) in σ. □

Therefore, to generate landmarks for a probabilistic planning problem $\mathcal{P} = (\mathcal{D}, s_0, g)$, we can use existing classical landmark extraction algorithms, like LM^{RHW} [64], to generate classical landmarks for the all-outcomes determinized planning problem $\mathcal{P}_D = (\mathcal{D}_D, s_0, g)$, and directly use those landmarks and their orderings in $\mathcal{P}$.

3.3.2 Landmarks and Subgoals: Expected Benefit

Before we introduce LAMP, we will discuss the expected benefit that landmarks might provide while also considering some subtleties of using landmarks to solve MDPs that make this a nontrivial task. In classical and probabilistic planning, subgoals can be used to decompose large planning problems into smaller, easier-to-solve subproblems. A domain-independent approach to generating subgoals is to compute landmarks and use them as subgoals. We can decompose a planning task into subtasks, each with the goal of achieving one landmark, then combine their solutions, a procedure inspired by Hoffmann et al. [43]. However, this straightforward sequential approach to achieving landmarks is not guaranteed to succeed, even if a solution exists for the original problem. A key limitation is that achieving one landmark can lead to a state from which future landmarks, or the final goal itself, are unreachable. We will elaborate on this point later.

Let $\mathcal{P}_C = (\mathcal{D}_C, s_0, g)$ be a classical planning problem, let $\langle \varphi_1, \ldots, \varphi_k \rangle$ be a sequence of landmarks, and suppose $\varphi_k \equiv g$. For each $i \in \{1, \ldots, k\}$, suppose there exists a solution plan π_C^i for the problem $(\mathcal{D}_C, s_{i-1}, \varphi_i)$, where $s_{i-1} = \gamma_C(s_0, \pi_C^1 \circ \cdots \circ \pi_C^{i-1})$ for $i > 1$. Then, a solution to $\mathcal{P}_C$ can be obtained by concatenating $\pi_C^1 \circ \cdots \circ \pi_C^k$. Note that this solution depends on the choice of sequence $\langle \varphi_1, \ldots, \varphi_k \rangle$ as well as the plans π_C^i.

A similar approach can be applied to probabilistic planning problems. To illustrate, let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem and let $\langle \varphi_1, \ldots, \varphi_k \rangle$ be a sequence of landmarks

Algorithm 3.1 (SEQUENTIAL-PLAN) A planning algorithm for a sequence of landmarks.

```
1: procedure SEQUENTIAL-PLAN(𝒟, s_0, ⟨φ_1, ..., φ_k⟩)
2:     s ← s_0
3:     for i = 1, ..., k do
4:         π_i ← policy for (𝒟, s, φ_i)
5:         while s ⊭ φ_i do
6:             s ← APPLY(s, π_i(s))
```

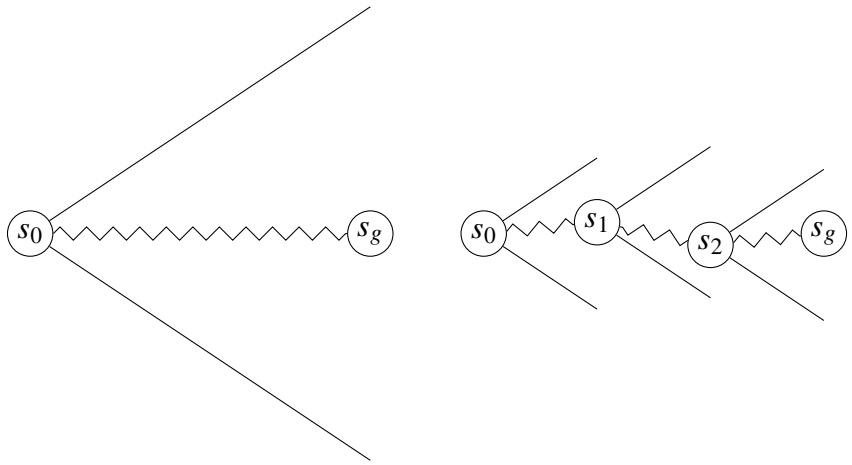


Figure 3.3: Suppose φ_1 and φ_2 are landmarks such that $s_1 \models \varphi_1$ and $s_2 \models \varphi_2$. With a sequence of well-chosen landmarks, a sequential planning approach can yield an exponential speed-up.

where $\varphi_k \equiv g$. For all $i \in \{1, \ldots, k\}$, suppose there exists a safe policy π_i which achieves φ_i from all states $s_{i-1} \models \varphi_{i-1}$ reachable by running π_{i-1}, or from s_0 if $i = 1$. We can obtain a solution to $\mathcal{P}$ by running policies $\pi_1, \ldots, \pi_k$ in sequence, as shown in Algorithm 3.1. This procedure again depends on the choice of sequence $\langle \varphi_1, \ldots, \varphi_k \rangle$ and policies π_i.

In the best case, the landmark-based approach could yield up to an exponential speed-up in plan time. Consider an unbounded search space with branching factor b and a goal g at depth d from the initial state s_0. A breadth-first planner would take $O(b^d)$ time to reach the goal. However, suppose there exists a sequence of landmarks $\langle \varphi_1, \ldots, \varphi_k \rangle$ where $\varphi_k \equiv g$, φ_1 is at depth d/k from s_0, and for all $i \in \{1, \ldots, k-1\}$, if $s_i \models \varphi_i$, then φ_{i+1} is at depth d/k from s_i. In this best case, a breadth-first

planner takes $O(b^{d/k})$ to achieve each landmark. With k landmarks, this reduces the total planning time to $O(kb^{d/k})$, an exponential speed-up. This is depicted in Figure 3.3.

In general, for a problem $\mathcal{P} = (\mathcal{D}, s_0, g)$ and an unordered collection of landmarks Φ, there is no unique "optimal" sequencing of Φ that minimizes expected cost. As illustrated in Figure 3.4, in the classical setting, the best order in which to achieve a set of landmarks may depend on the subplans used for each landmark. Moreover, in the probabilistic setting, the optimal sequence for attaining landmarks can even depend on the probabilistic outcomes of actions.

(a) classical problem (b) probabilistic problem

Figure 3.4: There is no optimal ordering of landmarks. In both problems, suppose φ_1 and φ_2 are landmarks such that $s_2, s_3 \models \varphi_1$ and $s_1, s_4 \models \varphi_2$. Without first fixing policies π_1 and π_2 which achieve φ_1 and φ_2, respectively, it is unclear whether SEQUENTIAL-PLAN$(\mathcal{D}, s_0, \langle \varphi_1, \varphi_2 \rangle)$ or SEQUENTIAL-PLAN$(\mathcal{D}, s_0, \langle \varphi_2, \varphi_1 \rangle)$ would be better.

However, by first fixing a collection of total, safe-cyclic policies for a collection of landmarks, we can show that there does exist an optimal ordering of landmarks with respect to those policies. Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem, let Φ be a collection of landmarks, and let $\pi_1, \ldots, \pi_k$ be total, safe-cyclic policies where π_i achieves $\varphi_i \in \Phi$. Then, there exists an optimal ordering that minimizes the expected cost of executing each of these policies in sequence. We define the cost of achieving the sequence of landmarks $\langle \varphi_1, \ldots, \varphi_k \rangle$ from state s as:

$$\text{cost}(s, \langle \varphi_i, \ldots, \varphi_k \rangle)$$

$$= \begin{cases} \sum_{\sigma \in H(s,\pi_i,\varphi_i)} \Pr(\sigma \mid \pi_i, s)|\sigma|, & \text{if } i = k \\ \sum_{\sigma \in H(s,\pi_i,\varphi_i)} \Pr(\sigma \mid \pi_i, s)\big(|\sigma| + \text{cost}(\sigma_{-1}, \langle \varphi_{i+1}, \ldots, \varphi_k \rangle)\big), & \text{otherwise} \end{cases} \tag{3.1}$$

where σ_{-1} is the final state in history σ and $|\sigma|$ is the length of σ. Since the set of permutations of Φ is finite and cost is bounded, there exists a minimum-cost sequencing of Φ.

However, even the existence of policies for each landmark is not guaranteed. This limitation underpins the incompleteness of this sequential landmark achievement approach: even when a safe policy for the original planning problem exists, this approach might fail to reach the top-level goal g. These failures can occur in two situations: the current, chosen landmark is unreachable, or a deadlock state is encountered, i.e., a state from which the final goal is unreachable [43].

(a) softlocking problem (b) deadlocking problem

Figure 3.5: Achieving some landmarks may lead to dead ends. In both problems, consider the execution of SEQUENTIAL-PLAN$(\mathcal{D}, s_0, \langle \varphi_1, \varphi_2 \rangle)$. In (a), if φ_1 is achieved by reaching state s_2, it is impossible to achieve φ_2 after φ_1, which is what the problem requires. In (b), if φ_1 is achieved by reaching state s_1, it is impossible to reach the final goal.

Consider the execution of SEQUENTIAL-PLAN$(\mathcal{D}, s_0, \langle \varphi_1, \varphi_2 \rangle)$. In Figure 3.5(a), suppose policy π_1 achieves φ_1 by reaching state s_2. Upon completion of φ_1, the next landmark φ_2 is impossible to achieve, and thus the SEQUENTIAL-PLAN procedure fails on Line 4. In Figure 3.5(b), if

Algorithm 3.2 (LANDMARK-PLAN) A general procedure for landmark-guided probabilistic planning.

procedure LANDMARK-PLAN($\mathcal{P}$)
  $\Phi \leftarrow$ LANDMARKGRAPH($\mathcal{P}$) $\cup \{g\}$
  $s \leftarrow s_0$
  $\varphi \leftarrow$ NIL
  while $\Phi \neq \varnothing$ and $s \not\models g$ **do**
    if $\varphi =$ NIL or $s \models \varphi$ **then** ▷ select landmark
      $\Phi \leftarrow \Phi \backslash \{\varphi\}$
      select $\varphi \in$ leaves(Φ)
    else ▷ select action
      select $a \in$ Applicable(s) to achieve φ
      $s \leftarrow$ APPLY(s, a)

SEQUENTIAL-PLAN achieves landmark φ_1 by reaching state s_1, a deadlock state is reached from which the final goal is no longer achievable.

These failures stem from the approach's inherent myopia: by focusing on achieving landmarks, the planner can be led to states that hinder or prevent further progress towards future landmarks or the final goal. In the classical setting, Hoffmann et al. [43] proposed a "safety net" to address the issue of unreachable landmarks—in the event that an unreachable landmark is selected, the planner reverts to a base planner that seeks to achieve the final goal, disregarding the landmark graph. In the probabilistic setting, we mitigate these failure modes with an approach that balances the pursuit of landmarks with the pursuit of the final goal by tuning a greediness parameter.

3.4 Landmark-Assisted Monte Carlo Planning (LAMP)

The second core contribution of this chapter is the implementation of an approach that uses probabilistic landmarks to guide planning. Intuitively, the idea is straightforward—Algorithm 3.2 shows a procedure, called LANDMARK-PLAN, that first computes a collection of landmarks Φ (Line 2) and then iteratively selects a landmark (Lines 6–8) and actions to achieve that landmark (Lines 9–11). The algorithm's behavior depends largely on how selection on Lines 8 and 10 is implemented.

Algorithm 3.3 shows LAMP, the main algorithm of this chapter, which is a specific implementation of LANDMARK-PLAN based on Monte Carlo tree search. Like LANDMARK-PLAN, it has stages for selecting landmarks (Lines 9–11) and actions (Lines 12–14). LAMP learns Q-functions (Lines 7–8) to predict the best landmark (Q_{LM}), the best action for the current landmark (Q_{φ}), and the best action for the final goal (Q_g). Crucially, LAMP uses a greediness parameter θ to achieve a balance between greedy actions to achieve the current landmark with actions that advance toward the top-level goal (Line 13). If a dead end is encountered, the algorithm fails.

Section 3.5 will demonstrate that LAMP performs well in several benchmark problems and that the best θ is problem dependent; the results present strong evidence that landmarks can be helpful for solving MDPs. More generally, landmarks can be incorporated into MDP algorithms in many ways; LAMP is one instance in a family based on LANDMARK-PLAN which uses UCT [47], a simple, well-established planning algorithm that allows us to easily modulate the amount of work done to learn a policy and examine the effect of using landmarks. Future work should explore other variations of LANDMARK-PLAN, including non-sampling-based approaches.

In the following sections, we describe the key components of LAMP in determining: how to choose a landmark, how to choose an action, and how to learn the Q-functions.

Algorithm 3.3 (LAMP) A landmark-assisted Monte Carlo planning algorithm for a planning problem $\mathcal{P} = (\mathcal{D}, s_0, g)$, where n_{ro} is the number of rollouts to perform, *budget* is a cost limit, *depth* is the maximum rollout depth, and θ is the greediness parameter.

procedure LAMP($\mathcal{P}, n_{ro}, \theta$)
  $\Phi \leftarrow$ LandmarkGraph($\mathcal{P}$) $\cup \{g\}$
  $\varphi \leftarrow$ NIL
  $s \leftarrow s_0$
  $cost \leftarrow 0$
  while $\Phi \neq \varnothing$ and $cost < budget$ **do**
    for n_{ro} times **do** ▷ learn Q_{LM}, Q_φ, Q_g
      LAMP-Rollout($s, \varphi, \Phi, depth, \theta, cost$)
    if $\varphi =$ NIL or $s \not\models \varphi$ **then** ▷ select landmark
      $\Phi \leftarrow \Phi \backslash \{\varphi\}$
      $\varphi \leftarrow \underset{\varphi \in \text{leaves}(\Phi)}{\arg\max} \; (Q_{LM}(\Phi, \varphi))$
    else ▷ select action
      $a \leftarrow \underset{a \in \text{Applicable}(s)}{\arg\max} \; (\theta Q_\varphi(s,a) + (1-\theta) Q_g(s,a))$
      $s \leftarrow$ apply(s, a)
    $cost \leftarrow cost + 1$

3.4.1 Choosing Landmarks in LAMP

To learn an ordering of the landmarks, we frame the selection of landmarks as a planning problem. Let $\mathcal{P} = (\mathcal{D}, s_0, g)$ be a probabilistic planning problem and let Φ be a collection of landmarks for $\mathcal{P}$. Let $\mathcal{D}_\Phi$ be a planning domain with state space $S_\Phi = 2^\Phi$, action space $A_\Phi = \Phi$, and state-transition function $\gamma_\Phi : (s, a) \mapsto s \backslash \{a\}$ if $a \in s$. Actions in this domain are deterministic, meaning $\Pr(s' \mid s, a) = \mathbf{1}_{\gamma_\Phi(s,a)}(s')$. However, these actions do not have unit cost. Instead, the cost of executing $a \in A_\Phi$ is determined by the cost of running policy π_a in $\mathcal{D}$ to achieve landmark a. This cost depends not only on the action $a \in A_\Phi$ and the current state $s \in S_\Phi$, but also on the state in $\mathcal{D}$ from which π_a is initiated, meaning that the cost also depends on the history of previously executed actions.

To further restrict the set of applicable actions in $\mathcal{D}_\Phi$, we can impose a strict partial order $\prec$ on Φ using landmark orderings. Then, for any $\Phi' \subseteq \Phi$, we can view Φ' as a directed acyclic graph and define

$$\text{leaves}(\Phi') = \{\varphi \in \Phi' \mid \varphi \text{ has no } \prec\text{-predecessors}\}. \tag{3.2}$$

We can then restrict $\text{Applicable}(s) = \text{leaves}(s)$. We will also assume $g \in \Phi$, with $\varphi \prec g$ for all $\varphi \in \Phi \backslash \{g\}$ to ensure the top-level goal g is achieved. Then, landmark selection can be viewed as a planning problem $\mathcal{P}_\Phi = (\mathcal{D}_\Phi, \Phi, g_\Phi)$, where $s \models g_\Phi$ iff $s = \varnothing$.

LAMP's landmark selection bears some similarity to the options framework in hierarchical reinforcement learning [79]. That is, $\mathcal{D}_\Phi$ is semi-Markov. To see why, consider an option $(\mathcal{I}, \dot{\pi}, \beta)$ which consists of an initiation set $\mathcal{I} \subseteq S$, a policy $\dot{\pi} : S \times A \to [0, 1]$, and a termination condition $\beta : S \to [0, 1]$. A total, safe policy π_a for landmark $a \in A_\Phi$ can be viewed as an option $(\mathcal{I}, \dot{\pi}, \beta)_a$ with initiation set $\mathcal{I} = S$, policy $\dot{\pi} = \mathbf{1}_{[\pi_a(s')=a']}(s', a')$, and termination condition $\beta = \mathbf{1}_{[s' \models a]}(s')$. Because

options are modeled as semi-Markov decision processes [79], we know that $\mathcal{D}_\Phi$ is semi-Markov, meaning that landmarks offer a similar benefit to options in providing temporal abstractions to decompose a problem.

3.4.2 Choosing Actions in LAMP

To solve $\mathcal{P}$ using the approach outlined in Algorithm 3.2, we must solve two planning problems concurrently: the landmark selection problem in $\mathcal{D}_\Phi$ (Algorithm 3.3 Lines 9–11) and the action selection problem in $\mathcal{D}$ (Algorithm 3.3 Lines 12–14). For our implementation of LAMP, we use UCT for both landmark selection and action selection, but we note that other techniques can be applied.

A greedy approach is one in which selection on Line 13 is done solely with respect to the current landmark φ, without regard for the top-level goal g. While this reduces the complexity of the planning task, it may compromise optimality. We can see this demonstrated in the example in Figure 3.6. Suppose we want to achieve landmark φ with top-level goal g. Assuming actions are deterministic, we can achieve φ by executing either $\langle a_1, a_2 \rangle$ or $\langle a_4 \rangle$. Although $\langle a_1, a_2 \rangle$ leads to a shorter solution for the top-level goal, the greedy planner favors $\langle a_4 \rangle$ because it achieves the landmark φ more efficiently.

To address this, we adopt a hybrid approach in LAMP that balances rewards for both φ and g. With a greediness parameter $\theta \in [0, 1]$, suppose Q_g and Q_φ are action value functions with respect to g and φ, respectively. Rather than using the greedy approach $\arg\max_{a \in \text{Applicable}(s)} Q_\varphi(s, a)$, the balanced approach uses

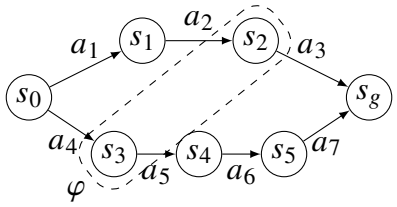


Figure 3.6: Greedy landmark achievement may lead to worse solutions. Suppose φ is a landmark such that $s_2 \models \varphi$ and $s_3 \models \varphi$. Starting at s_0, a greedy algorithm may achieve φ by performing action a_4, leading to a worse solution.

$$\underset{a \in \text{Applicable}(s)}{\arg\max} \big(\theta Q_\varphi(s,a) + (1-\theta) Q_g(s,a)\big) \tag{3.3}$$

(cf. Algorithm 3.3 Line 13). In Figure 3.6, suppose $Q_g(s_0, a_1) = \frac{1}{4}$, $Q_g(s_0, a_4) = \frac{1}{8}$, $Q_\varphi(s_0, a_1) = \frac{1}{2}$, $Q_\varphi(s_0, a_4) = 1$. So $\theta > \frac{1}{5}$ selects action a_4, while $\theta < \frac{1}{5}$ selects action a_1. Random selection is used to break ties.

3.4.3 Learning Q-Functions in LAMP

To learn these Q-values, LAMP uses an approach based on UCT (see Section 2.3). UCT planning traditionally uses reward signals from rollouts to iteratively update an action value function Q, where $Q(s,a)$ is an approximation of the expected reward of executing action a in state s. To tailor UCT for our setting of goal-directed MDPs, we use GUBS (Goals with Utility-Based Semantics), a criterion for solving SSPs with dead-ends which provides a principled trade-off between maximizing the probability of reaching the goal and minimizing expected cost with a utility-based model [33], [34]. The utility function U captures a smooth trade-off between goal achievement and cost, where the utility of a history σ is

$$U(\sigma) = \mathit{util}(|\sigma|) + K_g \mathbf{1}_{[\sigma_{-1} \models g]}, \tag{3.4}$$

Algorithm 3.4 (LAMP-ROLLOUT) The Monte Carlo rollout procedure for LAMP. s is the current state, φ is the current landmark, Φ is the current landmark graph, d is the remaining rollout depth, θ is the greediness parameter, c is the cumulative cost, u is a utility function over cost, and K_g is the goal-utility constant. Q_{LM}, N_{LM}, Q_φ, N_φ, Q_g, and N_g are maps with default value 0. ROLLOUT returns a tuple $(\lambda_\varphi, \beta_\varphi, \lambda_g, \beta_g)$, where λ_φ and λ_g are rollout costs for φ and g, respectively, and β_φ and β_g are booleans indicating the achievement of φ and g, respectively. The standard UCT rollout procedure is shown in gray and modifications or additions are emphasized in black.

procedure LAMP-ROLLOUT($s, \varphi, \Phi, d, \theta, c$)
  if $\Phi = \varnothing$ **then**
    return $(0, \text{true}, 0, \text{true})$
  if $\varphi = \text{NIL}$ or $s \models \varphi$ **then**
    $\varphi' \leftarrow \underset{\varphi' \in \text{leaves}(\Phi) \setminus \{\varphi\}}{\arg\max}\ \text{UCB1}(Q_{LM}, N_{LM}, \Phi, \varphi')$
    $(\lambda_\varphi, \beta_\varphi, \lambda_g, \beta_g) \leftarrow$ LAMP-ROLLOUT$(s, \varphi', \Phi \setminus \{\varphi\}, d, \theta, c)$
    UCB-UPDATE$(Q_{LM}, N_{LM}, \Phi, \varphi, \lambda_g + c, \beta_g)$
    return $(0, \text{true}, \lambda_g, \beta_g)$
  if $d = 0$ or Applicable$(s) = \varnothing$ **then**
    return $(d, \text{false}, d, \text{false})$
  $a \leftarrow \underset{a \in \text{Applicable}(s)}{\arg\max}\ (\theta \text{UCB1}(Q_\varphi, N_\varphi, s, a) + (1-\theta)\text{UCB1}(Q_g, N_g, s, a))$
  $s' \leftarrow$ SIMULATE(s, a)
  $(\lambda_\varphi, \beta_\varphi, \lambda_g, \beta_g) \leftarrow$ LAMP-ROLLOUT$(s', \varphi, \Phi, d-1, \theta, c+1)$
  $\lambda_\varphi \leftarrow \lambda_\varphi + 1$
  $\lambda_g \leftarrow \lambda_g + 1$
  UCB-UPDATE$(Q_\varphi, N_\varphi, s, a, \lambda_\varphi + c, \beta_\varphi)$
  UCB-UPDATE$(Q_g, N_g, s, a, \lambda_g + c, \beta_g)$
  return $(\lambda_\varphi, \beta_\varphi, \lambda_g, \beta_g)$
procedure UCB-UPDATE$(Q, N, s, a, \lambda, \beta)$
  $Q(s,a) \leftarrow \frac{N(s,a)Q(s,a) + u(\lambda) + K_g \mathbf{1}_{[\beta]}}{1 + N(s,a)}$
  $N(s) \leftarrow N(s) + 1$; $N(s,a) \leftarrow N(s,a) + 1$

util is a strictly decreasing utility function over cost, and K_g is a constant utility for reaching the goal g.

By using UCT with the GUBS criterion, Q-values estimate expected utility as specified by the GUBS utility function [27]. This allows it to maintain the exploration-exploitation balance of UCT while properly handling scenarios with unavoidable dead ends—cases where conventional expected cost minimization breaks down.

In LAMP, we further adapt UCT to learn action values for not only the top-level goal g, but also each landmark $\varphi \in \Phi$, as well as to learn landmark values.

- $Q_g(s, a)$ estimates the expected utility of selecting action a in state s to achieve goal g.
- $Q_\varphi(s, a)$ estimates the expected utility of selecting action a in state s to achieve landmark φ.
- $Q_{LM}(\Phi, \varphi)$ estimates the expected utility if we select landmark φ from landmark graph Φ to achieve next.

We also maintain corresponding values for N_g, N_φ, and N_{LM}, where $N(s, a)$ is the number of times a was selected in state s, and $N(s)$ is the number of times s was visited.

We modified the UCT rollouts to learn these Q-values simultaneously (Algorithm 3.4). Recall that in standard UCT, during each rollout, actions are selected using UCB1 values for each state-action pair, where

$$\text{UCB1}(Q, N, s, a) = Q(s, a) + C\sqrt{\frac{\log(N(s))}{N(s,a)}} \tag{3.5}$$

and C is the exploration constant. In state s, the action that yields the highest UCB1 value is selected. However, in Algorithm 3.4 Line 11, we adjusted the algorithm to account for the greediness

parameter θ; actions are selected using a linear combination of UCB1 values corresponding to both the top-level goal and the current landmark (cf. Equation 3.3).

During each rollout, four values are backpropagated through the search tree: the rollout cost for the current landmark φ, the rollout cost for the top-level goal g, and boolean values to indicate whether φ and g were actually achieved. The subgoal cost is used to update Q_φ, while the top-level cost is used to update Q_g and Q_{LM}. Whenever a landmark φ is achieved, it is removed from the landmark graph Φ, and a new landmark $\varphi' \in \text{leaves}(\Phi)$ is selected using UCB1 values calculated from Q_{LM}. The rollout then continues with the new subgoal, and a subgoal cost of 0 is backpropagated. Whenever the top-level goal is reached, the rollout terminates and both a top-level cost and subgoal cost of 0 are backpropagated.

We additionally imposed a maximum depth for rollouts. If a terminal state is not reached after a fixed number of actions, the rollout is halted. During rollouts, action pruning, as implemented in PROST [46], is used to reduce the branching factor by removing actions with the same distribution over successor states as another action.

We emphasize that these modifications to UCT only add minimal computational overhead. LAMP adds decision nodes to the Monte Carlo search tree whenever a landmark needs to be selected, which means that during each rollout, the number of nodes visited increases at most by the number of landmarks in Φ. Assuming the number of landmarks is small, i.e., $|\Phi| \ll |S|$, the runtime for a rollout does not significantly increase; each rollout incurs at most $O(|\Phi|)$ overhead. If no landmarks are present, a single trivial subgoal corresponding to the final goal g is used, and LAMP is equivalent to standard UCT.

3.5 Experiments and Results

We evaluated LAMP (Algorithm 3.3) on a set of benchmark probabilistic planning problems. For our experiments, we set an execution budget of 200 action executions, a maximum rollout depth of 20, and an exploration constant of $\sqrt{2}$. We used a non-heuristic implementation of UCT; all Q and N values were initialized to 0. For the GUBS criterion, the goal utility constant K_g was set to 1, and the utility function $u : cost \mapsto \exp\left(-\frac{1}{10} \cdot cost\right)$ was used, mirroring the default parameter settings of Crispino et al. [27].

We also used Fast Downward's implementation of the LM^{RHW} landmark extraction algorithm [64] to compute a landmark graph, including all landmark orderings. Landmarks trivially satisfied in the initial state were pruned from the landmark graph, and the top-level goal was added to the landmark graph, ordered such that all other landmarks are predecessors of the top-level goal.

In the experiments that follow, we varied the number of rollouts n_{ro} using values from $\{5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$ and the greediness parameter $\theta \in \{0, 0.2, 0.5, 0.8, 1\}$. When $\theta = 0$, LAMP is equivalent to standard UCT and serves as the baseline.

3.5.1 Simple Benchmarks

Our first set of experiments uses domains and problem instances from the probabilistic track of the Fifth International Planning Competition, which are publicly available [36].

- `prob_blocksworld`: a probabilistic variation of the standard "blocksworld" where blocks have a chance of slipping from the gripper onto the table when being picked up, stacked, and unstacked.

- `elevators`: a person must navigate between floors and elevator shafts to collect coins while

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0_{UCT}	0.2	0.5	0.8	1
5	190.7	**154.8**	**144.1**	**150.7**	**163.0**
10	179.3	**113.7**	**123.1**	**135.7**	**122.1**
20	168.9	**106.3**	**108.3**	**109.6**	**133.3**
50	149.0	**62.8**	**75.4**	**72.5**	**89.1**
100	113.4	**42.1**	**47.5**	**48.7**	**72.0**
200	69.5	**28.7**	**27.9**	**36.3**	**40.9**
500	37.4	**20.7**	**22.5**	**23.2**	**24.5**
1000	27.5	**18.4**	**19.6**	21.2	22.2
2000	20.8	**17.8**	18.5	**18.1**	19.4
5000	19.8	**16.1**	**16.5**	17.6	18.4

(a) `prob_blocksworld` problem `p5`

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0_{UCT}	0.2	0.5	0.8	1
5	179.6	**60.7**	**90.2**	**77.1**	**77.2**
10	175.7	**23.0**	**32.2**	**29.6**	**25.7**
20	159.3	**27.2**	**27.8**	**23.6**	**23.6**
50	105.0	**22.2**	**22.7**	**16.4**	**27.3**
100	74.0	**18.6**	**21.9**	**16.9**	**17.7**
200	42.1	**17.5**	**17.5**	**15.5**	**16.3**
500	24.8	**16.7**	**16.3**	**14.4**	**14.9**
1000	18.2	17.7	17.6	16.9	16.9
2000	16.2	15.3	**14.7**	**14.3**	**14.7**
5000	15.2	15.2	**13.8**	**13.7**	**14.2**

(b) `elevators` problem `p5`

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0_{UCT}	0.2	0.5	0.8	1
5	200.0	200.0	200.0	200.0	199.8
10	200.0	200.0	199.9	200.0	199.7
20	200.0	195.2	199.2	200.0	198.4
50	200.0	199.3	197.2	200.0	199.0
100	200.0	197.0	198.3	198.6	199.5
200	200.0	198.4	197.6	198.2	198.0
500	200.0	196.7	**195.1**	**197.3**	198.5
1000	200.0	197.5	**195.3**	**196.8**	**197.1**
2000	200.0	**193.0**	**192.3**	**190.6**	**190.3**
5000	200.0	**189.3**	**187.6**	**188.5**	**189.3**

(c) `zenotravel` problem `p5`

Figure 3.7: Average cost (lower is better) of the solutions generated by LAMP. LM^{RHW} generated 10, 11, and 12 nontrivial landmarks for these problem instances, respectively. Underlined values indicate the best performing θ-value in the row. Boldfaced values indicate where LAMP significantly ($p < 0.0125$) dominated standard UCT ($\theta = 0$) at the same number of rollouts. Statistical analysis and results from other problem instances are in Appendix A.

avoiding gates which may send the person back to the first floor.

- `zenotravel`: people must move between cities using airplanes. An airplane requires fuel to fly, and can be flown at two different speeds—the higher speed requiring more fuel.

We evaluated LAMP on 29 problem instances over these three domains. For illustrative purposes, Figure 3.7 presents results from one representative instance per domain. For each problem, we report the average cost of solutions generated by LAMP over 75 runs. Runs that exceeded the execution budget of 200 were halted and contributed a cost of 200 to the average. Results for the remaining problem instances were qualitatively similar and are provided in Appendix A.

These results show that a greedy approach can significantly improve performance when the planner is constrained to fewer rollouts. Even though standard UCT ($\theta = 0$) provably converges to an optimal solution [47], in domains like `prob_blocksworld`, `elevators`, and `zenotravel`, a greedy approach performs significantly better than standard UCT because the number of rollouts is insufficient for the non-greedy approach to converge. For the `zenotravel` problem, even with 5000 rollouts, standard UCT was never able to reach a goal within the cost budget while the greedy approach could. However, we also observe that the optimal greediness value θ depends not only on the problem, but also the number of rollouts performed. A greediness value of $\theta \approx 0.2$ yielded the lowest average cost for the `prob_blocksworld` problem, except for the cases with 5 and 200 rollouts, where $\theta = 0.5$ performed best. In contrast, $\theta = 0.8$ yielded the lowest average cost for the `elevators` problem, except in the cases of 5 and 10 rollouts. For the `zenotravel` problem, the optimal greediness value varied between 0.2 and 1.

These observed differences are often significant. We ran a Mann–Whitney U test to analyze the difference in performance between LAMP and baseline UCT ($\theta = 0$) at the same number of

rollouts for each value of θ. This non-parametric alternative to the t-test is ideal for our data as it does not assume a normal distribution, instead comparing the ranks of the scores to determine if one algorithm outperforms the other. LAMP statistically ($p < 0.0125$, the Bonferroni-adjusted $\alpha_{stat} = 0.05/4$) dominates standard UCT in 4 of 10 rows for the `zenotravel` problem, 9 of 10 rows for the `elevators` problem, and all 10 rows for the `prob_blocksworld` problem (details in Appendix A). In the complete set of results, LAMP significantly outperformed standard UCT in 25 of 29 simple benchmark problems, and in 172 of 290 total rows.

3.5.2 Probabilistically Interesting Benchmarks

The `prob_blocksworld`, `elevators`, and `zenotravel` domains do not contain deadlock states, i.e., states from which the top-level goal is not reachable, meaning some may find these problems to be "*probabilistically uninteresting*," a term coined by Little and Thiebaux [50]. There

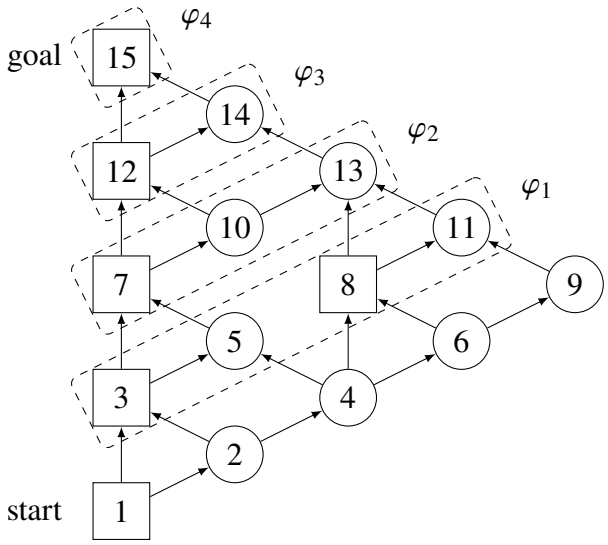


Figure 3.8: A depiction of `triangle_tireworld` problem `p2` of moving a car from node 1 to node 15. Each time the car moves between nodes, there is a 50% chance it gets a flat tire. There are spare tires at circular nodes. All roads are "one-way." The only safe solution is to move to node 9, then to node 15. The landmarks generated by LM^{RHW} are shown in the dashed boxes; e.g., $\varphi_2 \equiv (\text{car-at}(7) \vee \text{car-at}(10) \vee \text{car-at}(13))$.

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0_{UCT}	0.2	0.5	0.8	1
5	0.00	0.03	**0.17**	**0.12**	0.04
10	0.00	**0.12**	**0.09**	**0.15**	0.08
20	0.01	**0.20**	**0.20**	**0.17**	0.09
50	0.05	**0.24**	**0.27**	0.16	0.15
100	0.07	**0.29**	**0.25**	**0.28**	0.07
200	0.08	**0.40**	**0.40**	**0.24**	0.07
500	0.12	**0.41**	**0.41**	**0.44**	0.11
1000	0.41	0.39	0.47	0.52	0.12
2000	0.39	0.60	0.48	0.45	0.16
5000	0.55	0.48	0.59	0.59	0.08

(a) `exploding_blocksworld` problem `p4`

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0_{UCT}	0.2	0.5	0.8	1
5	0.37	**0.71**	**0.64**	**0.67**	0.60
10	0.49	**0.81**	**0.75**	**0.81**	0.68
20	0.77	0.81	0.80	0.81	0.81
50	0.77	0.81	0.85	0.84	0.76
100	0.85	0.84	0.87	0.77	0.76
200	0.84	0.77	0.83	0.80	0.87
500	0.81	0.84	0.85	0.85	0.84
1000	0.87	0.93	0.95	0.84	0.79
2000	0.96	0.97	0.97	0.80	0.83
5000	0.92	0.97	0.97	0.93	0.87

(b) `tireworld` problem `p15`

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0_{UCT}	0.2	0.5	0.8	1
5	0.71	0.21	0.27	0.29	0.35
10	0.71	0.21	0.28	0.27	0.25
20	0.89	0.27	0.13	0.16	0.27
50	0.97	0.25	0.20	0.11	0.29
100	0.99	0.33	0.20	0.13	0.15
200	1.00	0.47	0.12	0.15	0.31
500	1.00	0.35	0.23	0.15	0.08
1000	1.00	0.41	0.33	0.11	0.12
2000	1.00	0.43	0.29	0.09	0.23
5000	1.00	0.47	0.33	0.17	0.19

(c) `triangle_tireworld` problem `p2`

Figure 3.9: Success rate (higher is better) of LAMP reaching a goal state. LM^{RHW} generated 9, 3, and 4 nontrivial landmarks for these problem instances, respectively. Underlined values indicate the best performing θ-value in the row. Boldfaced values indicate where LAMP significantly ($p < 0.0125$) dominated standard UCT ($\theta = 0$) at the same number of rollouts. Statistical analysis and results from other problem instances are in Appendix B.

is, however, no free lunch. As we will see, in problems with deadlock states, a greedy approach together with a poorly placed landmark can quickly lead the planner to fail. In the results that follow, we identify problem instances from three deadlocking domains that demonstrate this to varying degrees:

- `exploding_blocksworld` [36]: a dead-end variation of standard blocksworld where blocks can detonate upon being put down.
- `tireworld` [36]: a vehicle must navigate a road network to reach a goal. Every time the vehicle moves, it has a 40% chance of getting a flat tire. Some locations have spare tires.
- `triangle_tireworld` [50]: a variation of `tireworld` with a 50% flat-tire rate and a triangular road network (Figure 3.8).

We evaluated LAMP on 29 problem instances over these three domains. For illustrative purposes, Figure 3.9 presents results from one representative instance per domain. Given the potential for failure, for each problem, we reported the success rate of LAMP in reaching the goal within the execution budget, over 75 runs. Results for the remaining problem instances are provided in Appendix B.

In the `exploding_blocksworld` problem, standard UCT performs significantly worse, even at 1000 rollouts, with a greediness value between 0.2 and 0.5 yielding the best results. In the `tireworld` problem, a greedy approach dominates standard UCT when constrained to fewer than 10 rollouts, but gradually loses its advantage as the number of rollouts increases.

The `triangle_tireworld` problem, depicted in Figure 3.8, is specifically chosen to elicit pathological behavior from a greedy planner, representing a worst-case scenario for LAMP. Although

a safe solution exists, where the car moves to node 9, then to the goal, the identified landmarks can mislead a greedy planner toward an unsafe path. Starting from node 1, the fastest way to achieve the first landmark, φ_1, is to move to node 3. Although moving to node 3 does not lead to an immediate deadlock, it is unsafe, which explains why any inclination toward greediness will only lower the overall success rate. A greedy approach may enter unsafe or deadlock states while achieving a landmark, overlooking that this may prevent it from reaching a future subgoal.

We analyzed the significance of the observed differences between LAMP and baseline UCT using a two-tailed Boschloo exact test. Boschloo's test is an unconditional version of Fisher's exact test appropriate for 2×2 contingency tables (success/failure vs. LAMP/baseline). With a Bonferroni-adjusted $\alpha_{stat} = 0.05/4$, LAMP statistically dominates standard UCT in 6 of 10 rows in the `exploding_blocksworld` problem, and in 2 of 10 rows in the `tireworld` problem. However, standard UCT dominates all greedy variants in the `triangle_tireworld` problem. Full statistical analysis and results from other problems are in Appendix B. In the complete set of results, LAMP significantly outperformed standard UCT in 11 of 29 probabilistically interesting benchmark problems, and in 42 of 290 total rows.

3.5.3 Discussion

The results indicate that in problems without deadlock states, when the algorithm is constrained to a small number of rollouts, a greedy approach ($\theta > 0$) often outperforms standard UCT ($\theta = 0$). As the number of rollouts grows, standard UCT will eventually converge to an optimal solution; however, for larger planning problems, this can take prohibitively long. In contrast, a greedy approach decomposes the large problem into smaller subproblems, allowing UCT to converge to viable policies for these subproblems much more quickly than for the top-level goal. This makes a

greedy algorithm an advantageous choice in large domains or for anytime settings. However, in a domain with deadlock states, a greedy approach may inadvertently enter a deadlock state while achieving a landmark, thereby preventing it from achieving the top-level goal. While a greedy approach can still outperform the baseline in such scenarios, careful selection of landmarks is crucial to ensure that none of them contain or lead to deadlock states.

3.6 Summary and Future Work

We extended classical landmarks to probabilistic settings and introduced LAMP, a particular implementation of LANDMARK-PLAN based on UCT, as a domain-independent probabilistic planner that leverages classical landmarks as subgoals in MDPs. LAMP outperforms standard UCT on several benchmark planning problems, demonstrating the effectiveness of using landmarks for probabilistic planning, and suggesting that it may be worthwhile to incorporate landmark guidance in more sophisticated probabilistic planning systems.

We also studied the trade-off between a greedy and non-greedy approach to achieving landmarks during planning. A non-greedy approach—equivalent to UCT—provably converges to an optimal solution without landmarks but may take prohibitively long to find solutions in large planning domains, making it impractical for online or time-constrained applications. In contrast, a greedy approach decomposes the planning task into a sequence of simpler subproblems. It often outperforms plain UCT with fewer rollouts, albeit at the cost of completeness. Problems where landmarks may include deadlock states or unsafe states may pose a challenge to a greedy algorithm.

Future work could focus on integrating deadlock detection and avoidance into LAMP, mitigating the risk of poorly chosen landmarks. More broadly, it could investigate alternative methods, beyond

probabilistic landmarks, to identify effective subgoals for problem decomposition. Since we observed that the optimal greediness value θ depends on both the problem domain and the number of rollouts, future work could investigate the potential for an adaptive θ-value that changes as planning progresses. Finally, using other probabilistic planning techniques, including non-sampling-based approaches, to implement the general landmark-based decomposition approach outlined in LANDMARK-PLAN (Algorithm 3.2) could yield improvements similar to those achieved by our UCT-based approach.

Chapter 4: Hierarchical Goal Networks for Probabilistic Planning[1]

Classical and probabilistic planners typically operate on a "flat" representation of a problem, searching the space of primitive actions to find a path from the initial state to a goal. While effective for some problems, this approach struggles as the required plan length and domain complexity grow. For long-horizon problems, the exponential size of the state space may make the discovery of a solution computationally intractable. To overcome this challenge, hierarchical planning introduces a more structured approach to reasoning. Instead of searching directly through primitive actions, it operates at multiple levels of abstraction, breaking down high-level objectives into smaller, more manageable subproblems, in much the same way that a person planning a trip would think about "booking a flight" and "packing a bag" rather than the individual muscle movements required.

A prominent formalism for this is *hierarchical task network* (HTN) planning. In HTN planning, the planner is given a set of high-level *tasks* to accomplish and a library of *methods*. These methods are expert-defined recipes for how to decompose a complex task into simpler subtasks. Each method addresses a specific task by providing an ordered set of subtasks; each

[1]This chapter is based on the following publications:
[22] D. H. Chan, M. Roberts, D. S. Nau, "Probabilistic Hierarchical Goal Network Planning with UCT," in *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence (AAAI 2026)*, Singapore, January 20–27, 2026 [in press].
[21] D. H. Chan, M. Roberts, D. S. Nau, "Probabilistic Hierarchical Goal Network Planning: Preliminary Results," in *Proceedings of the Eighth ICAPS Workshop on Hierarchical Planning (HPlan 2025)*, Melbourne, Australia, November 10, 2025.

subtask may, in turn, be addressed by other methods, or achieved by primitive actions. For example, a task travel(cityA, cityB) of traveling from cityA to cityB may be addressed by a method m-travel-by-plane(cityA, cityB) which decomposes the task travel(cityA, cityB) into a sequence of subtasks ⟨book-flight(cityA, cityB), pack-bag(cityB), fly(cityA, cityB)⟩. These subtasks may be further decomposed into subtasks, or addressed by primitive actions. By recursively applying these methods, the planner refines an abstract task until it arrives at a sequence of primitive, executable actions. To solve HTN planning problems, classical HTN planners typically use a form of backtracking search to identify the methods and actions required to accomplish the given high-level task.

While HTNs provide highly expressive, domain-specific guidance, they typically require carefully designed task hierarchies and often lack domain-independent heuristics. To address this, hierarchical goal networks (HGNs) were developed as an alternative that integrates more naturally with state-based planning [71]. Instead of decomposing abstract tasks, HGNs decompose goals, which are conditions that must be made true in the world state. Because goals correspond to world states, HGN planning allows for a more seamless application of classical reasoning techniques while still retaining the expressive power of a hierarchy.

Real-world domains often involve uncertainty, where actions can produce nondeterministic outcomes [56], [57]. While hierarchical planning has typically been studied in deterministic settings, recent work has extended HTN planning to fully observable nondeterministic (FOND) domains [23], [24], [86]. Despite these advances, no prior work has integrated hierarchical goal network planning with probabilistic action models.

In this chapter, we formalize probabilistic HGN planning semantics. We construct two UCT-based

algorithms [47] that leverage HGN knowledge: (1) $\text{UCT}^{base}_{\text{HGN}}$, a baseline approach that transforms a probabilistic HGN problem into an equivalent stochastic shortest path (SSP) problem and applies UCT directly to solve it, yielding asymptotically optimal behavior; and (2) $\text{UCT}^{comp}_{\text{HGN}}$, a compressed variant of UCT that reduces the size of the search tree by sharing learned value estimates across related goal networks. Results on four benchmarks adapted from FOND HTN domains show that $\text{UCT}^{base}_{\text{HGN}}$ performs well on smaller problems but struggles to scale to larger problems, while the $\text{UCT}^{comp}_{\text{HGN}}$ variant converges more quickly, requires less memory, and consistently finds lower-cost solutions on larger problems. These results suggest that a compressed UCT that abstracts over goal networks offers an effective, scalable approach to probabilistic HGN planning.

4.1 Related Works

Hierarchical task network (HTN) planning [30, 53, 54] addresses the challenges of large, complex problems by viewing planning as the accomplishment of tasks through hierarchical decomposition. In HTN planning, hierarchical structure is defined through methods, which specify how a compound task may be decomposed into a set of subtasks. Subtasks may themselves be compound tasks requiring further decomposition or primitive actions that can be executed directly. By recursively applying methods, the planner refines an initial high-level task into a sequence of executable actions. By encoding hierarchical structure and domain control knowledge, HTNs can significantly reduce search effort and are strictly more expressive than classical planning [29]. However, HTN tasks are syntactic constructs that do not directly correspond to world-state goals; rather, they represent abstract activities whose semantics are determined entirely by the set of available decomposition methods. This task-goal decoupling introduces several practical challenges. Reachability analysis

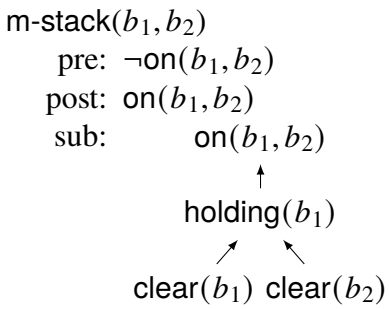


Figure 4.1: An example of a partial-order HGN method for the `blocksworld` problem in Figure 2.1. An HGN method consists of a precondition, postcondition, and subgoal decomposition, where the subgoals, denoted sub(m-stack), are represented in a directed acyclic graph. If an HGN planner is tasked with achieving a goal of the form on(b_1, b_2), then method m-stack may be invoked to decompose the goal into subgoals. Subgoals may either be further decomposed by other methods, or directly achieved by executing primitive actions.

becomes non-trivial because tasks and goals may not align; incomplete or incorrect methods can cause search to fail; and designing domain-independent heuristics is difficult since tasks do not explicitly specify the world conditions they achieve. Moreover, HTN planning typically requires a domain expert to provide complete models of domain-specific knowledge, which can be especially problematic in open and dynamic environments. Recent advances address some of these limitations, developing search mechanisms which include reachability analysis [8], [11], heuristic search [10], [45], compilation to classical planning [1], [3], [9], [44] or SAT [7], [61], [67], and Monte Carlo tree search adaptations [83].

To address the limitations of HTN planning, hierarchical goal networks (HGNs) were proposed as a hybrid formalism that combines the strengths of hierarchical and classical planning [71]. Unlike HTNs, which decompose abstract tasks, HGNs decompose explicit world-state goals into partially ordered sequences of subgoals. In HGN planning, methods specify how a goal may be decomposed into subgoals (see Figure 4.1). These subgoals may, in turn, be decomposed by other methods or achieved directly by primitive actions. This goal-centric formulation preserves the semantic

grounding of classical planning while retaining the hierarchical decomposition structure established by HTN planning. The decomposition semantics of HGNs resemble those of HTNs, but operate directly over goal conditions rather than syntactic tasks. Because goals are specified declaratively as logical conditions, HGN planning enables the planner to determine which methods and primitive actions are relevant and applicable based on the current state. As shown by Shivashankar et al. [71], HGN planning is as expressive as HTN planning, meaning that HGNs retain the expressivity to encode complex domain control knowledge—one of the principal advantages of HTN planning [2].

Building on this formalism, Shivashankar et al. [71] introduced the Goal Decomposition Planner (GDP), which integrates hierarchical decomposition with domain-independent classical planning techniques. In particular, GDP admits the use of domain-independent heuristic functions to guide both method selection and primitive action application. By making goals explicit in the hierarchical structure, HGNs simplify method authoring and enable the direct use of classical planning heuristics [68], [69]. However, the challenge of providing a complete and correct set of hierarchical methods remains. In hierarchical planning, domain knowledge is encoded through methods that specify how tasks or goals should be decomposed. In many practical domains, it is difficult for a domain expert to enumerate methods covering all possible situations the planner may encounter.

To work around this, Shivashankar et al. [70] developed the GoDeL (Goal Decomposition with Landmarks) planning algorithm, which adopts action-insertion semantics. Under action-insertion semantics, the planner is permitted to execute primitive actions even if they are not explicitly prescribed by the current goal hierarchy. Intuitively, action-insertion allows the planner to temporarily bypass the hierarchical structure when necessary, enabling classical planning search to achieve goals that are not covered by the available methods. This design ensures completeness

regardless of the completeness of the provided domain knowledge. Experimental results demonstrate that GoDeL functions correctly even when the available methods provide only partial coverage of the domain, meaning that some goals may not have corresponding decomposition methods. Furthermore, its performance improves as additional hierarchical knowledge is provided, matching GDP when domain knowledge is complete. By enabling hierarchical planning under incomplete domain knowledge, GoDeL improves the practical applicability of HGN planning.

While HGNs address many limitations of traditional HTN planning, both classical HTN and HGN approaches have been primarily limited to deterministic environments. HTN planning has recently been extended to the nondeterministic setting. Notably, work by Chen and Bercher [23] formalized FOND HTN planning. Subsequent work introduced policy-based variants that allow decomposition decisions to be contingent on outcomes and proposed methods for generating strong and weak solutions using deterministic relaxations [24], [86]. Planners such as UPOM [56] combine hierarchical decomposition with MCTS for planning under uncertainty. Concurrent with our work, Yousefi et al. [87] independently developed a probabilistic HTN formulation with known action outcome distributions. However, this does not leverage the representational advantages of HGNs.

No prior work has addressed the use of HGNs in a probabilistic setting. Yet, many properties of HGN planning that make it attractive for deterministic planning extend to the probabilistic setting. The goal-based structure of HGNs enables planners to reason over ordered sequences of goals—a form of temporally extended goal—and 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; in such environments, the strict procedural task sequences of HTNs may be too brittle. Furthermore, action-insertion semantics support the

interleaving of goal decomposition and heuristically-guided action selection. These benefits provide a principled basis for integrating hierarchical domain knowledge with search.

Building on the foundation of HGN planning and recent advances in FOND HTN planning, our work extends hierarchical planning to probabilistic domains with known action outcome distributions. In contrast to the HTN-based approaches, our method leverages the goal-centric structure of HGNs, applying them to stochastic domains with UCT-based search. Probabilistic HGN planning leverages the expressiveness of hierarchical models while maintaining the flexibility required in probabilistic domains.

4.2 Preliminaries

Our formalism for probabilistic hierarchical goal network planning rests on the same foundations as classical HGN planning.

Definition 1 (Goal Network [2])**.** Let $\mathcal{L}$ be a propositional language with a set of 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 for goals, $\prec \subseteq I \times I$ is a partial order on I, and $\alpha : I \rightarrow \mathcal{F}$ maps each symbol in I to a goal formula.

A goal network is a partially ordered multiset of goals, represented using a set I of indices, or labels, for goals. 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 $UC(gn) = \{i \in I \mid i \text{ has no } \prec\text{-predecessors}\}$.

Because the specific symbols used to label goals in a goal network are arbitrary and semantically unimportant, it is standard in the current literature to define an isomorphism relation to formally capture when two goal networks are structurally equivalent [2]. To simplify our reasoning about

goal networks and avoid label collisions, we fix a canonical set G of all goal networks such that all index sets are disjoint. That is, $G = \{(I_1, \prec_1, \alpha_1), (I_2, \prec_2, \alpha_2), \dots\}$ is a complete set of goal networks up to isomorphism such that for all $i \neq j$, $I_i \cap I_j = \varnothing$. We also fix $\tilde{I} = \bigcup_{(I,\prec,\alpha)\in G} I$ to be the set of all labels across all goal networks in G.

Definition 2 (Method [2])**.** A *(goal) method* m is a tuple $(\text{pre}(m), \text{post}(m), \text{sub}(m))$ where $\text{pre}(m), \text{post}(m) \in \mathcal{F}$ are the pre- and postconditions of m, respectively, and $\text{sub}(m) = (I_m, \prec_m, \alpha_m) \in G$ is a goal network.

A method m 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 $\text{post}(m)$ matches a literal in the NNF of $\alpha(i)$. This ensures that at least part of $\alpha(i)$ is true by accomplishing $\text{post}(m)$. To ensure that m accomplishes its own goal, we require that there exists $i \in I_m$ such that $\alpha_m(i) = \text{post}(m)$ and for all $j \in I_m$, if $j \neq i$ then $j \prec i$.

4.3 Extension to Probabilistic Planning

We adapt partial-order classical HGN planning [2] to incorporate probabilistic actions, drawing inspiration from FOND HTN planning [86].

Definition 3 (Probabilistic HGN Planning Domain)**.** Let $\mathcal{L}$ be a propositional language with a set of atoms $\mathcal{X}$ and propositional formulae $\mathcal{F}$. A *probabilistic HGN planning domain* is a tuple $\mathcal{D}_{\text{HGN}} = (\mathcal{L}, M, A, \gamma, \Pr)$, where:

- $S = 2^{\mathcal{X}}$ 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 $a = (\text{pre}(a), \text{eff}(a)) \in A$ where $\text{pre}(a) \in \mathcal{F}$ is the precondition, and $\text{eff}(a) \in 2^{2^{\mathcal{X}} \times 2^{\mathcal{X}}}$ is a set of pairs $(\text{add}(a), \text{del}(a))$ of add

effects $\text{add}(a) \subseteq \mathcal{X}$ and delete effects $\text{del}(a) \subseteq \mathcal{X}$;

- $\gamma : S \times A \to 2^S$ is a transition function where $\gamma(s,a)$ is defined iff $s \models \text{pre}(a)$, and $\gamma(s,a) = \{(s \backslash \text{del}(a)) \cup \text{add}(a) \mid (\text{add}(a), \text{del}(a)) \in \text{eff}(a)\}$; 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.

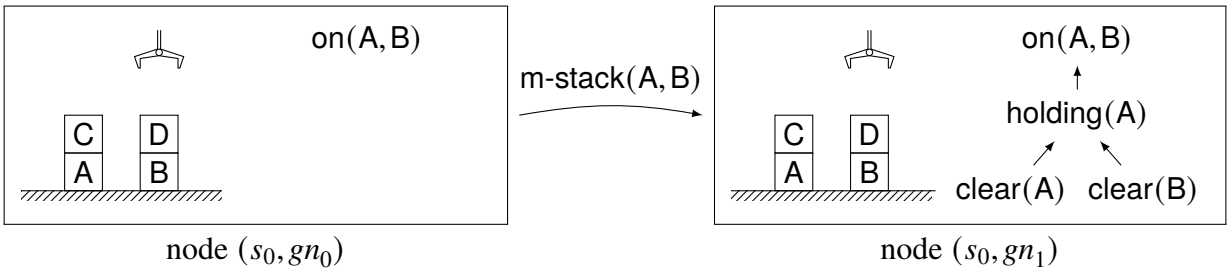
An action or method u is *applicable* in state s if $s \models \text{pre}(u)$. For simplicity, we assume all actions have unit cost.

Definition 4 (Node)**.** 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.

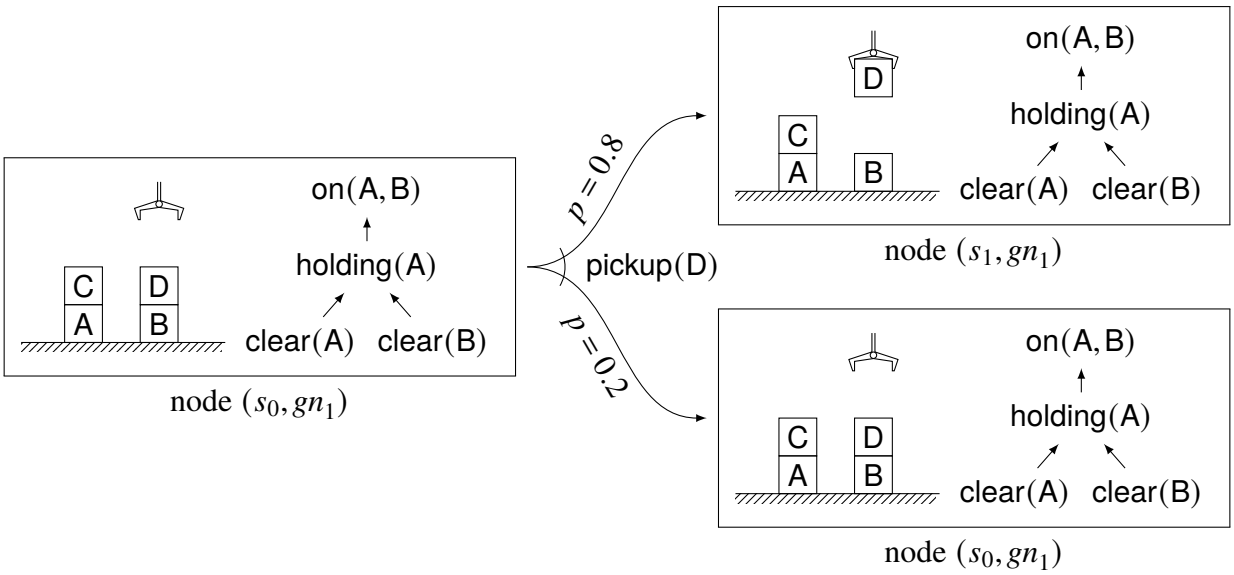
We adapt the three forms of node progression from classical HGN planning: goal release, goal decomposition, and action application. An example of each is depicted in Figure 4.2.

Definition 5 (Node Progression)**.** Let (s, gn) be a node, where $gn = (I, \prec, \alpha)$.

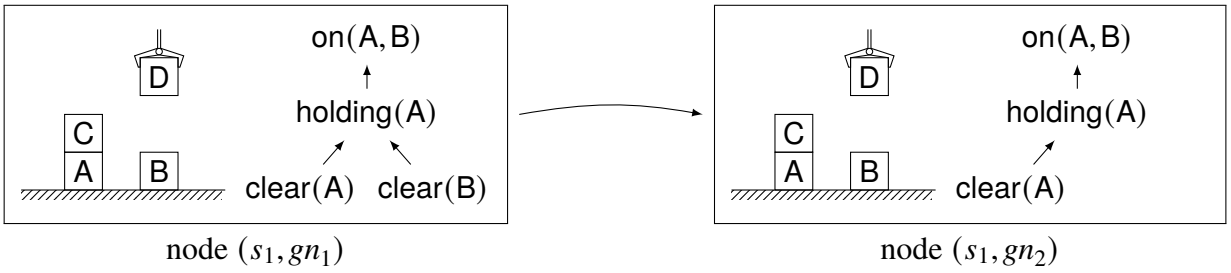
- *Deterministic Goal Release* (P^i_{rel}): Let $i \in UC(gn)$ such that $s \models \alpha(i)$. Release of i removes it from gn to yield the node (s, gn'), where $gn' = (I \backslash \{i\}, \{(i_1, i_2) \in \prec \mid i_1 \neq i\}, \{(i', g) \in \alpha \mid i' \neq i\})$. We denote this by $(s, gn') = P^i_{rel}(s, gn)$.
- *Deterministic Goal Decomposition* ($P^{i,m}_{dec}$): Suppose $s \not\models \alpha(j)$ for all $j \in UC(gn)$. Let $i \in UC(gn)$, and let $m \in M$ be relevant to i and applicable in s, where $\text{sub}(m) = (I_m, \prec_m, \alpha_m)$. Decomposition by m prepends gn with $\text{sub}(m)$ 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 $(s, gn') = P^{i,m}_{dec}(s, gn)$.
- *Probabilistic Action Application* (P^a_{app}): Suppose $s \not\models \alpha(i)$ for all $i \in UC(gn)$, and let $a \in A$ be applicable in s. Application of 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 $(s', gn) \sim P^a_{app}(s, gn)$.



(a) Deterministic goal decomposition



(b) Probabilistic action application



(c) Deterministic goal release

Figure 4.2: The three forms of node progression for probabilistic HGN planning in the `blocksworld` domain. Method m-stack is as defined in Figure 4.1, and action pickup is a probabilistic action that has an 80% chance of picking up a block, and a 20% chance of failure resulting in no operation.

Our model of probabilistic HGN planning with action-insertion semantics allows actions to be executed independently of the hierarchy, as in work by Shivashankar et al. [70]. That is, an action that is applicable in the current state can be executed without requiring "relevance" to an unconstrained goal i. This flexibility is particularly helpful in probabilistic settings where primitive action chaining may be needed to recover from unexpected states caused by stochastic actions, or incomplete hierarchical models.

We also impose an additional constraint: subgoals are immediately released from the goal network once satisfied. That is, goal decomposition and action application are disallowed until all unconstrained subgoals satisfied in the current state are released. This captures our adaptation of the action-insertion semantics defined by Shivashankar et al. [70].

Definition 6 (Probabilistic HGN Planning Problem)**.** A *probabilistic HGN planning problem* is a tuple $\mathcal{P}_{\text{HGN}} = (\mathcal{D}_{\text{HGN}}, s_0, gn_0)$, where $\mathcal{D}_{\text{HGN}}$ is a probabilistic HGN planning domain, $s_0 \in S$ is the initial state, and $gn_0 \in G$ is the initial goal network.

In probabilistic HGN planning, the planning objective is specified using an initial goal network rather than a single goal condition. This goal network effectively serves as a form of temporally extended goal, and planning is complete once all goals in the network have been successfully released. We denote the empty goal network as $gn_\varnothing = (\varnothing, \varnothing, \varnothing)$. 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. In fact, since we allow action insertion, if no HGN methods are provided and the initial goal network contains only a single goal, probabilistic HGN planning reduces to standard (non-hierarchical) probabilistic planning.

While solutions to standard probabilistic planning problems are policies over actions, HGN

planning requires selecting both actions and methods. To that end, we introduce *action-method policies* to represent solutions to probabilistic HGN planning problems, where actions and methods can be selected based on the current node. These are analogous to the "*method-based policies*" of Chen and Bercher [24].

Definition 7 (Action-Method Policy)**.** An *action-method policy* is a partial mapping $\hat{\pi} : S \times G \to (\{\epsilon\} \times A) \cup (\tilde{I} \times M)$, where ϵ is a reserved symbol to indicate action application. $\hat{\pi}$ is *well-formed* if for all (s, gn) in the domain of $\hat{\pi}$ with $\hat{\pi}(s, gn) = (i, u)$, the following conditions hold: $gn = (I, \prec, \alpha) \neq gn_{\varnothing}$; $s \not\models \alpha(j)$ for all $j \in UC(gn)$; $s \models \mathrm{pre}(u)$; and if $u \in M$ then $i \in UC(gn)$ and u is relevant to i.

To classify the different types of solutions, we define the *reachability graph* of a policy.

Definition 8 (Reachability Graph)**.** Let $\hat{\pi}$ be a well-formed action-method policy for $\mathcal{P}_{\mathrm{HGN}}$. Its *reachability graph* is a tuple $\mathrm{Graph}(\hat{\pi}, s_0, gn_0) = (V, E, p)$, with nodes $V \subseteq S \times G$, edges $E \subseteq V \times V$, and a weight function $p : E \to (0, 1]$.

$\mathrm{Graph}(\hat{\pi}, s_0, gn_0)$ is constructed as follows:

- $(s_0, gn_0) \in V$
- [*goal release*] If $(s, gn) \in V$ and there exists $i \in UC(gn)$ such that $s \models \alpha(i)$, then $(s, gn') \in V$ and $e = ((s, gn), (s, gn')) \in E$, where $(s, gn') = P^{i}_{rel}(s, gn)$ and $p(e) = 1$.
- [*goal decomposition*] If $(s, gn) \in V$ and $\hat{\pi}(s, gn) = (i, m)$ where $(i, m) \in \tilde{I} \times M$, then $(s, gn') \in V$ and $e = ((s, gn), (s, gn')) \in E$, where $(s, gn') = P^{i,m}_{dec}(s, gn)$ and $p(e) = 1$.
- [*action application*] If $(s, gn) \in V$ and $\hat{\pi}(s, gn) = (\epsilon, a)$ where $a \in A$, then for all $s' \in \gamma(s, a)$, $(s', gn) \in V$ and $e = ((s, gn), (s', gn)) \in E$, with $p(e) = \Pr(s' \mid s, a)$.

A node $n \in V$ is *terminal* if it has no outgoing edges. A *goal node* is a terminal node (s, gn) where $gn = gn_{\varnothing}$.

$\text{Graph}(\pi, s_0, gn_0) = (V, E, p)$ defines the reachability graph—called the "*execution structure*" by Yousefi and Bercher (2024)—induced by following $\hat{\pi}$ from $n_0 = (s_0, gn_0)$ using node progression with edge weights corresponding to probability.

Definition 9 (History)**.** Let $\mathcal{P}_{\text{HGN}} = (\mathcal{D}_{\text{HGN}}, s_0, gn_0)$, let $\hat{\pi}$ be a well-formed action-method policy for $\mathcal{P}_{\text{HGN}}$ with reachability graph $\text{Graph}(\hat{\pi}, s_0, gn_0) = (V, E, p)$, and let $n = (s, gn) \in V$. A *history* σ of $\hat{\pi}$ from n is a finite sequence of nodes $\sigma = \langle n_0, n_1, \ldots, n_h \rangle$ such that $n_0 = n$, $\forall j \in \{1, \ldots, h\}$, $(n_{j-1}, n_j) \in E$, and n_h is terminal.

We write $|\sigma|$ to denote the length of σ, and σ_{-1} to denote the final node in σ. We write $H_{\text{HGN}}(\hat{\pi}, n)$ to denote the set of all histories of policy $\hat{\pi}$ from node n. The probability of a history is defined as $\Pr(\sigma \mid \hat{\pi}, n) = \prod_{j=1}^{|\sigma|} p(n_{j-1}, n_j)$.

Definition 10 (Solution Policy)**.** Let $\hat{\pi}$ be a well-formed action-method policy for $\mathcal{P}_{\text{HGN}} = (\mathcal{D}_{\text{HGN}}, s_0, gn_0)$. Let $H^* = \{\sigma \in H_{\text{HGN}}(\hat{\pi}, (s_0, gn_0)) \mid \sigma \text{ terminates at a goal node}\}$. Let $\rho \in [0, 1]$. $\hat{\pi}$ is a *ρ-policy* if $\sum_{\sigma \in H^*} \Pr(\sigma \mid \hat{\pi}, (s_0, gn_0)) \geq \rho$.

$\hat{\pi}$ is *(cyclic)* safe if it is a 1-policy; or equivalently, for all $n \in \text{Graph}(\hat{\pi}, s_0, gn_0)$, there exists $\sigma \in H_{\text{HGN}}(\hat{\pi}, n)$ such that σ_{-1} is a goal node. $\hat{\pi}$ is *acyclic safe* if $\hat{\pi}$ is safe and $\text{Graph}(\hat{\pi}, s_0, gn_0)$ contains no cycles. Otherwise, $\hat{\pi}$ is *unsafe*.

4.4 Probabilistic HGN Planning Algorithms

The second core contribution of this chapter is implementing and evaluating the role of HGNs in probabilistic planning. To that end, we propose two HGN-guided probabilistic planning algorithms based on UCT (see Section 2.3). UCT's simplicity makes it a suitable baseline for analyzing the impact of HGNs. We adapt UCT for goal-directed planning by integrating the GUBS (Goals with Utility-Based Semantics) criterion [33], [34]. GUBS jointly maximizes the probability of goal achievement and minimizes expected cost using a utility function

$$U(\sigma) = util(|\sigma|) + K_g \mathbf{1}_{[\sigma_{-1} \text{ is a goal node}]}, \tag{4.1}$$

where *util* is a strictly decreasing utility over cost and K_g is a goal utility constant. UCT-GUBS, as described by Crispino et al. [27], estimates expected utility using Q-values, preserving the exploration-exploitation balance while effectively handling dead ends.

4.4.1 Baseline UCT ($\text{UCT}^{base}_{\text{HGN}}$)

We first establish a baseline approach using standard probabilistic planning techniques. A probabilistic HGN planning problem $\mathcal{P}_{\text{HGN}} = (\mathcal{D}_{\text{HGN}}, s_0, gn_0)$ can be reformulated as an SSP problem over nodes, where each node captures both the current world state and the current goal network.

Construction 1. Let $\mathcal{P}_{\text{HGN}} = (\mathcal{D}_{\text{HGN}}, s_0, gn_0)$ be a probabilistic HGN planning problem. We construct a (non-hierarchical) SSP $\tilde{\mathcal{P}} = (\tilde{S}, \tilde{A}, \tilde{\Pr}, \tilde{s_0}, \tilde{S_g})$ to represent the planning problem over nodes, where

- $\tilde{S} = S \times G$ is the state space comprising all nodes;
- $\tilde{A} = A_{rel} \cup A_{dec} \cup A_{app}$ where $A_{rel} = \{P^i_{rel} \mid i \in \tilde{I}\}$, $A_{dec} = \{P^{i,m}_{dec} \mid i \in \tilde{I}, m \in M\}$, and $A_{app} = \{P^a_{app} \mid a \in A\}$;
- $\tilde{\Pr}(\tilde{s}' \mid \tilde{s}, \tilde{a}) = \Pr[\tilde{a}(\tilde{s}) = \tilde{s}']$ defines transition probabilities for all $\tilde{a} \in \tilde{A}$ and $\tilde{s} \in \tilde{S}$;
- $\tilde{s_0} = (s_0, gn_0)$ is the initial node; and
- $\tilde{S_g} = S \times \{gn_\varnothing\}$ is the set of goal nodes.

Following our definitions for node progressions, $\tilde{a} \in \tilde{A}$ is applicable in $(s, gn) \in \tilde{S}$ if any of the following hold:

- $\tilde{a} = P^i_{rel} \in A_{rel}$, where $i \in UC(gn)$ and $s \models \alpha(i)$;
- $\tilde{a} = P^{i,m}_{dec} \in A_{dec}$, where $i \in UC(gn), m$ is relevant to i, $s \models \text{pre}(m)$, and $s \not\models \alpha(j)$ for all $j \in UC(gn)$; or
- $\tilde{a} = P^a_{app} \in A_{app}$, $s \models \text{pre}(a)$, and $s \not\models \alpha(i)$ for all $i \in UC(gn)$.

Proposition 1. *Construction 1 maps the probabilistic HGN problem to an equivalent SSP problem.*

Proof Sketch. Construction 1 creates a direct correspondence between solutions to $\tilde{\mathcal{P}}$ and solutions to $\mathcal{P}_{\text{HGN}}$. The state space $\tilde{S}$ mirrors the set of nodes, and $\tilde{S_g}$ corresponds to the set of goal nodes. Each SSP action in A_{rel}, A_{dec}, and A_{app} directly encodes one of the three types of node progression. The only subtlety is that action-method policies for $\mathcal{P}_{\text{HGN}}$ do not explicitly trigger goal releases; instead, goal release is handled automatically by the semantics of node progression whenever a subgoal is satisfied. □

This equivalence allows us to directly apply any SSP algorithm to solve $\mathcal{P}_{\text{HGN}}$. As a baseline, we use the UCT algorithm, which provably converges to the optimal value function with probability 1 in the limit of infinite rollouts [47]. Therefore, UCT applied to the SSP $\tilde{\mathcal{P}}$ will converge to a policy

for $\mathcal{P}_{\text{HGN}}$ which maximizes the GUBS utility. In our implementation, the SSP is not precompiled, but rather it is expanded at runtime by UCT.

4.4.2 Compressed UCT ($\text{UCT}^{comp}_{\text{HGN}}$)

While the baseline $\text{UCT}^{base}_{\text{HGN}}$ is theoretically sound, it suffers from scalability limitations. The state space $\tilde{S} = S \times G$ introduces an additional exponential factor, and thus can grow prohibitively large in problems with complex goal networks, leading to degraded performance in time- or compute-limited settings. In such cases, UCT may fail to converge to a viable solution within the available computational resources.

The inefficiency stems from treating each node (s, gn) as a distinct state. This overlooks the crucial observation that nodes sharing the same underlying world state s and unconstrained subgoals in gn are likely to benefit from similar actions. In other words, the optimal action in a given world state with respect to a goal network can often be approximated using only the immediate subgoals that need to be achieved, rather than the complete, potentially complex structure of the entire goal network.

To address this, we propose a "compressed" UCT approach that creates only one tree node per underlying world state s. Recall that standard UCT builds a search tree where each tree node corresponds to a world state s and maintains a table of Q-values, $Q(s, a)$, for every action $a \in \text{Applicable}(s)$. Our compressed UCT extends this structure for HGN planning so that the node for state s maintains multiple separate tables of Q-values, $Q_g(s, a)$, one for each unconstrained subgoal g in the goal network. This allows the algorithm to share learned information across nodes with different goal networks, leading to more efficient exploration and faster convergence.

Algorithm 4.1 ($\text{UCT}^{comp}_{\text{HGN}}$) A compressed UCT algorithm for probabilistic HGN planning problems. $\mathcal{D}_{\text{HGN}} = (\mathcal{L}, M, A, \gamma, \Pr)$ 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 performed at each step, d is the maximum rollout depth, and c is the cumulative cost (initially 0).

procedure $\text{UCT}^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, n_{ro}, d, c)$
  if $gn = gn_{\varnothing}$ **then**
    return success
  if $\exists i \in UC(gn)$ such that $s \models \alpha(i)$ **then**
    $(s, gn) \leftarrow P^{i}_{rel}(s, gn)$
    return $\text{UCT}^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, n_{ro}, d, c)$
  for n_{ro} times **do**
    UCT-ROLLOUT$^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, d, c)$ ▷ learn $Q_{\alpha(i)}$
  $U \leftarrow \{(i, m) \in UC(gn) \times M \mid m \text{ is applicable in } s \text{ and relevant to } i\}$
  $U \leftarrow U \cup \{a \in A \mid a \text{ is applicable in } s\}$
  if $U = \varnothing$ **then**
    return failure
  $u \leftarrow \arg\max_{u \in U} \sum_{i \in UC(gn)} Q_{\alpha(i)}(s, u)$
  if $u \in A$ **then**
    $s' \leftarrow \text{APPLY}(s, u)$
    return $\text{UCT}^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s', gn, n_{ro}, d, c + 1)$
  else ▷ u is a subgoal-method pair
    $(i, m) \leftarrow u$
    $(s, gn) \leftarrow P^{i,m}_{dec}(s, gn)$
    return $\text{UCT}^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, n_{ro}, d, c)$

Algorithm 4.2 (UCT-ROLLOUT$^{comp}_{\text{HGN}}$) The Monte Carlo rollout procedure for UCT$^{comp}_{\text{HGN}}$. $\mathcal{D}_{\text{HGN}}$, s, gn, and c are as defined in Algorithm 4.1. d is the remaining rollout depth, *util* is a strictly decreasing utility function over cost, and K_g is the goal-utility constant. Q_* and N_* are global maps with default value 0. UCT-ROLLOUT$^{comp}_{\text{HGN}}$ returns a pair of maps $\lambda : I \to \mathbb{R}^{\geq 0}$ and $\beta : I \to \{\mathsf{true}, \mathsf{false}\}$ where $\lambda(i)$ is the rollout cost for subgoal i, and $\beta(i)$ indicates whether subgoal i was achieved.

procedure UCT-ROLLOUT$^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, d, c)$

  if $gn = gn_{\varnothing}$ **then return** $(\varnothing, \varnothing)$

  if $\exists i \in UC(gn)$ such that $s \models \alpha(i)$ **then**

    $(s, gn) \leftarrow P^i_{rel}(s, gn)$

    $(\lambda, \beta) \leftarrow$ UCT-ROLLOUT$^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, d, c)$

    return $(\lambda \cup \{(i, 0)\}, \beta \cup \{(i, \mathsf{true})\})$

  $U \leftarrow \{(i, m) \in UC(gn) \times M \mid m \text{ is applicable in } s \text{ and relevant to } i\}$

  $U \leftarrow U \cup \{a \in A \mid a \text{ is applicable in } s\}$

  if $d = 0$ or $U = \varnothing$ **then return** $(I \times \{d\}, I \times \{\mathsf{false}\})$

  $u \leftarrow \arg\max_{u \in U} \sum_{i \in UC(gn)} \text{UCB1}(Q_{\alpha(i)}, N_{\alpha(i)}, s, u)$

  if $u \in A$ **then**

    sample $(s', gn) \sim P^u_{app}(s, gn)$

    $(\lambda, \beta) \leftarrow$ UCT-ROLLOUT$^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s', gn, d - 1, c + 1)$

    $\lambda \leftarrow \{(i, \lambda(i) + 1) \mid i \in I\}$

  else ▷ u is a subgoal-method pair

    $(i, m) \leftarrow u$

    $(s, gn) \leftarrow P^{i,m}_{dec}(s, gn)$

    $(\lambda, \beta) \leftarrow$ UCT-ROLLOUT$^{comp}_{\text{HGN}}(\mathcal{D}_{\text{HGN}}, s, gn, d, c)$

  for all $i \in UC(gn)$ **do**

    $Q_{\alpha(i)}(s, u) \leftarrow \dfrac{N_{\alpha(i)}(s, u) Q_{\alpha(i)}(s, u) + util(\lambda(i) + c) + K_g \mathbf{1}_{[\beta(i)]}}{1 + N_{\alpha(i)}(s, u)}$

    $N_{\alpha(i)}(s) \leftarrow N_{\alpha(i)}(s) + 1$

    $N_{\alpha(i)}(s, u) \leftarrow N_{\alpha(i)}(s, u) + 1$

  return (λ, β)

We implement this compressed approach in our algorithm, $\text{UCT}^{comp}_{\text{HGN}}$ (Algorithm 4.1), a UCT-based probabilistic planner that uses the rollout procedure in Algorithm 4.2. At each step, a fixed number of rollouts are performed (Lines 7–8). Using collected rollout statistics, the algorithm selects and applies the most promising node progression (Line 13), observes the resulting state, and repeats until the goal network is empty.

$\text{UCT}^{comp}_{\text{HGN}}$ introduces several key modifications to the standard UCT framework. First, it extends action selection to include both primitive actions and methods, reflecting the structure of action-method policies. At a node $n = (s, gn)$, the algorithm considers all actions applicable in s as well as all methods applicable in s that are relevant to some subgoal $i \in UC(gn)$. To capture this, the Q-function is extended to estimate values for both actions and methods:

- for actions a, $Q(s, a)$ is the estimated reward of executing a in state s, corresponding to the node progression P^a_{app}.
- for decompositions, $Q(s, (i, m))$ denotes the estimated reward of using method m to decompose subgoal i in state s, corresponding to the node progression $P^{i,m}_{dec}$.

Second, to optimize for multiple goals in the goal network, $\text{UCT}^{comp}_{\text{HGN}}$ learns a separate Q-function for each unconstrained subgoal. That is, for all $g \in \alpha[UC(gn)]$ (the image of $UC(gn)$ under α), $Q_g(s, u)$ estimates the expected reward of applying u in state s with respect to g. Here, u is either a primitive action $a \in A$, or a decomposition represented by a subgoal-method pair (i, m). 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 is in contrast to standard UCT planning, which learns a single Q-function aimed at the final planning goal.

To progress nodes, $\text{UCT}^{comp}_{\text{HGN}}$ chooses the action or method that maximizes the sum of these

Q-values (Algorithm 4.1 Line 13), jointly optimizing for multiple subgoals. During rollouts, UCB1-values are used instead of Q-values, where

$$\text{UCB1}(Q, N, s, u) = Q(s, u) + C\sqrt{\frac{\log(N(s))}{N(s,u)}} \tag{4.2}$$

and C is the exploration constant (Algorithm 4.2 Line 10). Our UCT algorithm uses random selection to break ties.

The compressed approach offers computational advantages over $\text{UCT}^{base}_{\text{HGN}}$ by using a smaller state space, but it compromises optimality. By maintaining separate Q-functions for each subgoal, $\text{UCT}^{comp}_{\text{HGN}}$ may fail to capture interdependencies among subgoals within a goal network. The selection process—which selects actions and methods based on Q-functions for each individual subgoal—implicitly assumes that subgoals can be optimized independently, treating them as separable objectives. However, this assumption breaks down in problems where the optimal strategy for achieving one subgoal depends on the configuration of other subgoals in the goal network. In such cases, the compressed representation may lead the algorithm toward suboptimal solutions, as it cannot reason about future, constrained subgoals in the full goal network. This efficiency-optimality trade-off implies that $\text{UCT}^{comp}_{\text{HGN}}$, unlike $\text{UCT}^{base}_{\text{HGN}}$, cannot guarantee convergence to an optimal policy even with unlimited compute.

4.5 Experiments and Results

We compared the performance of $\text{UCT}^{base}_{\text{HGN}}$ and $\text{UCT}^{comp}_{\text{HGN}}$ on a collection of benchmark domains adapted for probabilistic HGN planning. These domains were originally developed for FOND HTN

planning [86], so it was necessary to modify them to support the probabilistic action outcomes and hierarchical goal structures within our framework. First, nondeterministic actions—which in FOND planning specify multiple possible outcomes without associated probabilities—were converted into probabilistic actions by assigning uniform probabilities over the original outcome set. This allowed us to preserve the branching behavior of nondeterministic effects in a manner compatible with probabilistic planning. Second, we replaced each domain's HTN methods with a new set of HGN methods tailored to our formalism. While there is no direct one-to-one mapping from HTN to HGN methods, we manually constructed the HGN methods to preserve the structure and intent of the original decompositions wherever possible; when HTN methods decomposed tasks into specific sequences of primitive actions, the corresponding HGN methods were designed to produce equivalent primitive solutions.

Each domain consists of 15 problems. Every problem defines an initial state and an initial task network, which was manually translated into a corresponding initial goal network for use in our HGN planning framework. These domain adaptations preserve the essential planning challenges of the original benchmarks while enabling evaluation under the probabilistic HGN framework. Both UCT variants were tested on the same adapted domains and problems.

For each benchmark problem, we evaluated $\text{UCT}^{base}_{\text{HGN}}$ and $\text{UCT}^{comp}_{\text{HGN}}$ by running 20 independent trials per algorithm and averaging results (Figure 4.3). Both UCT variants commit to actions during search. Execution for both algorithms was bounded by 100 action executions per trial; runs that exceeded the 100 action budget were halted and contributed 100 to the average. During each trial, each planner performed 1000 rollouts at each decision step. Both UCT variants used a non-heuristic implementation in which all Q-values were initialized to 0, and a maximum rollout depth of 20

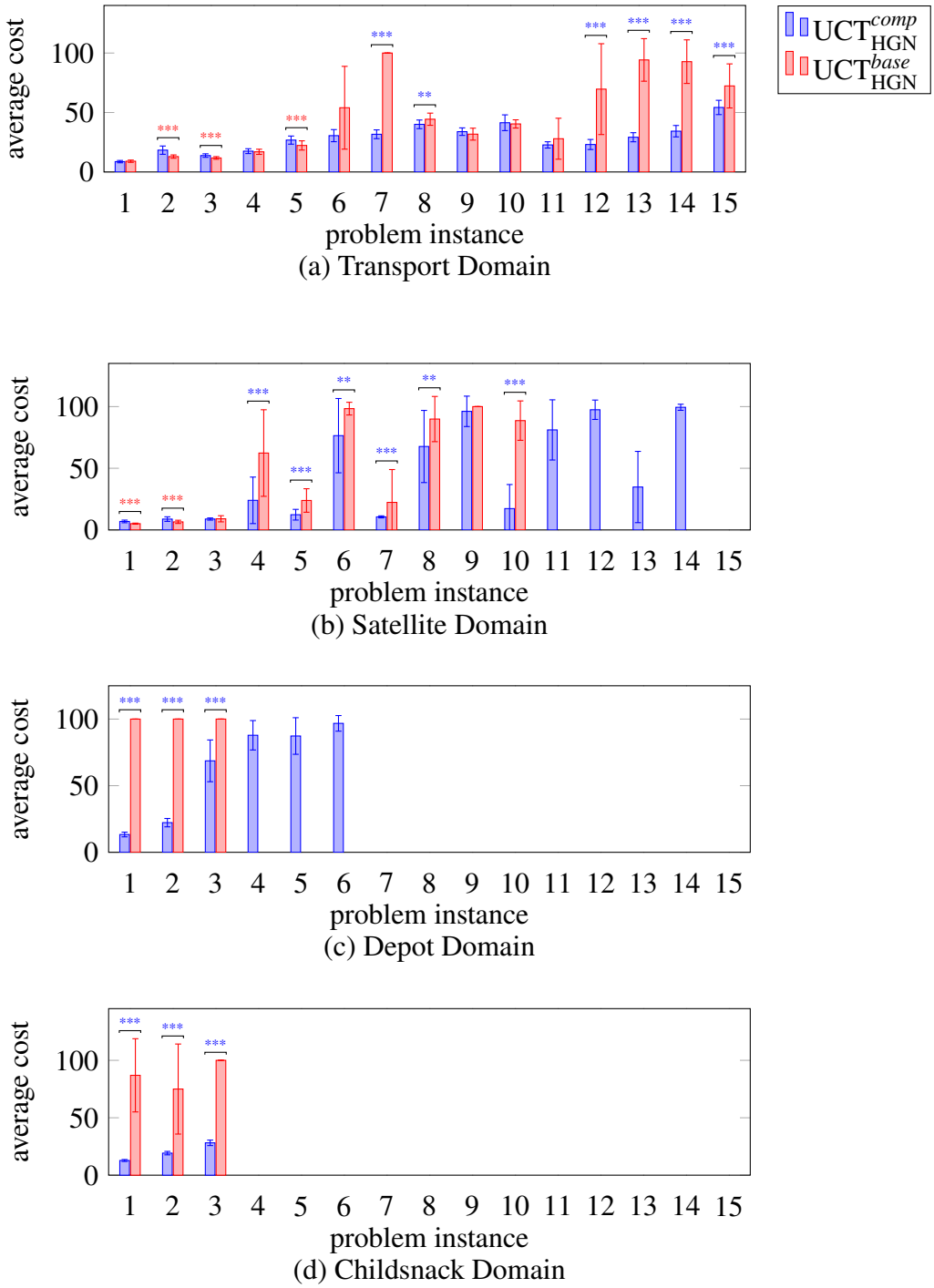


Figure 4.3: Average cost (lower is better) of solutions generated by $\text{UCT}^{comp}_{\text{HGN}}$ and $\text{UCT}^{base}_{\text{HGN}}$ on problem instances of each domain. Missing bars indicate runs did not finish due to memory constraints. Error bars show ±1 standard deviation. Statistical significance was determined using a two-sided Mann–Whitney U test, a non-parametric version of the t-test. ** indicates $p < 0.01$ and *** indicates $p < 0.001$; stars are colored according to the better variant.

was enforced. The UCT exploration constant was set to $\sqrt{2}$ following Kocsis and Szepesvári [47]. Planning was evaluated under the GUBS criterion, with the goal utility constant K_g set to 1 and the utility function defined as $util(cost) = \exp\left(-\frac{1}{10} \cdot cost\right)$, consistent with the default settings used by Crispino et al. [27]. All trials were independently run on compute nodes with 32 GB memory.

Our experimental results highlight key trade-offs between $\text{UCT}^{base}_{\text{HGN}}$ and $\text{UCT}^{comp}_{\text{HGN}}$ across problem sizes and domains. In smaller problems—such as Satellite problems 1–3 and Transport problems 1–5—both planning approaches consistently converged within the allotted budget of 1000 rollouts per decision step. In these cases, $\text{UCT}^{base}_{\text{HGN}}$ outperformed the compressed variant in terms of solution quality. This is expected: $\text{UCT}^{base}_{\text{HGN}}$ operates on the full SSP formulation, which guarantees convergence to an optimal solution given sufficient rollouts. For smaller problems, this optimality can often be achieved within the given computational budget.

However, for larger, more complex problems, $\text{UCT}^{base}_{\text{HGN}}$ struggled to converge within the computational limits. In particular, for problems with larger state spaces—such as Depot problems 4–6 and Satellite problems 11–14—the baseline planner often failed to return any solution due to exceeding memory constraints. In contrast, the compressed $\text{UCT}^{comp}_{\text{HGN}}$ variant was able to solve many of these larger problems within the same computational environment, demonstrating greater memory efficiency and broader scalability.

On these larger problems, $\text{UCT}^{comp}_{\text{HGN}}$ consistently outperformed $\text{UCT}^{base}_{\text{HGN}}$ in terms of average solution cost, and was the only variant that could solve the most memory-intensive instances. This behavior stems from the compressed state representation used by $\text{UCT}^{comp}_{\text{HGN}}$ and the ability to generalize across goal networks that share the same underlying world state. While this sacrifices UCT's asymptotic optimality guarantees—meaning it may converge to suboptimal policies even

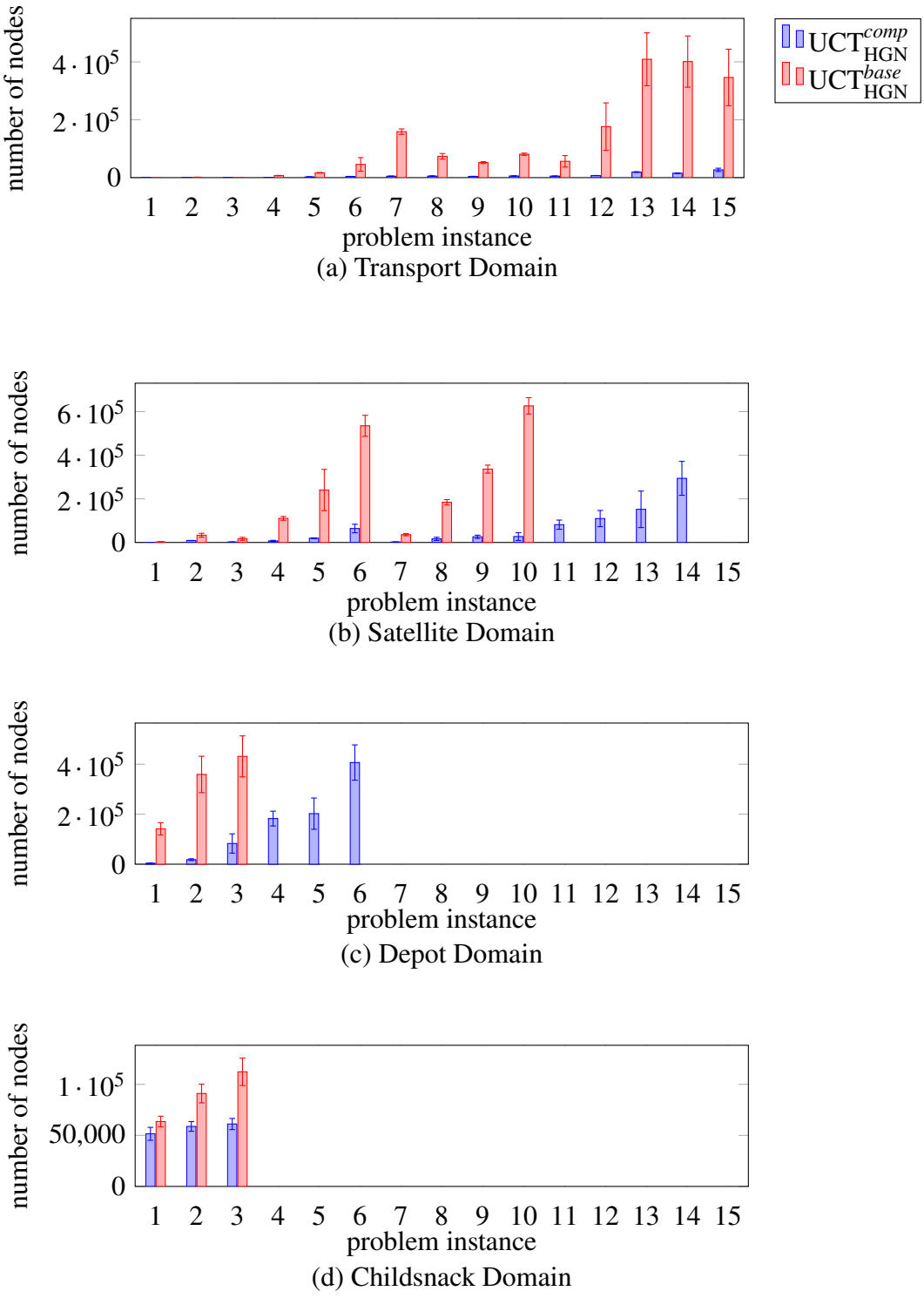


Figure 4.4: Average number of nodes in the search tree for $\text{UCT}^{base}_{\text{HGN}}$ and $\text{UCT}^{comp}_{\text{HGN}}$ on problem instances of each domain. Missing bars indicate runs did not finish due to memory constraints. Error bars show ±1 standard deviation.

with unlimited computation—it has significantly improved performance in compute-constrained settings.

To further understand the efficiency gains of the compressed variant, we analyzed the size of the search trees generated by both algorithms (Figure 4.4). Our measurements showed that $\text{UCT}^{base}_{\text{HGN}}$ consistently produced larger search trees, with node counts significantly exceeding those of the compressed $\text{UCT}^{comp}_{\text{HGN}}$ variant. However, we note that node count does not directly translate to memory footprint, as the two algorithms employ fundamentally different node structures. In $\text{UCT}^{base}_{\text{HGN}}$, each node represents a specific state–goal network pair and stores a single value estimate for the node. In contrast, each node in the compressed $\text{UCT}^{comp}_{\text{HGN}}$ tree represents an underlying world state and maintains value estimates for multiple goals simultaneously.

Overall, the compressed $\text{UCT}^{comp}_{\text{HGN}}$ variant demonstrates a favorable balance between solution quality and scalability. It retains comparable performance to $\text{UCT}^{base}_{\text{HGN}}$ on small problems, while offering considerable advantages on larger domains where memory and computation are limiting factors.

4.6 Summary and Future Work

We formalized probabilistic HGN planning to extend classical HGN planning to probabilistic domains and we evaluated two UCT planning algorithms: $\text{UCT}^{base}_{\text{HGN}}$, a baseline which transforms the probabilistic HGN problem into an SSP problem, preserving asymptotic optimality, and $\text{UCT}^{comp}_{\text{HGN}}$, a compressed variant of UCT that reduces the size of the search tree by sharing learned Q-values across similar goal network configurations within the same world state.

Experimental results on four adapted FOND HTN planning problems indicate that, although

$\text{UCT}^{base}_{\text{HGN}}$ performs well on small problems, it struggles to scale under limited rollout and memory budgets. By contrast, $\text{UCT}^{comp}_{\text{HGN}}$ consistently handles larger problems and produces lower-cost solutions within the same limited rollouts and memory. It achieves faster convergence and reduced memory usage, particularly in larger problems. These findings suggest that the compressed variant is well-suited for anytime planning, where quick, high-quality decisions are required under computational constraints. Moreover, its efficiency and low overhead make it a strong candidate for use as a heuristic or policy prior to guide more computationally intensive optimal planners.

The current implementation of $\text{UCT}^{comp}_{\text{HGN}}$ uses a greedy action selection strategy that focuses only on the immediate unconstrained subgoals in the goal network. While this design choice is more efficient, it can be myopic—leading to locally optimal behaviors that interfere with future subgoals, especially in the presence of misleading or unsafe methods. Future work could explore ways to incorporate broader goal network context into decision-making, potentially improving long-horizon performance. Additionally, although our focus in this work was not on incomplete domain models, the use of action insertion naturally supports planning with an incomplete set of HGN methods by allowing planners to interleave hierarchical decomposition with primitive action chaining. Further investigation could reveal how partial domain knowledge provided in the form of HGN methods could be effectively leveraged to guide search.

Chapter 5: Goal-Network-Directed Probabilistic Planning

The preceding chapters demonstrated that probabilistic planning can benefit substantially from structured goal decomposition. Both the LAMP algorithm (Chapter 3) and the $\text{UCT}^{comp}_{\text{HGN}}$ algorithm (Chapter 4) augment MCTS with a partially ordered collection of subgoals that guide exploration toward key intermediate states. This additional structure is particularly beneficial in large planning problems and anytime planning, where standard Monte Carlo search may struggle to find good solutions quickly.

Although LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ share this central principle, they differ fundamentally in how the goal structure is obtained and exploited. LAMP derives its guidance automatically through domain-independent landmark extraction, discovering subgoal structure from a deterministic relaxation of the problem. In contrast, $\text{UCT}^{comp}_{\text{HGN}}$ leverages manually authored hierarchical goal decomposition methods, encoding expert knowledge directly into the planning model. These differences induce important trade-offs in domain dependence, adaptability of the goal network, and strategies for balancing multiple objectives during search. Section 5.1 presents a comparative analysis that outlines the strengths and limitations of each approach.

Despite these distinctions, a deeper analysis reveals that both approaches share the same underlying components: a Monte Carlo search augmented with mechanisms for subgoal prioritization

and goal network management. Motivated by this shared foundation, Section 5.2 introduces a unified framework for goal-network-directed probabilistic planning. We present a generalized MCTS algorithm that abstracts these core mechanics and demonstrates how, by adjusting these mechanisms, the specific behaviors of both LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ can be recovered. This unified perspective not only clarifies the underlying relationship between these two algorithms but also provides an extensible blueprint for the future development of similar probabilistic planners that integrate with partially ordered goal structures.

5.1 Landmarks versus Hierarchical Goal Networks for Probabilistic Planning

Before analyzing the specific algorithmic trade-offs between LAMP and $\text{UCT}^{comp}_{\text{HGN}}$, it is important to understand the fundamental differences between the goal structures they employ: landmark graphs and hierarchical goal networks. Both are methods for decomposing complex planning problems into simpler subproblems, but they differ in two key aspects that inform the design of the algorithms that use them.

The first distinction lies in the origin of the structure. Landmarks are typically automatically generated by a domain-independent landmark extraction algorithm like LM^{RHW} [64]. This makes landmark-based guidance broadly applicable to new domains without requiring domain expertise. In contrast, HGN methods are generally manually authored by a human domain expert. This reliance on human expertise makes HGN method creation expensive but allows the planner to exploit any well-understood problem structures. These structural differences give rise to distinct algorithmic design choices in LAMP and $\text{UCT}^{comp}_{\text{HGN}}$.

A related distinction is the theoretical guarantees that each structure provides. Landmarks

Table 5.1: A summary of the trade-offs between LAMP and $\text{UCT}^{comp}_{\text{HGN}}$.

	LAMP	$\text{UCT}^{comp}_{\text{HGN}}$
Domain knowledge	**Domain-independent.** The landmark graph is generated algorithmically from the domain model without human input.	**Domain-dependent.** Requires a human expert to write HGN decomposition methods based on domain knowledge.
Goal network	**Static.** The landmark graph is generated once and does not change during planning.	**Dynamic.** The goal network actively expands and adapts during planning by invoking methods, allowing the structure to change based on the current state.
Strategy	**Balanced.** Uses a learned policy and a greediness parameter to explicitly balance focus between an immediate landmark and the final goal.	**Greedy.** Focuses exclusively on achieving the set of currently unconstrained subgoals, without explicit consideration for the final goal.

are, by definition, necessary conditions: each landmark must become true in every valid solution plan. In contrast, the subgoals in a hierarchical goal network are not required conditions for all solutions. Instead, they encode expert-provided knowledge that guides the planner toward preferred solutions. HGNs therefore offer greater flexibility, at the cost of depending heavily on the quality of the authored methods.

At their core, both LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ are algorithms for MCTS guided by a partially ordered goal network. In both algorithms, the planner maintains separate Q-value estimates associated with each subgoal, allowing action selection to be directed towards intermediate objectives rather than solely the final goal. This biases rollouts toward more immediate and tractable objectives. Despite this common foundation, the algorithms differ in three important respects, summarized in Table 5.1:

- **Domain Knowledge vs. Domain Independence.** $\text{UCT}^{comp}_{\text{HGN}}$ is inherently domain-dependent.

It relies on manually authored HGN methods to define the decomposition structure. This enables direct encoding of expert knowledge and known problem structures. For domains where efficient solution strategies are well-understood (e.g., manufacturing, logistics, or game-playing with established strategies), such curated knowledge can be very effective at guiding search towards preferred or high-quality solutions.

LAMP, in contrast, is domain-independent. It automatically discovers its goal structure by applying a classical landmark extraction algorithm (e.g., LM^{RHW}) to the deterministic relaxation of the planning problem. This automated discovery makes LAMP well-suited for new domains where problem structure is unknown or for which domain expertise is unavailable. However, the extracted landmark structure may not capture the most efficient strategy an expert might provide.

- **Static vs. Dynamic Goal Networks.** In LAMP, the landmark graph is generated once at the beginning of the planning episode. As landmarks are completed, they are removed, but no new subgoals can be added.

  In $\text{UCT}^{comp}_{\text{HGN}}$, the goal network is dynamic. The planner may invoke HGN methods during search to refine unconstrained subgoals into more detailed goal networks. This refinement can depend on the current state, allowing the decomposition to adapt to the outcomes of previously executed actions. Consequently, $\text{UCT}^{comp}_{\text{HGN}}$ must explore both actions and HGN methods, increasing representational power but also enlarging the search space.

- **Multi-Objective Optimization Strategy.** The standard $\text{UCT}^{comp}_{\text{HGN}}$ algorithm adopts a greedy subgoal prioritization strategy. At each decision point, it considers only the set of uncon-

strained subgoals—those with no remaining predecessors in the goal network—and selects actions according to $\sum_{i \in UC(gn)} Q_{\alpha(i)}(s, a)$. Action evaluation is restricted to the currently unconstrained subgoals, without explicitly accounting for downstream effects on future subgoals. This approach simplifies the decision-making process significantly but can be shortsighted, as it does not explicitly consider how achieving an immediate subgoal might affect the cost of achieving future subgoals.

LAMP employs a more nuanced approach. It learns a separate landmark-value function, Q_{LM}, over landmarks in the landmark graph to determine which landmark, φ, to pursue next. It also introduces a greediness parameter, θ, to explicitly balance its priority between short-term and long-term objectives. During action selection, it evaluates an action a using a weighted combination, parameterized by θ, of its value with respect to the chosen landmark $Q_\varphi(s, a)$ and its value with respect to the final goal $Q_g(s, a)$. This mechanism balances short-term subgoal achievement against long-term goals, reducing the risk of purely myopic decisions.

The choice between leveraging landmarks or hierarchical goal networks for probabilistic planning boils down to a fundamental trade-off between automatic, domain-independent landmark extraction and more curated, domain-specific expertise. This distinction influences their respective strategies. Landmarks may require careful balancing to avoid shortsighted behavior while HGN guidance offers richer, strategy-level control but depends heavily on the quality and completeness of the authored methods. Ultimately, LAMP provides a strong baseline for unknown domains, while $\text{UCT}^{comp}_{\text{HGN}}$ offers a mechanism to inject human-curated domain knowledge into well-understood problems.

5.2 A Unified Framework

Despite their differences in origin and optimization, LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ are distinct instantiations of the same fundamental idea: using a goal network to guide a probabilistic planner. This observation motivates a generalized framework for goal-network-directed probabilistic planning.

To do this, we adopt the probabilistic HGN planning formalism introduced in Chapter 4. Recall that a probabilistic HGN planning problem is a tuple

$$\mathcal{P}_{\text{HGN}} = (\mathcal{D}_{\text{HGN}}, s_0, gn_0) \tag{5.1}$$

where $\mathcal{D}_{\text{HGN}} = (\mathcal{L}, M, A, \gamma, \Pr)$ is a probabilistic HGN domain, s_0 is the initial state, and gn_0 is the initial goal network. Importantly, the search space in this framework includes not only the world state s, but also the current goal network gn. Planning thus proceeds in a joint space of state, goal network pairs (s, gn).

5.2.1 Goal Network Expansion

Within the HGN formalism, goal networks evolve through the application of methods that decompose goals into partially ordered sets of subgoals. This mechanism is sufficiently expressive to capture both dynamic and static goal-network behaviors. In full probabilistic HGN planning, methods in M are applied during search to expand and refine the goal network in a state-dependent manner. The planner may therefore interleave method application and action execution, dynamically shaping the goal network based on observed action outcomes.

The behavior of LAMP can also be modeled within this same formalism. In this case, the set of

methods M is empty, and the initial goal network gn_0 is the landmark graph produced by the LM^{RHW} landmark extraction algorithm. Because no decomposition methods are available, the goal network cannot expand; it can only shrink as achieved landmarks are removed. The static landmark-guided structure of LAMP therefore emerges as a special case of probabilistic HGN planning where $M = \varnothing$.

To present a clean, generalized algorithm (Algorithm 5.1), we abstract the goal network management into a single black-box procedure, UpdateGoalNetwork (Line 7), which is responsible for modifying gn prior to action selection.

For $\text{UCT}^{comp}_{\text{HGN}}$, UpdateGoalNetwork encapsulates a UCT-based decision process over both actions and methods. Since $\text{UCT}^{comp}_{\text{HGN}}$ maintains Q-value estimates for both actions and decomposition methods, this procedure compares their respective values and selects the highest-valued option. If the highest Q-value is assigned to a method, that method is invoked to decompose the goal network gn before the algorithm proceeds with action selection. During rollouts (Algorithm 5.2), UCB1 values are used instead of Q-values.

For LAMP, UpdateGoalNetwork is a trivial no-operation—since the set of methods, M, is empty, there are no decompositions to consider, and the function returns without altering the goal network.

5.2.2 Subgoal Prioritization

With goal-network dynamics unified, the remaining question is how the planner selects actions in the presence of multiple concurrent subgoals. Both LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ extend standard UCT by replacing a single objective with a structured, multi-objective formulation defined over the current goal network. Instead of learning a single action-value function for the final goal, the planner maintains a separate value function for each subgoal g in the current goal network gn. For each g,

Algorithm 5.1 (GN-UCT) The general form of a goal-network directed UCT algorithm for probabilistic planning. $\mathcal{D}_{\text{HGN}}$ is a probabilistic planning domain, s is the current state, gn is the current goal network, n_{ro} is the number of rollouts performed at each planning step, d is the maximum rollout depth, and c is the cumulative cost (initially 0).

1: **procedure** GN-UCT($\mathcal{D}_{\text{HGN}}, s, gn, n_{ro}, d, c$)
2: **if** $gn = gn_{\varnothing}$ **then**
3: **return** success
4: **if** $\exists i \in UC(gn)$ such that $s \models \alpha(i)$ **then**
5: $gn \leftarrow (I \backslash \{i\}, \{(i_1, i_2) \in \prec \mid i_1 \neq i\}, \{(i', g) \in \alpha \mid i' \neq i\})$ ▷ remove i from gn
6: **return** GN-UCT($\mathcal{D}_{\text{HGN}}, s, gn, n_{ro}, d, c$)
7: $gn \leftarrow$ UpdateGoalNetwork
8: $w \leftarrow$ PriorityFunction
9: **for** n_{ro} times **do** ▷ learn Q-functions
10: GN-UCT-Rollout($\mathcal{D}_{\text{HGN}}, s, gn, d, c$)
11: $U \leftarrow \{a \in A \mid a \text{ is applicable in } s\}$
12: **if** $U = \varnothing$ **then**
13: **return** failure
14: $u \leftarrow \arg\max_{u \in U} \sum_{i \in I} w(i) Q_{\alpha(i)}(s, u)$
15: $s' \leftarrow \text{apply}(s, u)$ ▷ execute u and observe the state
16: **return** GN-UCT($\mathcal{D}_{\text{HGN}}, s', gn, n_{ro}, d, c + 1$)

Algorithm 5.2 (GN-UCT-ROLLOUT) The Monte Carlo rollout procedure for GN-UCT. $\mathcal{D}_{\text{HGN}}$, s, gn, and c are as defined in Algorithm 5.1. d is the remaining rollout depth, *util* is a strictly decreasing utility function over cost, and K_g is the goal-utility constant. Q_* and N_* are global maps with default value 0. GN-UCT-ROLLOUT returns a pair of maps $\lambda : I \to \mathbb{R}^{\geq 0}$ and $\beta : I \to \{\mathsf{true}, \mathsf{false}\}$ where $\lambda(i)$ is the rollout cost for subgoal i, and $\beta(i)$ indicates whether subgoal i was achieved.

procedure GN-UCT-ROLLOUT($\mathcal{D}_{\text{HGN}}, s, gn, d, c$)
  if $gn = gn_{\varnothing}$ **then**
    return $(\varnothing, \varnothing)$
  if $\exists i \in UC(gn)$ such that $s \models \alpha(i)$ **then**
    $gn \leftarrow (I \backslash \{i\}, \{(i_1, i_2) \in \prec \mid i_1 \neq i\}, \{(i', g) \in \alpha \mid i' \neq i\})$ ▷ remove i from gn
    $(\lambda, \beta) \leftarrow$ GN-UCT-ROLLOUT($\mathcal{D}_{\text{HGN}}, s, gn, d, c$)
    return $(\lambda \cup \{(i, 0)\}, \beta \cup \{(i, \mathsf{true})\})$
  $gn \leftarrow$ UPDATEGOALNETWORK ▷ $gn = (I, \prec, \alpha)$
  $w \leftarrow$ PRIORITYFUNCTION
  $U \leftarrow \{a \in A \mid a \text{ is applicable in } s\}$
  if $d = 0$ or $U = \varnothing$ **then**
    return $(I \times \{d\}, I \times \{\mathsf{false}\})$
  $a \leftarrow \arg\max_{a \in U} \sum_{i \in I'} w(i) \text{UCB1}(Q_{\alpha(i)}, N_{\alpha(i)}, s, a)$
  sample $s' \sim \Pr(\cdot \mid s, a)$
  $(\lambda, \beta) \leftarrow$ GN-UCT-ROLLOUT($\mathcal{D}_{\text{HGN}}, s', gn, d - 1, c + 1$)
  $\lambda \leftarrow \{(i, \lambda(i) + 1) \mid i \in I\}$
  for all $i \in \text{supp}(w)$ **do**
    $Q_{\alpha(i)}(s, u) \leftarrow \frac{N_{\alpha(i)}(s,u) Q_{\alpha(i)}(s,u) + util(\lambda(i)+c) + K_g \mathbf{1}_{[\beta(i)]}}{1 + N_{\alpha(i)}(s,u)}$
    $N_{\alpha(i)}(s) \leftarrow N_{\alpha(i)}(s) + 1$
    $N_{\alpha(i)}(s, u) \leftarrow N_{\alpha(i)}(s, u) + 1$
  return (λ, β)

the function $Q_g(s,a)$ estimates the expected utility of executing action a in state s with respect to achieving subgoal g. Correspondingly, separate visit statistics $N_g(s)$ and $N_g(s,a)$ are maintained for each subgoal.

Suppose $gn = (I, \prec, \alpha)$ is the current goal network, where I indexes the goal network and $\alpha(i)$ maps each index to its corresponding goal formula. Action selection is performed by aggregating the subgoal-specific value estimates using a priority weight function $w : I \to \mathbb{R}_{\geq 0}$. Given state s, the selected action is

$$a^* = \underset{a \in \text{Applicable}(s)}{\arg\max} \sum_{i \in I} w(i) Q_{\alpha(i)}(s,a). \tag{5.2}$$

During UCT rollouts, this aggregation is applied to UCB1 scores:

$$a^* = \underset{a \in \text{Applicable}(s)}{\arg\max} \sum_{i \in I} w(i) \text{UCB1}(Q_{\alpha(i)}, N_{\alpha(i)}, s, a) \tag{5.3}$$

$$= \underset{a \in \text{Applicable}(s)}{\arg\max} \sum_{i \in I} w(i) \left(Q_{\alpha(i)}(s,a) + C \sqrt{\frac{\log(N_{\alpha(i)}(s))}{N_{\alpha(i)}(s,a)}} \right) \tag{5.4}$$

where $C > 0$ is the exploration constant. Thus, action selection becomes a weighted multi-objective optimization over subgoals, balancing exploitation and exploration independently for each objective according to the priority weights w.

To present the generalized algorithm (Algorithm 5.1), we abstract this subgoal prioritization logic into a black-box procedure, PriorityFunction (Line 8). This procedure returns the weight function w that determines how subgoal-specific value estimates are aggregated during action selection. The specific implementation of PriorityFunction therefore defines the planner's prioritization strategy.

In $\text{UCT}^{comp}_{\text{HGN}}$, PriorityFunction implements a simple greedy strategy. The priority function w

assigns a uniform positive weight to all unconstrained subgoals and a weight of zero to all others, forcing the planner to focus exclusively on the set of immediate next subgoals.

$$w(i) = \mathbf{1}_{UC(gn)}(i) \tag{5.5}$$

In LAMP, PriorityFunction operates in two stages. First, a UCT-based policy selects a single target landmark, φ, from the unconstrained set of landmarks according to the landmark-value function Q_{LM}. Then, using the greediness parameter, θ, the priority function w assigns a nonzero weight to at most two subgoals: it assigns a weight of θ to the chosen landmark ($w(i) = \theta$ where $\alpha(i) = \varphi$) and a weight of $1 - \theta$ to the final goal of the entire problem ($w(j) = 1 - \theta$ where $\alpha(j) = g$).

$$w(i) = \begin{cases} \theta & \text{if } \alpha(i) = \varphi, \\ 1 - \theta & \text{if } \alpha(i) = g, \\ 0 & \text{otherwise,} \end{cases} \tag{5.6}$$

This formulation explicitly trades off progress towards an immediate subgoal with progress towards the final goal using the greediness parameter θ.

After each rollout, only the subgoal-specific value functions corresponding to subgoals with nonzero priority weight are updated. That is, for each i such that $w(i) > 0$, the visit counts $N_{\alpha(i)}(s)$, $N_{\alpha(i)}(s,a)$, and the action-value estimates $Q_{\alpha(i)}(s,a)$ are updated using the observed return. Subgoals assigned zero weight do not receive updates, as they did not influence action selection.

This generalized subgoal prioritization captures both LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ as specific choices of

the weight function w. The distinction between the two algorithms therefore lies not in the underlying search architecture, but in the prioritization strategy imposed over a common multi-objective MCTS framework. This abstraction provides a general design template for future goal-directed probabilistic planners, where behaviors can be adjusted by tuning the subgoal priority function w.

5.3 Future Work

The unified framework for goal-network-directed probabilistic planning reveals a broader design space beyond the specific instantiations of LAMP and $\text{UCT}^{comp}_{\text{HGN}}$. By separating out goal-network management and subgoal prioritization, the framework exposes two dimensions along which new planning algorithms may be developed.

Both LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ rely on UCT not only for action selection, but also to guide higher-level decisions within the planner. In LAMP, a UCT policy is used to select which landmark to pursue next, and in $\text{UCT}^{comp}_{\text{HGN}}$, UCT is used to select among decomposition methods, enabling dynamic expansion of the goal network. While this reuse of UCT is simple and conceptually consistent, there is opportunity to explore alternative strategies for these components. More sophisticated subgoal prioritization strategies could interpolate between greedy and non-greedy approaches, potentially mitigating some myopic behaviors observed in LAMP while preserving heuristic speedups. Additionally, hybrid approaches that combine automatically extracted landmarks with HGN methods may offer a practical middle ground between domain-independent and manually specified domain knowledge. Learning-based techniques from classical planning, such as those that derive hierarchical structure from landmarks [49], could be extended to probabilistic planning algorithms as well.

The unified formulation also raises theoretical questions. While standard MCTS is asymptotically optimal, the heuristic approaches of goal-network-directed UCT generally trade this optimality in favor of improved practical performance in anytime planning settings. A deeper analysis of convergence properties under different subgoal prioritization strategies would strengthen the theoretical foundations of the approach and could clarify the conditions under which optimality guarantees are preserved.

Chapter 6: Conclusion

This dissertation extended goal decomposition techniques from classical planning to probabilistic planning in order to improve scalability and performance in nondeterministic domains. Classical planning has long benefited from structured goal representations such as landmark graphs and hierarchical goal networks. In contrast, many probabilistic planners, including those based on MCTS, reason directly over the stochastic state space without incorporating information on subgoals. This work bridges that gap by integrating structured goal decomposition with probabilistic planning.

In Chapter 3, we generalized classical landmarks to probabilistic settings and introduced LAMP, a domain-independent probabilistic planner that leverages probabilistic landmarks as intermediate objectives. By decomposing the planning task into a sequence of landmark-directed subproblems, LAMP significantly improves performance over standard UCT in several benchmark domains. We analyzed the trade-off between greedy and non-greedy landmark achievement, showing that while pure UCT is asymptotically optimal, greedy landmark-driven strategies often provide substantial empirical improvements in large problems or time-constrained anytime planning. At the same time, we identified limitations of greedy approaches, particularly in domains where landmarks may include unsafe or dead-end states.

In Chapter 4, we formalized the use of hierarchical goal networks in probabilistic planning.

We developed two UCT-based algorithms for probabilistic HGN planning: a baseline method that searches the augmented state space of nodes (s, gn) as a stochastic shortest path problem, and $\text{UCT}^{comp}_{\text{HGN}}$, which compresses the search tree by sharing value estimates across related goal networks. Experimental results demonstrated that while the baseline approach performs adequately on small problem instances, the compressed variant scales more effectively, highlighting the advantages of explicitly reasoning over hierarchical goal structure in probabilistic planning.

In Chapter 5, we demonstrated that LAMP and $\text{UCT}^{comp}_{\text{HGN}}$ are not isolated techniques, but specific instantiations of a more general architecture of goal-network-directed probabilistic planning. By abstracting goal-network expansion and subgoal prioritization as separate components, we introduced a unified framework that captures the behaviors of both algorithms. This perspective clarifies the relationship between the use of automatically extracted landmarks and manually authored hierarchical decomposition methods, revealing a broader design space for probabilistic planning over goal networks.

The primary contributions of this dissertation include:

- formalizing probabilistic landmarks;
- integrating probabilistic landmarks into UCT through the LAMP algorithm and conducting empirical evaluations demonstrating significant improvements over a standard UCT baseline in several domains;
- formalizing probabilistic HGN planning with action-insertion semantics;
- developing two scalable UCT-based algorithms—$\text{UCT}^{base}_{\text{HGN}}$ and $\text{UCT}^{comp}_{\text{HGN}}$—and conducting empirical evaluations demonstrating a scalability-optimality trade-off for probabilistic HGN

planning with UCT; and

- developing a unified framework for goal-network-directed probabilistic planning that abstracts goal network expansion and subgoal prioritization as modular components.

More broadly, this work connects goal decomposition with probabilistic sequential decision-making. It demonstrates that structured goal decomposition substantially speeds up MCTS and improves scalability in online and anytime settings, serving as a powerful bias for sampling-based planners. By integrating these classical planning structures with Monte Carlo search, the work advances the development of scalable, goal-directed planning systems capable of operating effectively in large, nondeterministic domains.

As planning domains continue to grow in complexity, flat planning representations become increasingly insufficient. The results presented here suggest that incorporating structured knowledge of subgoals—whether automatically extracted or manually specified—provides a practical means of guiding search in such domains. The unified framework introduced in this work establishes a foundation for future research on goal-network-directed probabilistic planning.

Appendices

Appendix A: LAMP Simple Benchmark Results

This appendix contains additional results from LAMP experiments. In each of the following figures, the top table reports the average cost (lower is better) of the solutions generated by LAMP over 75 runs. Underlined values indicate the best performing θ-value in the row. The bottom table reports p-values comparing the average cost of solutions generated by LAMP with the average cost of solutions generated by standard UCT ($\theta = 0$) at the same number of rollouts. Significance was determined using a two-sided Mann–Whitney U test, a non-parametric version of the t-test. In both tables, boldfaced values indicate where LAMP significantly dominated standard UCT at the same number of rollouts, with $p < 0.0125$, the adjusted level using a Bonferroni adjustment of $\alpha_{stat} = 0.05/4$.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT $\bar{x}$	0/UCT σ	0.2 $\bar{x}$	0.2 σ	0.5 $\bar{x}$	0.5 σ	0.8 $\bar{x}$	0.8 σ	1 $\bar{x}$	1 σ
5	190.1	32.3	**162.4**	62.7	**167.5**	58.9	180.8	45.8	173.8	52.9
10	184.0	41.5	**145.5**	65.7	**142.1**	70.5	**152.2**	67.7	167.3	59.4
20	175.0	51.2	**120.8**	68.6	**114.8**	75.6	**118.3**	70.4	**127.6**	70.9
50	157.8	58.9	**69.1**	57.0	**70.1**	58.2	**86.5**	70.6	**87.8**	65.6
100	145.4	59.9	**41.0**	33.4	**51.1**	38.2	**53.1**	47.6	**61.8**	55.5
200	110.9	58.4	**32.3**	15.1	**40.3**	40.6	**33.4**	21.6	**36.9**	27.2
500	65.9	40.7	**28.4**	18.0	**27.5**	9.1	**32.1**	23.6	**28.3**	10.8
1000	44.3	19.5	**28.8**	19.0	**25.7**	7.8	**27.5**	10.3	**26.1**	5.2
2000	39.6	14.5	**24.3**	6.1	**25.3**	5.0	**26.7**	7.1	**26.2**	5.2
5000	32.6	10.0	**23.2**	4.0	**24.1**	4.3	**23.6**	4.4	**25.1**	4.4

Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	**2.62e-03**	**5.84e-03**	2.29e-01	4.16e-02
N/A	**1.12e-05**	**3.07e-05**	**3.18e-04**	1.31e-02
N/A	**3.18e-07**	**2.89e-07**	**1.04e-07**	**1.15e-05**
N/A	**9.44e-14**	**4.88e-13**	**5.61e-09**	**2.13e-09**
N/A	**2.65e-21**	**3.19e-18**	**2.95e-17**	**1.08e-14**
N/A	**3.14e-20**	**3.71e-17**	**8.25e-20**	**3.82e-18**
N/A	**9.68e-18**	**1.07e-17**	**8.90e-15**	**4.89e-17**
N/A	**1.34e-11**	**1.17e-13**	**1.21e-11**	**3.31e-13**
N/A	**1.31e-14**	**4.52e-13**	**2.55e-10**	**3.42e-11**
N/A	**1.03e-10**	**5.94e-09**	**4.54e-10**	**3.97e-07**

p-values

Figure A.1(a): `prob_blocksworld` problem `p1`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT $\bar{x}$	0/UCT σ	0.2 $\bar{x}$	0.2 σ	0.5 $\bar{x}$	0.5 σ	0.8 $\bar{x}$	0.8 σ	1 $\bar{x}$	1 σ
5	193.3	23.3	**124.9**	77.7	**161.0**	64.3	**145.5**	70.4	**138.9**	73.1
10	188.3	34.5	**99.1**	76.6	**111.7**	79.1	**138.5**	72.8	**118.6**	78.8
20	170.1	55.4	**85.9**	68.4	**107.6**	77.4	**107.7**	76.2	**118.8**	77.5
50	133.4	68.7	**52.3**	45.5	**68.9**	62.9	**71.1**	63.3	**72.6**	66.3
100	112.9	65.6	**34.3**	25.2	**37.6**	28.0	**46.2**	43.2	**47.8**	42.4
200	74.0	51.8	**30.5**	23.9	**29.1**	14.8	**33.3**	21.6	**31.5**	24.1
500	41.2	20.0	**24.0**	10.0	**24.9**	7.9	**26.2**	11.5	**29.8**	17.5
1000	28.0	10.0	**20.4**	4.2	**23.8**	4.5	25.0	6.1	26.9	10.3
2000	23.5	7.9	**19.8**	5.2	21.5	4.3	22.2	4.1	24.4	4.8
5000	22.4	7.9	**18.1**	5.5	21.4	4.4	23.0	5.1	24.6	5.7

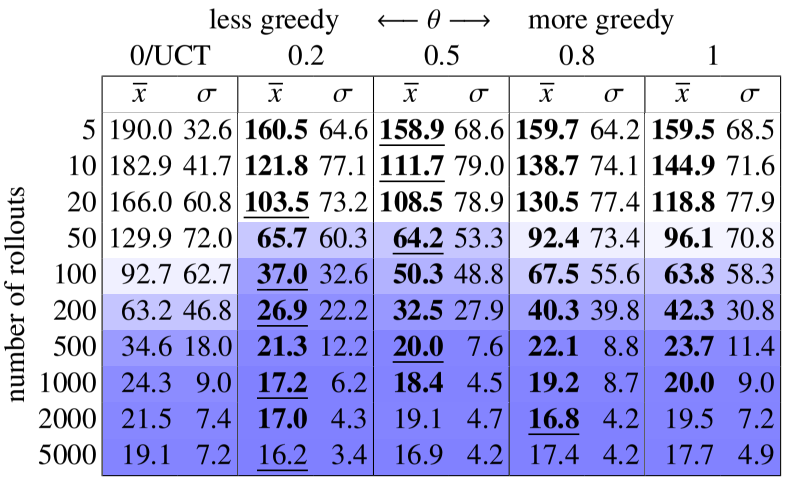
Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	**8.56e-10**	**4.16e-04**	**8.65e-07**	**1.28e-07**
N/A	**3.73e-13**	**7.06e-10**	**6.93e-06**	**6.89e-09**
N/A	**3.62e-12**	**5.12e-07**	**2.88e-07**	**5.97e-05**
N/A	**2.01e-13**	**9.39e-10**	**3.04e-09**	**6.19e-09**
N/A	**2.40e-16**	**3.80e-15**	**2.77e-12**	**2.03e-11**
N/A	**1.61e-14**	**8.54e-16**	**9.38e-13**	**9.61e-15**
N/A	**1.52e-11**	**1.72e-10**	**2.00e-09**	**1.29e-06**
N/A	**2.99e-09**	**9.07e-03**	1.08e-01	4.11e-01
N/A	**1.70e-03**	3.77e-01	8.77e-01	3.96e-02
N/A	**1.25e-04**	9.95e-01	1.54e-01	5.85e-03

p-values

Figure A.1(b): `prob_blocksworld` problem `p2`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.



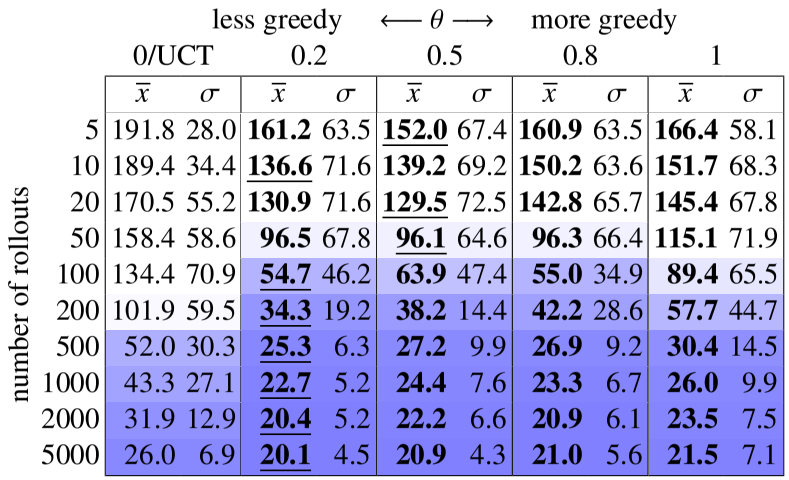
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	190.0	32.6	**160.5**	64.6	**158.9**	68.6	**159.7**	64.2	**159.5**	68.5
10	182.9	41.7	**121.8**	77.1	**111.7**	79.0	**138.7**	74.1	**144.9**	71.6
20	166.0	60.8	**103.5**	73.2	**108.5**	78.9	**130.5**	77.4	**118.8**	77.9
50	129.9	72.0	**65.7**	60.3	**64.2**	53.3	**92.4**	73.4	**96.1**	70.8
100	92.7	62.7	**37.0**	32.6	**50.3**	48.8	**67.5**	55.6	**63.8**	58.3
200	63.2	46.8	**26.9**	22.2	**32.5**	27.9	**40.3**	39.8	**42.3**	30.8
500	34.6	18.0	**21.3**	12.2	**20.0**	7.6	**22.1**	8.8	**23.7**	11.4
1000	24.3	9.0	**17.2**	6.2	**18.4**	4.5	**19.2**	8.7	**20.0**	9.0
2000	21.5	7.4	**17.0**	4.3	19.1	4.7	**16.8**	4.2	19.5	7.2
5000	19.1	7.2	16.2	3.4	16.9	4.2	17.4	4.2	17.7	4.9

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	**1.04e-03**	**9.15e-04**	**4.20e-04**	**1.06e-03**
N/A	**1.53e-06**	**1.88e-08**	**2.13e-04**	**1.60e-03**
N/A	**2.22e-07**	**6.74e-06**	**2.61e-03**	**1.87e-04**
N/A	**2.99e-08**	**2.79e-07**	**6.34e-04**	**3.30e-03**
N/A	**8.58e-12**	**2.57e-07**	**2.21e-03**	**2.45e-04**
N/A	**6.89e-12**	**2.99e-08**	**1.01e-05**	**1.60e-03**
N/A	**3.93e-08**	**6.36e-09**	**2.75e-06**	**4.25e-05**
N/A	**3.47e-08**	**1.91e-05**	**4.25e-05**	**1.86e-04**
N/A	**1.99e-04**	1.17e-01	**5.06e-05**	5.64e-02
N/A	2.85e-02	1.59e-01	4.19e-01	4.56e-01

p-values

Figure A.1(c): `prob_blocksworld` problem `p3`. The landmark extraction algorithm LM^{RHW} generated 12 nontrivial landmarks for this problem.



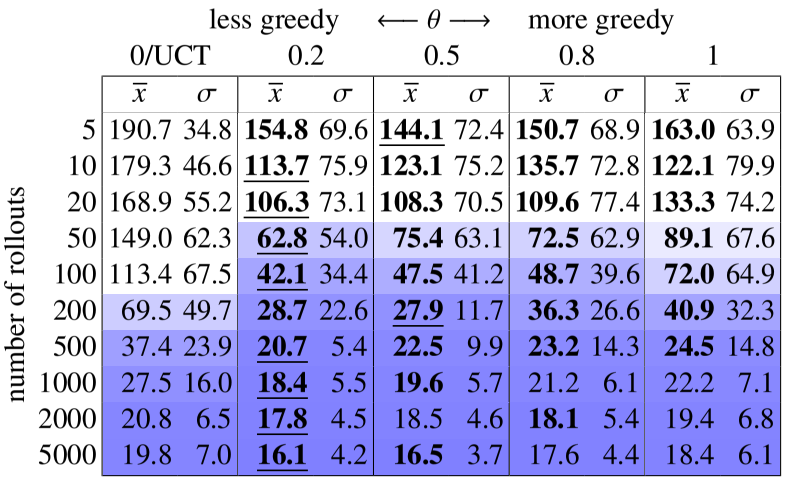
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	191.8	28.0	**161.2**	63.5	**152.0**	67.4	**160.9**	63.5	**166.4**	58.1
10	189.4	34.4	**136.6**	71.6	**139.2**	69.2	**150.2**	63.6	**151.7**	68.3
20	170.5	55.2	**130.9**	71.6	**129.5**	72.5	**142.8**	65.7	**145.4**	67.8
50	158.4	58.6	**96.5**	67.8	**96.1**	64.6	**96.3**	66.4	**115.1**	71.9
100	134.4	70.9	**54.7**	46.2	**63.9**	47.4	**55.0**	34.9	**89.4**	65.5
200	101.9	59.5	**34.3**	19.2	**38.2**	14.4	**42.2**	28.6	**57.7**	44.7
500	52.0	30.3	**25.3**	6.3	**27.2**	9.9	**26.9**	9.2	**30.4**	14.5
1000	43.3	27.1	**22.7**	5.2	**24.4**	7.6	**23.3**	6.7	**26.0**	9.9
2000	31.9	12.9	**20.4**	5.2	**22.2**	6.6	**20.9**	6.1	**23.5**	7.5
5000	26.0	6.9	**20.1**	4.5	**20.9**	4.3	**21.0**	5.6	**21.5**	7.1

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	**3.23e-04**	**1.79e-05**	**8.54e-04**	**7.00e-04**
N/A	**9.79e-08**	**4.98e-07**	**1.47e-06**	**6.21e-05**
N/A	**3.42e-04**	**1.88e-04**	**4.36e-03**	**9.28e-03**
N/A	**3.59e-08**	**1.09e-08**	**7.10e-08**	**4.71e-05**
N/A	**2.46e-10**	**1.08e-08**	**3.61e-10**	**4.10e-04**
N/A	**1.20e-16**	**8.12e-14**	**7.91e-13**	**2.35e-07**
N/A	**4.43e-13**	**6.99e-11**	**2.13e-11**	**2.42e-08**
N/A	**1.72e-12**	**3.07e-10**	**7.06e-12**	**2.18e-08**
N/A	**9.79e-11**	**3.52e-08**	**2.68e-10**	**5.42e-06**
N/A	**1.24e-08**	**1.64e-06**	**7.43e-07**	**7.20e-06**

p-values

Figure A.1(d): `prob_blocksworld` problem `p4`. The landmark extraction algorithm LM^{RHW} generated 12 nontrivial landmarks for this problem.



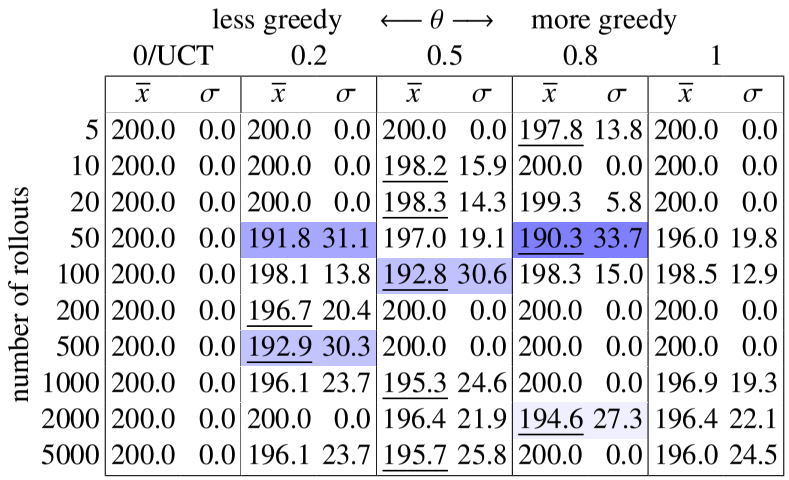
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	190.7	34.8	**154.8**	69.6	**144.1**	72.4	**150.7**	68.9	**163.0**	63.9
10	179.3	46.6	**113.7**	75.9	**123.1**	75.2	**135.7**	72.8	**122.1**	79.9
20	168.9	55.2	**106.3**	73.1	**108.3**	70.5	**109.6**	77.4	**133.3**	74.2
50	149.0	62.3	**62.8**	54.0	**75.4**	63.1	**72.5**	62.9	**89.1**	67.6
100	113.4	67.5	**42.1**	34.4	**47.5**	41.2	**48.7**	39.6	**72.0**	64.9
200	69.5	49.7	**28.7**	22.6	**27.9**	11.7	**36.3**	26.6	**40.9**	32.3
500	37.4	23.9	**20.7**	5.4	**22.5**	9.9	**23.2**	14.3	**24.5**	14.8
1000	27.5	16.0	**18.4**	5.5	**19.6**	5.7	21.2	6.1	22.2	7.1
2000	20.8	6.5	**17.8**	4.5	18.5	4.6	**18.1**	5.4	19.4	6.8
5000	19.8	7.0	**16.1**	4.2	**16.5**	3.7	17.6	4.4	18.4	6.1

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	**3.43e-04**	**6.85e-06**	**1.03e-05**	**2.40e-03**
N/A	**1.73e-07**	**4.31e-06**	**3.19e-04**	**3.83e-05**
N/A	**7.45e-08**	**3.85e-08**	**1.23e-06**	**8.17e-04**
N/A	**7.08e-14**	**9.26e-11**	**3.48e-11**	**3.87e-07**
N/A	**1.81e-11**	**2.91e-10**	**7.70e-10**	**3.09e-05**
N/A	**5.00e-12**	**2.95e-11**	**1.75e-07**	**9.13e-06**
N/A	**1.67e-07**	**3.62e-06**	**4.77e-06**	**5.44e-05**
N/A	**3.03e-06**	**1.65e-04**	1.35e-02	5.63e-02
N/A	**5.90e-03**	9.97e-02	**5.51e-03**	1.37e-01
N/A	**6.15e-04**	**2.31e-03**	6.25e-02	1.99e-01

p-values

Figure A.1(e): `prob_blocksworld` problem `p5`. The landmark extraction algorithm LM^{RHW} generated 10 nontrivial landmarks for this problem.



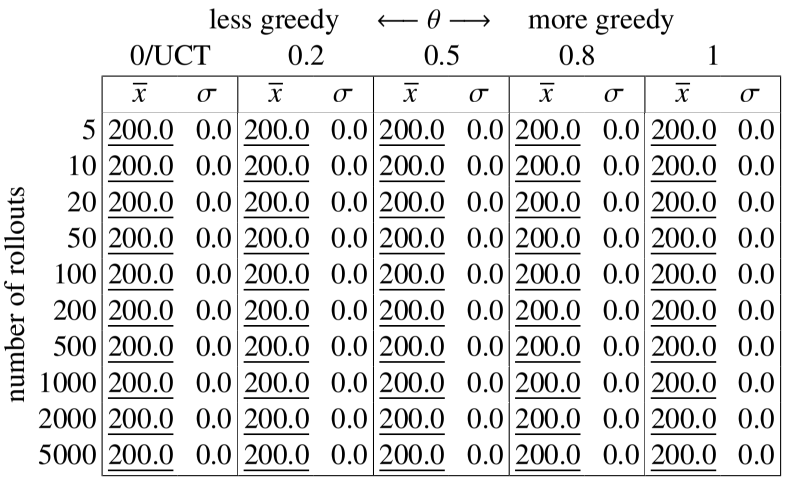
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	200.0	0.0	200.0	0.0	200.0	0.0	197.8	13.8	200.0	0.0
10	200.0	0.0	200.0	0.0	198.2	15.9	200.0	0.0	200.0	0.0
20	200.0	0.0	200.0	0.0	198.3	14.3	199.3	5.8	200.0	0.0
50	200.0	0.0	191.8	31.1	197.0	19.1	190.3	33.7	196.0	19.8
100	200.0	0.0	198.1	13.8	192.8	30.6	198.3	15.0	198.5	12.9
200	200.0	0.0	196.7	20.4	200.0	0.0	200.0	0.0	200.0	0.0
500	200.0	0.0	192.9	30.3	200.0	0.0	200.0	0.0	200.0	0.0
1000	200.0	0.0	196.1	23.7	195.3	24.6	200.0	0.0	196.9	19.3
2000	200.0	0.0	200.0	0.0	196.4	21.9	194.6	27.3	196.4	22.1
5000	200.0	0.0	196.1	23.7	195.7	25.8	200.0	0.0	196.0	24.5

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	1	1	1.59e-01	1
N/A	1	3.24e-01	1	1
N/A	1	3.24e-01	3.24e-01	1
N/A	2.38e-02	1.59e-01	1.29e-02	8.26e-02
N/A	1.59e-01	4.41e-02	3.24e-01	3.24e-01
N/A	1.59e-01	1	1	1
N/A	4.41e-02	1	1	1
N/A	1.59e-01	8.26e-02	1	1.59e-01
N/A	1	1.59e-01	8.26e-02	1.59e-01
N/A	1.59e-01	1.53e-01	1	1.56e-01

p-values

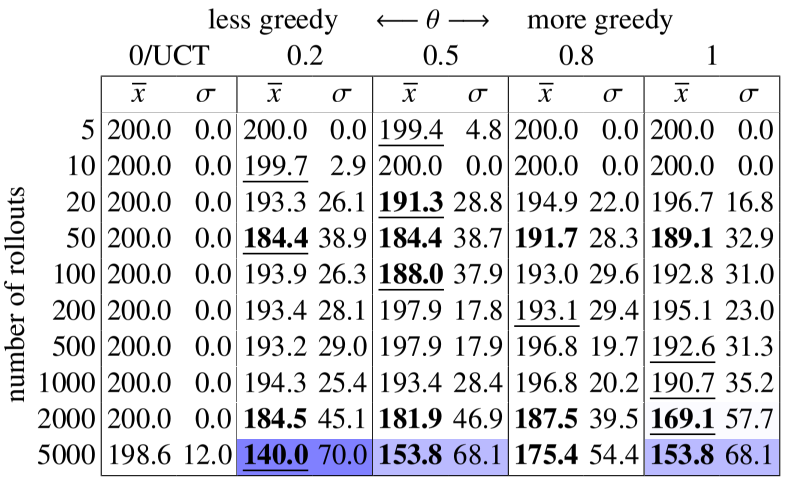
Figure A.1(f): `prob_blocksworld` problem `p6`. The landmark extraction algorithm LM^{RHW} generated 20 nontrivial landmarks for this problem.



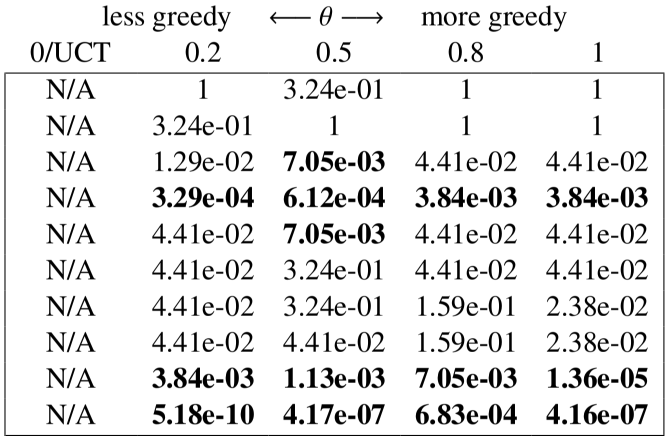
Average cost

p-values

Figure A.1(g): `prob_blocksworld` problem `p7`. The landmark extraction algorithm LM^{RHW} generated 26 nontrivial landmarks for this problem.

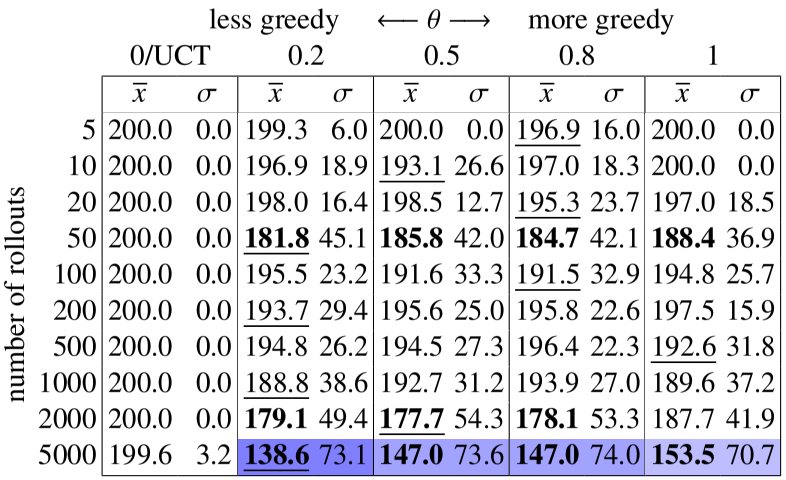


Average cost



p-values

Figure A.1(h): `prob_blocksworld` problem `p8`. The landmark extraction algorithm LM^{RHW} generated 24 nontrivial landmarks for this problem.



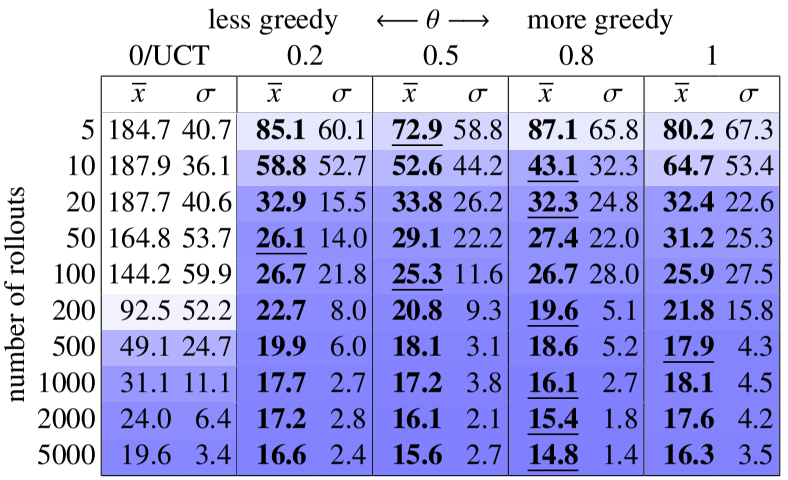
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	200.0	0.0	199.3	6.0	200.0	0.0	196.9	16.0	200.0	0.0
10	200.0	0.0	196.9	18.9	193.1	26.6	197.0	18.3	200.0	0.0
20	200.0	0.0	198.0	16.4	198.5	12.7	195.3	23.7	197.0	18.5
50	200.0	0.0	**181.8**	45.1	**185.8**	42.0	**184.7**	42.1	**188.4**	36.9
100	200.0	0.0	195.5	23.2	191.6	33.3	191.5	32.9	194.8	25.7
200	200.0	0.0	193.7	29.4	195.6	25.0	195.8	22.6	197.5	15.9
500	200.0	0.0	194.8	26.2	194.5	27.3	196.4	22.3	192.6	31.8
1000	200.0	0.0	188.8	38.6	192.7	31.2	193.9	27.0	189.6	37.2
2000	200.0	0.0	**179.1**	49.4	**177.7**	54.3	**178.1**	53.3	187.7	41.9
5000	199.6	3.2	**138.6**	73.1	**147.0**	73.6	**147.0**	74.0	**153.5**	70.7

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	3.24e-01	1	8.26e-02	1
N/A	1.59e-01	2.38e-02	1.59e-01	1
N/A	1.59e-01	3.24e-01	8.26e-02	1.59e-01
N/A	**6.12e-04**	**3.84e-03**	**2.09e-03**	**7.05e-03**
N/A	8.26e-02	2.38e-02	2.38e-02	8.26e-02
N/A	4.41e-02	8.26e-02	8.26e-02	1.59e-01
N/A	8.26e-02	8.26e-02	1.59e-01	4.41e-02
N/A	1.29e-02	4.41e-02	4.41e-02	1.29e-02
N/A	**3.29e-04**	**6.12e-04**	**6.11e-04**	1.29e-02
N/A	**4.69e-10**	**4.81e-08**	**4.81e-08**	**8.49e-07**

p-values

Figure A.1(i): `prob_blocksworld` problem `p9`. The landmark extraction algorithm LM^{RHW} generated 22 nontrivial landmarks for this problem.



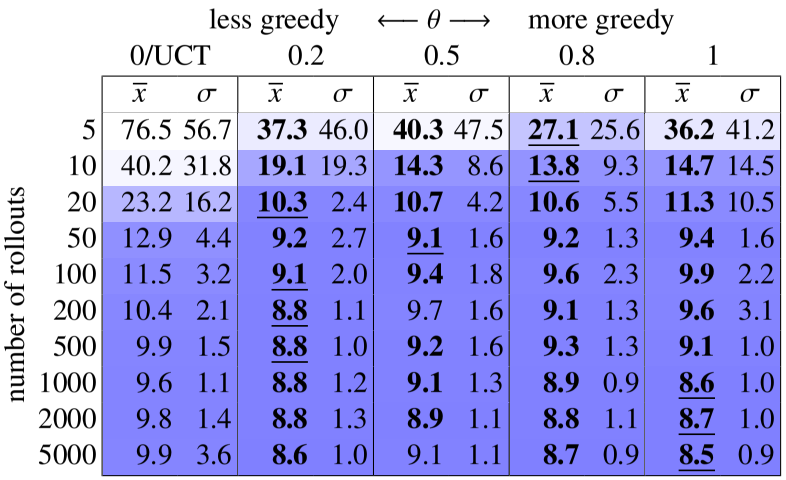
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	184.7	40.7	**85.1**	60.1	**72.9**	58.8	**87.1**	65.8	**80.2**	67.3
10	187.9	36.1	**58.8**	52.7	**52.6**	44.2	**43.1**	32.3	**64.7**	53.4
20	187.7	40.6	**32.9**	15.5	**33.8**	26.2	**32.3**	24.8	**32.4**	22.6
50	164.8	53.7	**26.1**	14.0	**29.1**	22.2	**27.4**	22.0	**31.2**	25.3
100	144.2	59.9	**26.7**	21.8	**25.3**	11.6	**26.7**	28.0	**25.9**	27.5
200	92.5	52.2	**22.7**	8.0	**20.8**	9.3	**19.6**	5.1	**21.8**	15.8
500	49.1	24.7	**19.9**	6.0	**18.1**	3.1	**18.6**	5.2	**17.9**	4.3
1000	31.1	11.1	**17.7**	2.7	**17.2**	3.8	**16.1**	2.7	**18.1**	4.5
2000	24.0	6.4	**17.2**	2.8	**16.1**	2.1	**15.4**	1.8	**17.6**	4.2
5000	19.6	3.4	**16.6**	2.4	**15.6**	2.7	**14.8**	1.4	**16.3**	3.5

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	**1.24e-17**	**6.48e-20**	**2.01e-16**	**4.64e-17**
N/A	**3.17e-23**	**3.87e-24**	**2.16e-26**	**6.39e-22**
N/A	**3.40e-27**	**3.90e-27**	**5.64e-27**	**5.09e-27**
N/A	**6.75e-26**	**4.84e-25**	**2.58e-25**	**1.08e-24**
N/A	**8.00e-24**	**1.82e-24**	**8.67e-24**	**8.61e-24**
N/A	**2.44e-23**	**6.34e-24**	**9.21e-25**	**1.70e-23**
N/A	**4.89e-19**	**1.81e-21**	**2.09e-20**	**3.14e-21**
N/A	**3.32e-20**	**4.94e-21**	**4.06e-23**	**6.24e-17**
N/A	**3.76e-16**	**3.75e-20**	**8.22e-23**	**5.93e-12**
N/A	**1.45e-09**	**1.33e-16**	**6.49e-21**	**7.94e-10**

p-values

Figure A.2(a): `elevators` problem `p1`. The landmark extraction algorithm LM^{RHW} generated 10 nontrivial landmarks for this problem.



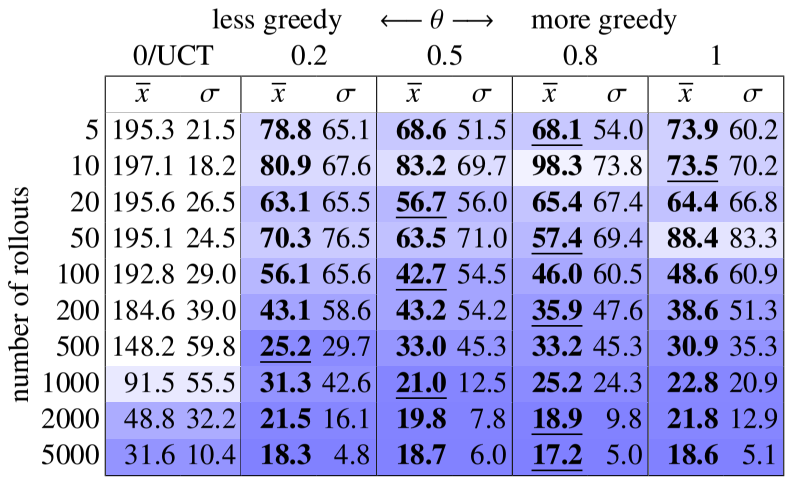
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	76.5	56.7	**37.3**	46.0	**40.3**	47.5	**27.1**	25.6	**36.2**	41.2
10	40.2	31.8	**19.1**	19.3	**14.3**	8.6	**13.8**	9.3	**14.7**	14.5
20	23.2	16.2	**10.3**	2.4	**10.7**	4.2	**10.6**	5.5	**11.3**	10.5
50	12.9	4.4	**9.2**	2.7	**9.1**	1.6	**9.2**	1.3	**9.4**	1.6
100	11.5	3.2	**9.1**	2.0	**9.4**	1.8	**9.6**	2.3	**9.9**	2.2
200	10.4	2.1	**8.8**	1.1	9.7	1.6	**9.1**	1.3	**9.6**	3.1
500	9.9	1.5	**8.8**	1.0	**9.2**	1.6	**9.3**	1.3	**9.1**	1.0
1000	9.6	1.1	**8.8**	1.2	**9.1**	1.3	**8.9**	0.9	**8.6**	1.0
2000	9.8	1.4	**8.8**	1.3	**8.9**	1.1	**8.8**	1.1	**8.7**	1.0
5000	9.9	3.6	**8.6**	1.0	9.1	1.1	**8.7**	0.9	**8.5**	0.9

Average cost

0/UCT	less greedy 0.2	0.5	⟵ θ ⟶ 0.8	more greedy 1
N/A	**5.18e-09**	**6.08e-08**	**1.70e-12**	**5.43e-09**
N/A	**2.27e-11**	**3.61e-15**	**1.01e-16**	**2.09e-16**
N/A	**8.39e-14**	**2.60e-13**	**4.21e-14**	**2.96e-14**
N/A	**3.20e-11**	**2.51e-10**	**8.09e-09**	**6.40e-08**
N/A	**3.13e-08**	**4.31e-06**	**2.11e-05**	**1.60e-03**
N/A	**2.64e-09**	5.74e-02	**1.34e-05**	**3.27e-04**
N/A	**9.08e-07**	**2.60e-04**	**5.72e-03**	**3.82e-04**
N/A	**1.86e-05**	**1.02e-02**	**1.43e-04**	**5.10e-08**
N/A	**1.65e-06**	**2.73e-05**	**2.55e-06**	**6.72e-08**
N/A	**8.67e-06**	1.01e-01	**9.62e-05**	**1.21e-06**

p-values

Figure A.2(b): `elevators` problem `p2`. The landmark extraction algorithm LM^{RHW} generated 16 nontrivial landmarks for this problem.



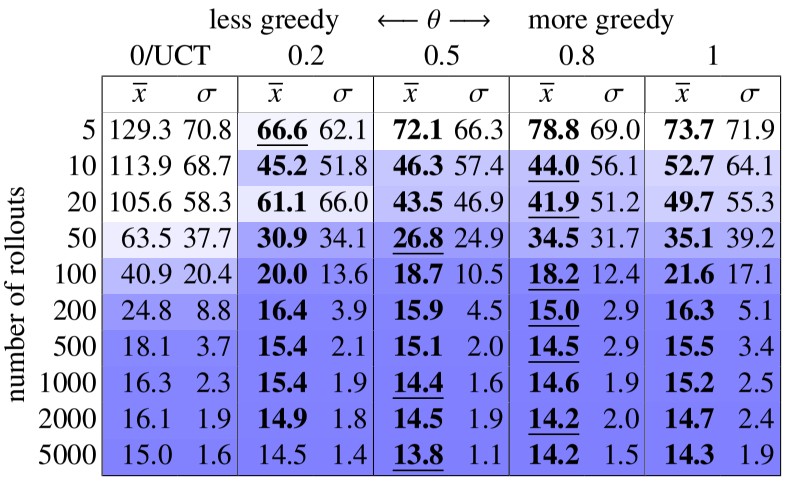
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	195.3	21.5	**78.8**	65.1	**68.6**	51.5	**68.1**	54.0	**73.9**	60.2
10	197.1	18.2	**80.9**	67.6	**83.2**	69.7	**98.3**	73.8	**73.5**	70.2
20	195.6	26.5	**63.1**	65.5	**56.7**	56.0	**65.4**	67.4	**64.4**	66.8
50	195.1	24.5	**70.3**	76.5	**63.5**	71.0	**57.4**	69.4	**88.4**	83.3
100	192.8	29.0	**56.1**	65.6	**42.7**	54.5	**46.0**	60.5	**48.6**	60.9
200	184.6	39.0	**43.1**	58.6	**43.2**	54.2	**35.9**	47.6	**38.6**	51.3
500	148.2	59.8	**25.2**	29.7	**33.0**	45.3	**33.2**	45.3	**30.9**	35.3
1000	91.5	55.5	**31.3**	42.6	**21.0**	12.5	**25.2**	24.3	**22.8**	20.9
2000	48.8	32.2	**21.5**	16.1	**19.8**	7.8	**18.9**	9.8	**21.8**	12.9
5000	31.6	10.4	**18.3**	4.8	**18.7**	6.0	**17.2**	5.0	**18.6**	5.1

Average cost

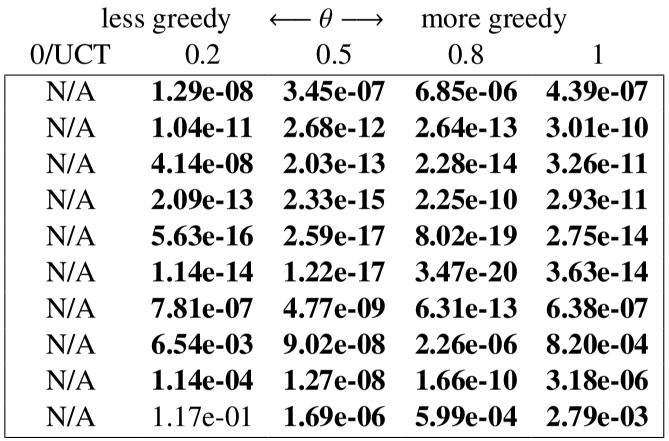
0/UCT	less greedy 0.2	0.5	⟵ θ ⟶ 0.8	more greedy 1
N/A	**2.17e-21**	**2.89e-25**	**5.79e-25**	**3.60e-22**
N/A	**3.57e-20**	**2.71e-19**	**6.88e-17**	**2.45e-20**
N/A	**3.11e-21**	**4.44e-23**	**8.96e-21**	**1.14e-20**
N/A	**1.35e-17**	**5.23e-20**	**3.19e-20**	**1.08e-14**
N/A	**4.02e-21**	**4.76e-24**	**5.35e-23**	**1.86e-22**
N/A	**9.86e-22**	**1.41e-22**	**4.37e-24**	**1.99e-23**
N/A	**5.33e-24**	**4.74e-21**	**6.40e-21**	**4.05e-22**
N/A	**6.98e-18**	**1.18e-22**	**1.15e-19**	**3.47e-22**
N/A	**3.93e-20**	**2.18e-20**	**2.25e-21**	**8.80e-17**
N/A	**3.15e-21**	**2.86e-19**	**2.63e-22**	**2.39e-17**

p-values

Figure A.2(c): `elevators` problem `p3`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.

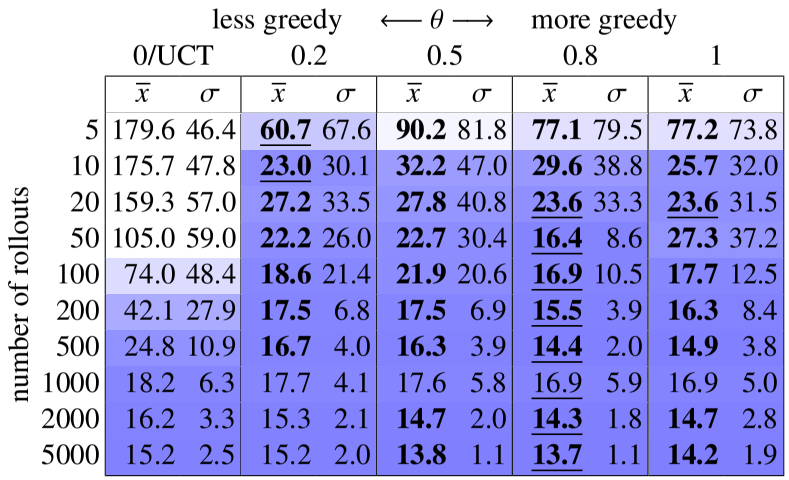


Average cost

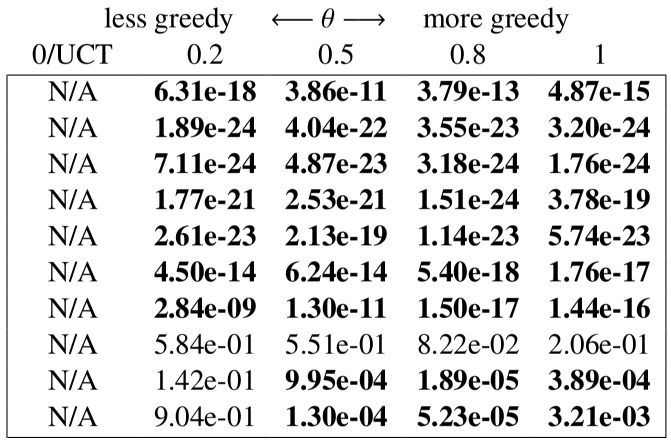


p-values

Figure A.2(d): `elevators` problem `p4`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.

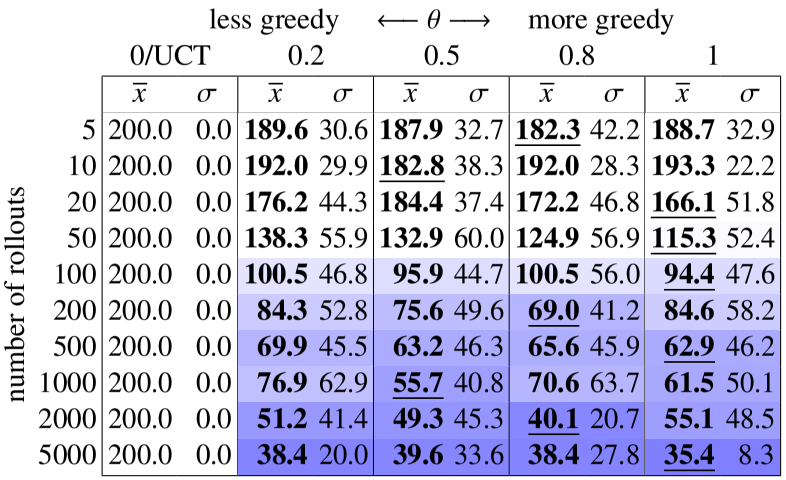


Average cost

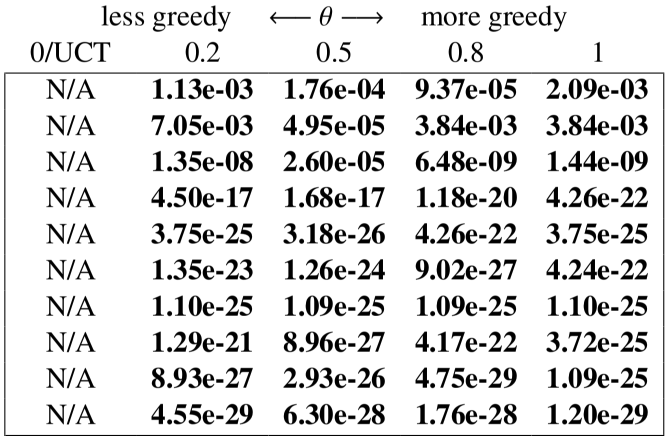


p-values

Figure A.2(e): `elevators` problem `p5`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.

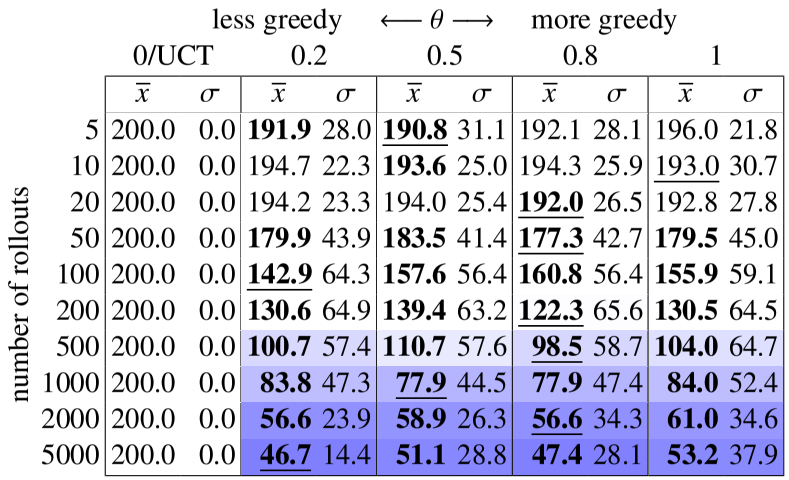


Average cost

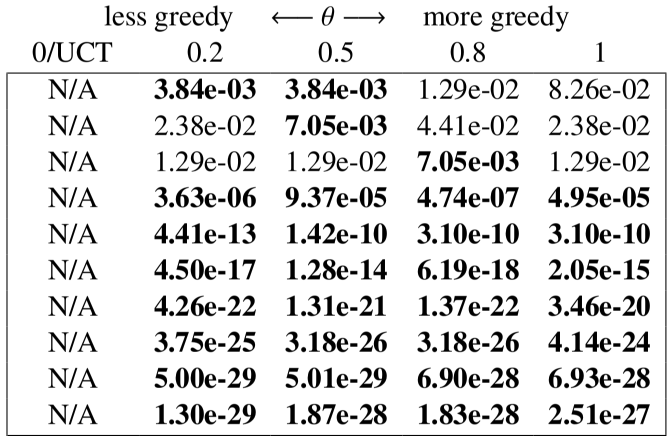


p-values

Figure A.2(f): `elevators` problem `p6`. The landmark extraction algorithm LM^{RHW} generated 15 nontrivial landmarks for this problem.

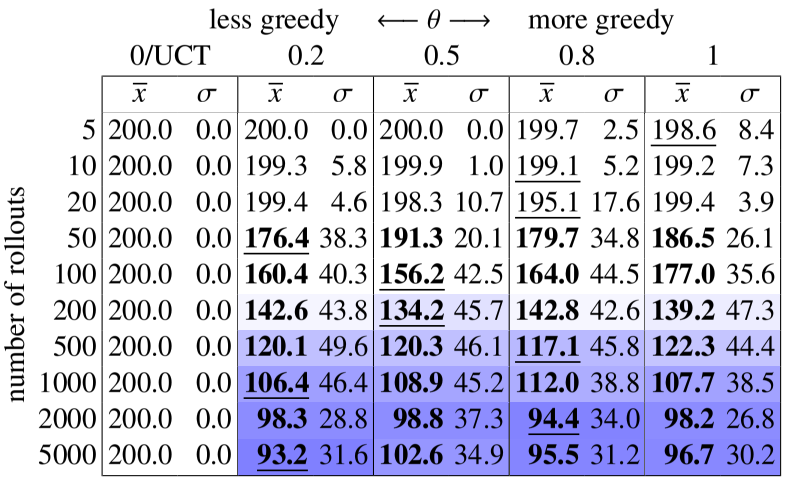


Average cost



p-values

Figure A.2(g): `elevators` problem `p7`. The landmark extraction algorithm LM^{RHW} generated 16 nontrivial landmarks for this problem.



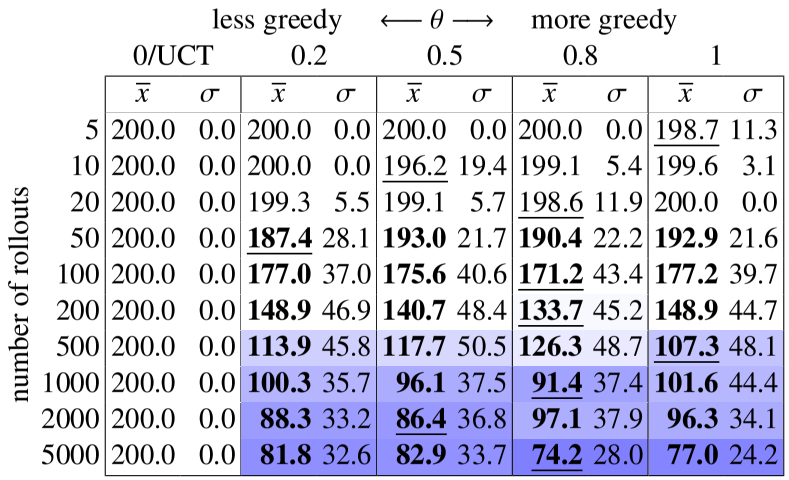
number of rollouts	less greedy 0/UCT $\bar{x}$	σ	0.2 $\bar{x}$	σ	⟵ θ ⟶ 0.5 $\bar{x}$	σ	more greedy 0.8 $\bar{x}$	σ	1 $\bar{x}$	σ
5	200.0	0.0	200.0	0.0	200.0	0.0	199.7	2.5	198.6	8.4
10	200.0	0.0	199.3	5.8	199.9	1.0	199.1	5.2	199.2	7.3
20	200.0	0.0	199.4	4.6	198.3	10.7	195.1	17.6	199.4	3.9
50	200.0	0.0	**176.4**	38.3	**191.3**	20.1	**179.7**	34.8	**186.5**	26.1
100	200.0	0.0	**160.4**	40.3	**156.2**	42.5	**164.0**	44.5	**177.0**	35.6
200	200.0	0.0	**142.6**	43.8	**134.2**	45.7	**142.8**	42.6	**139.2**	47.3
500	200.0	0.0	**120.1**	49.6	**120.3**	46.1	**117.1**	45.8	**122.3**	44.4
1000	200.0	0.0	**106.4**	46.4	**108.9**	45.2	**112.0**	38.8	**107.7**	38.5
2000	200.0	0.0	**98.3**	28.8	**98.8**	37.3	**94.4**	34.0	**98.2**	26.8
5000	200.0	0.0	**93.2**	31.6	**102.6**	34.9	**95.5**	31.2	**96.7**	30.2

Average cost

less greedy ⟵ θ ⟶ more greedy 0/UCT	0.2	0.5	0.8	1
N/A	1	1	3.24e-01	1.59e-01
N/A	3.24e-01	3.24e-01	1.59e-01	3.24e-01
N/A	1.59e-01	1.59e-01	1.29e-02	1.59e-01
N/A	**3.07e-09**	**1.36e-05**	**5.76e-08**	**4.74e-07**
N/A	**1.19e-16**	**3.11e-16**	**7.70e-14**	**6.40e-11**
N/A	**3.95e-21**	**3.46e-20**	**1.18e-20**	**2.87e-19**
N/A	**4.26e-22**	**4.34e-23**	**1.35e-23**	**3.76e-25**
N/A	**1.26e-24**	**4.16e-24**	**3.18e-26**	**3.18e-26**
N/A	**1.32e-29**	**9.04e-27**	**6.97e-28**	**1.32e-29**
N/A	**1.89e-28**	**1.89e-28**	**5.03e-29**	**1.32e-29**

p-values

Figure A.2(h): `elevators` problem `p8`. The landmark extraction algorithm LM^{RHW} generated 16 nontrivial landmarks for this problem.



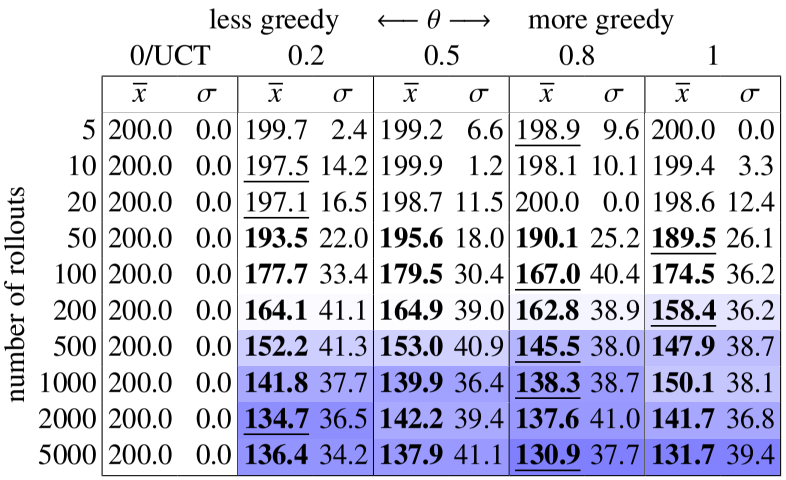
number of rollouts	less greedy 0/UCT $\bar{x}$	σ	0.2 $\bar{x}$	σ	⟵ θ ⟶ 0.5 $\bar{x}$	σ	more greedy 0.8 $\bar{x}$	σ	1 $\bar{x}$	σ
5	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	198.7	11.3
10	200.0	0.0	200.0	0.0	196.2	19.4	199.1	5.4	199.6	3.1
20	200.0	0.0	199.3	5.5	199.1	5.7	198.6	11.9	200.0	0.0
50	200.0	0.0	**187.4**	28.1	**193.0**	21.7	**190.4**	22.2	**192.9**	21.6
100	200.0	0.0	**177.0**	37.0	**175.6**	40.6	**171.2**	43.4	**177.2**	39.7
200	200.0	0.0	**148.9**	46.9	**140.7**	48.4	**133.7**	45.2	**148.9**	44.7
500	200.0	0.0	**113.9**	45.8	**117.7**	50.5	**126.3**	48.7	**107.3**	48.1
1000	200.0	0.0	**100.3**	35.7	**96.1**	37.5	**91.4**	37.4	**101.6**	44.4
2000	200.0	0.0	**88.3**	33.2	**86.4**	36.8	**97.1**	37.9	**96.3**	34.1
5000	200.0	0.0	**81.8**	32.6	**82.9**	33.7	**74.2**	28.0	**77.0**	24.2

Average cost

less greedy ⟵ θ ⟶ more greedy 0/UCT	0.2	0.5	0.8	1
N/A	1	1	1	3.24e-01
N/A	1	8.26e-02	1.59e-01	3.24e-01
N/A	1.59e-01	1.59e-01	3.24e-01	1
N/A	**7.05e-06**	**2.09e-03**	**2.60e-05**	**3.84e-03**
N/A	**6.72e-10**	**3.07e-09**	**6.40e-11**	**6.48e-09**
N/A	**8.09e-19**	**3.46e-20**	**1.37e-22**	**1.68e-17**
N/A	**1.35e-23**	**3.95e-21**	**3.95e-21**	**1.26e-24**
N/A	**2.53e-27**	**9.05e-27**	**6.97e-28**	**3.75e-25**
N/A	**1.89e-28**	**2.53e-27**	**2.53e-27**	**1.89e-28**
N/A	**6.98e-28**	**6.98e-28**	**5.01e-29**	**1.32e-29**

p-values

Figure A.2(i): `elevators` problem `p9`. The landmark extraction algorithm LM^{RHW} generated 14 nontrivial landmarks for this problem.



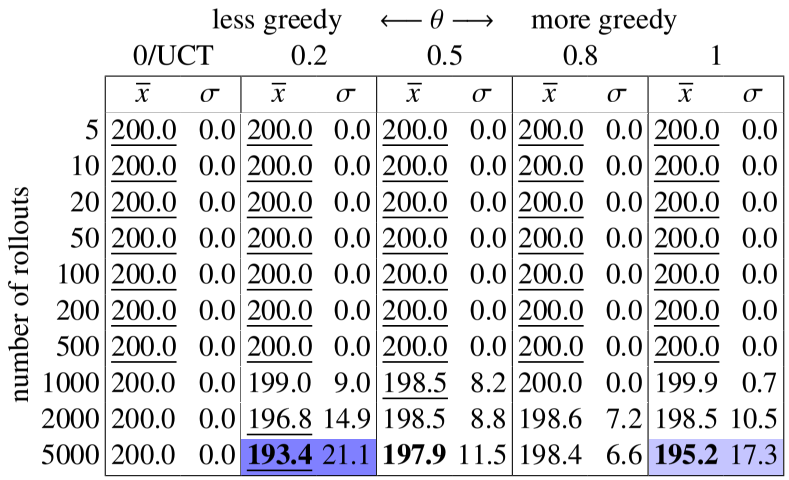
number of rollouts	less greedy ⟵ θ ⟶ more greedy									
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	200.0	0.0	199.7	2.4	199.2	6.6	198.9	9.6	200.0	0.0
10	200.0	0.0	197.5	14.2	199.9	1.2	198.1	10.1	199.4	3.3
20	200.0	0.0	197.1	16.5	198.7	11.5	200.0	0.0	198.6	12.4
50	200.0	0.0	**193.5**	22.0	**195.6**	18.0	**190.1**	25.2	**189.5**	26.1
100	200.0	0.0	**177.7**	33.4	**179.5**	30.4	**167.0**	40.4	**174.5**	36.2
200	200.0	0.0	**164.1**	41.1	**164.9**	39.0	**162.8**	38.9	**158.4**	36.2
500	200.0	0.0	**152.2**	41.3	**153.0**	40.9	**145.5**	38.0	**147.9**	38.7
1000	200.0	0.0	**141.8**	37.7	**139.9**	36.4	**138.3**	38.7	**150.1**	38.1
2000	200.0	0.0	**134.7**	36.5	**142.2**	39.4	**137.6**	41.0	**141.7**	36.8
5000	200.0	0.0	**136.4**	34.2	**137.9**	41.1	**130.9**	37.7	**131.7**	39.4

Average cost

0/UCT	less greedy ⟵ θ ⟶ more greedy			
	0.2	0.5	0.8	1
N/A	3.24e-01	3.24e-01	3.24e-01	1
N/A	8.26e-02	3.24e-01	4.41e-02	8.26e-02
N/A	8.26e-02	3.24e-01	1	3.24e-01
N/A	**2.09e-03**	**3.84e-03**	**4.95e-05**	**1.76e-04**
N/A	**3.10e-10**	**6.40e-11**	**1.85e-13**	**6.40e-11**
N/A	**3.16e-14**	**8.02e-16**	**2.25e-18**	**2.25e-18**
N/A	**1.37e-22**	**2.87e-19**	**1.35e-23**	**1.31e-21**
N/A	**1.26e-24**	**1.10e-25**	**1.26e-24**	**1.37e-22**
N/A	**9.06e-27**	**3.75e-25**	**1.10e-25**	**4.16e-24**
N/A	**6.98e-28**	**1.26e-24**	**3.18e-26**	**3.76e-25**

p-values

Figure A.2(j): `elevators` problem `p10`. The landmark extraction algorithm LM^{RHW} generated 16 nontrivial landmarks for this problem.



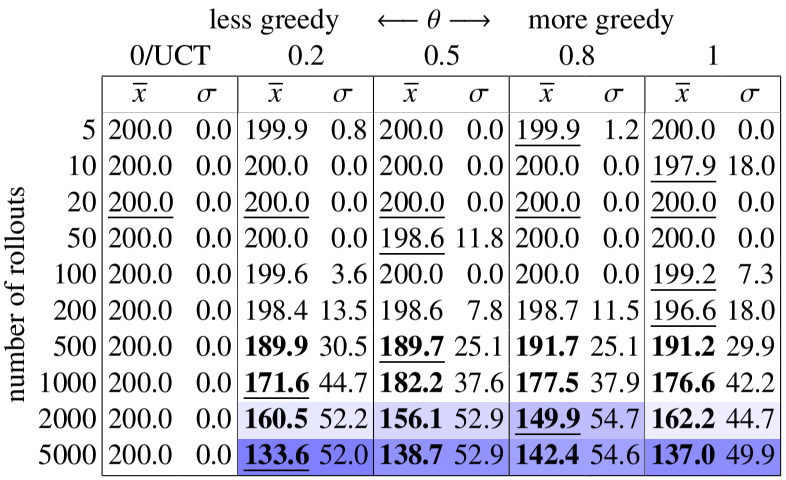
number of rollouts	less greedy ⟵ θ ⟶ more greedy									
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
10	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
20	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
50	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
100	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
200	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
500	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
1000	200.0	0.0	199.0	9.0	198.5	8.2	200.0	0.0	199.9	0.7
2000	200.0	0.0	196.8	14.9	198.5	8.8	198.6	7.2	198.5	10.5
5000	200.0	0.0	**193.4**	21.1	**197.9**	11.5	198.4	6.6	**195.2**	17.3

Average cost

0/UCT	less greedy ⟵ θ ⟶ more greedy			
	0.2	0.5	0.8	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	3.24e-01	8.26e-02	1	3.24e-01
N/A	4.41e-02	2.38e-02	4.41e-02	8.26e-02
N/A	**6.12e-04**	**7.05e-03**	1.29e-02	**3.84e-03**

p-values

Figure A.2(k): `elevators` problem `p11`. The landmark extraction algorithm LM^{RHW} generated 26 nontrivial landmarks for this problem.



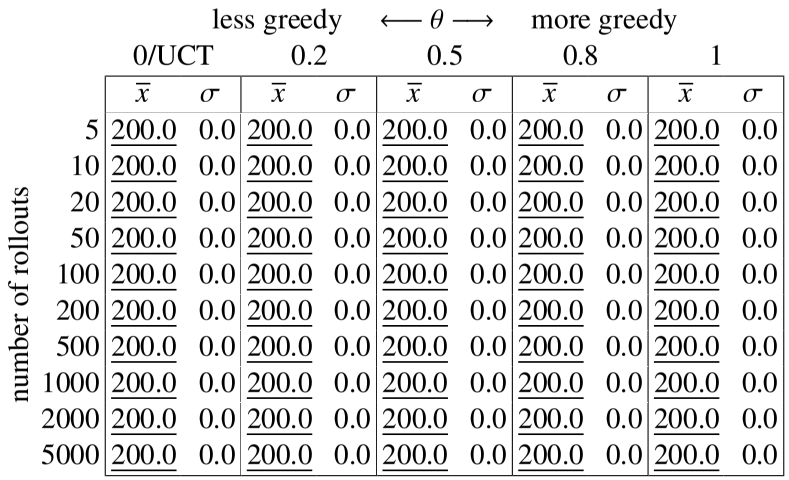
	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	200.0	0.0	199.9	0.8	200.0	0.0	199.9	1.2	200.0	0.0
10	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	197.9	18.0
20	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
50	200.0	0.0	200.0	0.0	198.6	11.8	200.0	0.0	200.0	0.0
100	200.0	0.0	199.6	3.6	200.0	0.0	200.0	0.0	199.2	7.3
200	200.0	0.0	198.4	13.5	198.6	7.8	198.7	11.5	196.6	18.0
500	200.0	0.0	**189.9**	30.5	**189.7**	25.1	**191.7**	25.1	**191.2**	29.9
1000	200.0	0.0	**171.6**	44.7	**182.2**	37.6	**177.5**	37.9	**176.6**	42.2
2000	200.0	0.0	**160.5**	52.2	**156.1**	52.9	**149.9**	54.7	**162.2**	44.7
5000	200.0	0.0	**133.6**	52.0	**138.7**	52.9	**142.4**	54.6	**137.0**	49.9

Average cost

	0/UCT	0.2	0.5	0.8	1
5	N/A	3.24e-01	1	3.24e-01	1
10	N/A	1	1	1	3.24e-01
20	N/A	1	1	1	1
50	N/A	1	3.24e-01	1	1
100	N/A	3.24e-01	1	1	3.24e-01
200	N/A	3.24e-01	8.26e-02	1.59e-01	8.26e-02
500	N/A	**1.13e-03**	**3.29e-04**	**6.12e-04**	**2.09e-03**
1000	N/A	**5.76e-08**	**7.05e-06**	**3.07e-09**	**5.76e-08**
2000	N/A	**1.04e-12**	**1.85e-13**	**4.41e-13**	**7.70e-14**
5000	N/A	**1.18e-20**	**8.09e-19**	**1.68e-17**	**2.87e-19**

p-values

Figure A.2(l): `elevators` problem `p12`. The landmark extraction algorithm LM^{RHW} generated 18 nontrivial landmarks for this problem.



	0/UCT		0.2		0.5		0.8		1	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
10	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
20	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
50	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
100	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
200	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
500	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
1000	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
2000	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
5000	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0

Average cost

	0/UCT	0.2	0.5	0.8	1
5	N/A	1	1	1	1
10	N/A	1	1	1	1
20	N/A	1	1	1	1
50	N/A	1	1	1	1
100	N/A	1	1	1	1
200	N/A	1	1	1	1
500	N/A	1	1	1	1
1000	N/A	1	1	1	1
2000	N/A	1	1	1	1
5000	N/A	1	1	1	1

p-values

Figure A.2(m): `elevators` problem `p13`. The landmark extraction algorithm LM^{RHW} generated 30 nontrivial landmarks for this problem.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT $\bar{x}$	0/UCT σ	0.2 $\bar{x}$	0.2 σ	0.5 $\bar{x}$	0.5 σ	0.8 $\bar{x}$	0.8 σ	1 $\bar{x}$	1 σ
5	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
10	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
20	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
50	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
100	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
200	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
500	200.0	0.0	198.2	11.3	198.6	8.6	199.3	3.5	199.5	2.5
1000	200.0	0.0	196.8	12.8	**193.5**	14.6	**192.0**	17.9	**191.6**	19.8
2000	200.0	0.0	**188.3**	21.4	**185.3**	24.7	**186.2**	23.9	**178.9**	29.4
5000	200.0	0.0	**162.3**	30.5	**156.9**	32.7	**162.5**	31.5	**166.3**	32.0

Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	8.26e-02	1.59e-01	4.41e-02	4.41e-02
N/A	2.38e-02	**9.42e-07**	**7.05e-06**	**1.36e-05**
N/A	**1.35e-08**	**2.81e-08**	**1.17e-07**	**6.40e-11**
N/A	**3.46e-20**	**3.94e-21**	**3.46e-20**	**2.87e-19**

p-values

Figure A.2(n): `elevators` problem `p14`. The landmark extraction algorithm LM^{RHW} generated 24 nontrivial landmarks for this problem.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT $\bar{x}$	0/UCT σ	0.2 $\bar{x}$	0.2 σ	0.5 $\bar{x}$	0.5 σ	0.8 $\bar{x}$	0.8 σ	1 $\bar{x}$	1 σ
5	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
10	200.0	0.0	200.0	0.0	200.0	0.0	199.0	8.8	200.0	0.0
20	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
50	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
100	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
200	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0
500	200.0	0.0	200.0	0.0	200.0	0.0	199.7	1.9	200.0	0.0
1000	200.0	0.0	199.4	5.3	199.9	1.0	200.0	0.0	199.3	4.0
2000	200.0	0.0	198.3	8.3	199.0	5.6	199.4	4.4	199.0	7.5
5000	200.0	0.0	**195.1**	18.0	**197.1**	10.8	**196.6**	10.1	**196.7**	11.2

Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	1	1	1	1
N/A	1	1	3.24e-01	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1.59e-01	1
N/A	3.24e-01	3.24e-01	1	1.59e-01
N/A	2.38e-02	8.26e-02	1.59e-01	1.59e-01
N/A	**1.13e-03**	**2.09e-03**	**1.13e-03**	**3.84e-03**

p-values

Figure A.2(o): `elevators` problem `p15`. The landmark extraction algorithm LM^{RHW} generated 24 nontrivial landmarks for this problem.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT $\bar{x}$	0/UCT σ	0.2 $\bar{x}$	0.2 σ	0.5 $\bar{x}$	0.5 σ	0.8 $\bar{x}$	0.8 σ	1 $\bar{x}$	1 σ
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure A.3(a): `zenotravel` problem `p1`. The landmark extraction algorithm LM^{RHW} generated 0 nontrivial landmarks for this problem. This problem instance is trivial—the initial state satisfies the goal condition.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT $\bar{x}$	0/UCT σ	0.2 $\bar{x}$	0.2 σ	0.5 $\bar{x}$	0.5 σ	0.8 $\bar{x}$	0.8 σ	1 $\bar{x}$	1 σ
5	200.0	0.0	200.0	0.0	200.0	0.0	199.9	0.9	197.9	9.5
10	200.0	0.0	199.2	4.5	199.3	4.5	199.5	4.0	196.3	18.2
20	200.0	0.0	197.8	15.6	199.9	1.3	198.2	12.7	199.9	0.8
50	200.0	0.0	198.4	7.7	196.5	17.8	**195.7**	15.8	197.4	10.5
100	198.8	10.6	196.8	17.6	197.7	12.1	195.4	19.9	196.4	15.3
200	196.8	14.2	190.7	25.2	198.7	6.3	192.4	21.0	194.0	17.4
500	198.0	9.8	**191.2**	21.3	**188.5**	23.9	**189.2**	25.2	**189.9**	22.9
1000	196.3	22.3	**190.1**	21.6	**183.2**	30.6	**192.5**	22.5	**185.3**	27.6
2000	194.3	21.8	**175.4**	32.5	**178.1**	30.3	**185.6**	26.4	**185.3**	27.1
5000	194.9	22.5	**172.0**	35.2	**169.4**	35.7	**173.3**	36.3	**175.3**	33.1

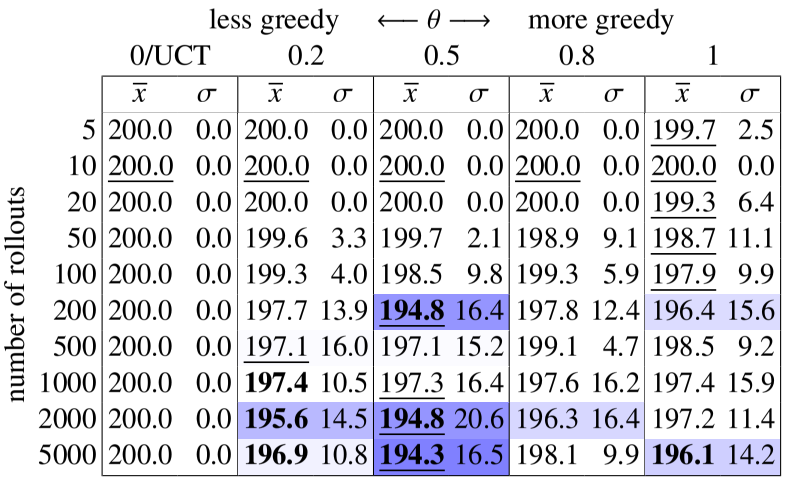
Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	1	1	3.24e-01	2.38e-02
N/A	8.26e-02	1.59e-01	3.24e-01	4.41e-02
N/A	1.59e-01	3.24e-01	1.59e-01	1.59e-01
N/A	8.26e-02	8.26e-02	**7.05e-03**	2.38e-02
N/A	3.12e-01	1.84e-01	1.01e-01	1.84e-01
N/A	2.30e-02	9.58e-01	6.25e-02	6.77e-02
N/A	**1.13e-02**	**2.31e-04**	**6.17e-03**	**3.76e-03**
N/A	**1.03e-03**	**7.68e-06**	**1.02e-02**	**1.48e-05**
N/A	**1.14e-07**	**2.90e-07**	**8.54e-04**	**3.30e-04**
N/A	**5.64e-11**	**2.20e-10**	**9.85e-08**	**3.12e-08**

p-values

Figure A.3(b): `zenotravel` problem `p2`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.

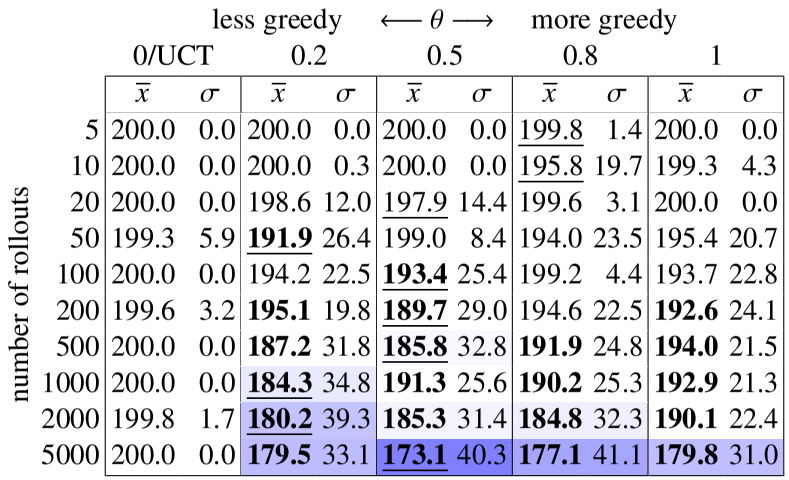


Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1	1	1	3.24e-01
N/A	1	1	1	1
N/A	1	1	1	3.24e-01
N/A	3.24e-01	1.59e-01	3.24e-01	3.24e-01
N/A	8.26e-02	8.26e-02	1.59e-01	4.41e-02
N/A	1.59e-01	**2.09e-03**	8.26e-02	4.41e-02
N/A	4.41e-02	4.41e-02	8.26e-02	8.26e-02
N/A	**7.05e-03**	2.38e-02	2.38e-02	4.41e-02
N/A	**2.09e-03**	**2.09e-03**	2.38e-02	2.38e-02
N/A	**7.05e-03**	**6.12e-04**	1.29e-02	**3.84e-03**

p-values

Figure A.3(c): `zenotravel` problem `p3`. The landmark extraction algorithm LM^{RHW} generated 13 nontrivial landmarks for this problem.



Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1	1	3.24e-01	1
N/A	3.24e-01	1	4.41e-02	1.59e-01
N/A	3.24e-01	1.59e-01	3.24e-01	1
N/A	**9.11e-03**	1	2.99e-02	9.68e-02
N/A	1.29e-02	**7.05e-03**	8.26e-02	1.29e-02
N/A	**9.59e-03**	**1.43e-03**	9.24e-02	**8.66e-03**
N/A	**1.36e-05**	**4.95e-05**	**2.09e-03**	**7.05e-03**
N/A	**1.36e-05**	**6.12e-04**	**3.29e-04**	**6.12e-04**
N/A	**1.43e-05**	**3.87e-06**	**2.72e-05**	**5.60e-05**
N/A	**1.44e-09**	**6.40e-11**	**3.07e-09**	**3.07e-09**

p-values

Figure A.3(d): `zenotravel` problem `p4`. The landmark extraction algorithm LM^{RHW} generated 11 nontrivial landmarks for this problem.

number of rollouts	less greedy ⟵ θ ⟶ more greedy									
	0/UCT		0.2		0.5		0.8		1	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
5	200.0	0.0	200.0	0.0	200.0	0.0	200.0	0.0	199.8	1.7
10	200.0	0.0	200.0	0.0	199.9	0.6	200.0	0.0	199.7	2.9
20	200.0	0.0	195.2	19.0	199.2	7.3	200.0	0.0	198.4	14.0
50	200.0	0.0	199.3	5.2	197.2	15.2	200.0	0.0	199.0	8.5
100	200.0	0.0	197.0	14.1	198.3	10.3	198.6	8.4	199.5	3.3
200	200.0	0.0	198.4	10.4	197.6	11.5	198.2	8.0	198.0	10.4
500	200.0	0.0	196.7	14.4	**195.1**	16.3	**197.3**	16.6	198.5	8.9
1000	200.0	0.0	197.5	11.6	**195.3**	14.9	**196.8**	11.5	**197.1**	10.7
2000	200.0	0.0	**193.0**	16.3	**192.3**	20.7	**190.6**	26.0	**190.3**	24.4
5000	200.0	0.0	**189.3**	22.8	**187.6**	22.4	**188.5**	19.7	**189.3**	22.6

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1	1	1	3.24e-01
N/A	1	3.24e-01	1	3.24e-01
N/A	2.38e-02	3.24e-01	1	3.24e-01
N/A	1.59e-01	8.26e-02	1	3.24e-01
N/A	4.41e-02	8.26e-02	1.59e-01	1.59e-01
N/A	1.59e-01	4.41e-02	2.38e-02	4.41e-02
N/A	4.41e-02	**3.84e-03**	**7.05e-03**	8.26e-02
N/A	1.29e-02	**1.13e-03**	**3.84e-03**	**3.84e-03**
N/A	**1.86e-06**	**9.37e-05**	**6.12e-04**	**2.60e-05**
N/A	**9.42e-07**	**5.76e-08**	**6.72e-10**	**9.42e-07**

p-values

Figure A.3(e): `zenotravel` problem `p5`. The landmark extraction algorithm LM^{RHW} generated 12 nontrivial landmarks for this problem.

Appendix B: LAMP Probabilistically Interesting Benchmark Results

This appendix contains additional results from LAMP experiments. In each of the following figures, the top table reports the success rate (higher is better) of the solutions generated by LAMP over 75 runs. Underlined values indicate the best performing θ-value in the row. The bottom table reports p-values computed using a two-tailed Boschloo exact test comparing the success rate of LAMP with the success rate of standard UCT ($\theta = 0$) at the same number of rollouts. In both tables, boldfaced values indicate where LAMP significantly dominated standard UCT at the same number of rollouts, with $p < 0.0125$, the adjusted level using a Bonferroni adjustment of $\alpha_{stat} = 0.05/4$.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	0.8	more greedy 1
N/A	**4.35e-03**	1.79e-01	6.72e-02	**4.35e-03**
N/A	8.25e-01	2.79e-01	6.09e-01	4.25e-01
N/A	8.43e-02	9.44e-01	9.44e-01	9.44e-01
N/A	1	4.74e-01	8.02e-02	4.18e-02
N/A	5.60e-01	1	1.79e-01	2.93e-04
N/A	9.44e-01	9.44e-01	7.94e-01	5.00e-04
N/A	9.44e-01	1	7.91e-01	5.69e-03
N/A	6.64e-01	6.64e-01	1	5.00e-04
N/A	1	9.44e-01	1	4.00e-02
N/A	5.50e-01	1	7.91e-01	9.14e-05

p-values

Figure B.1(a): `exploding_blocksworld` problem `p1`. The landmark extraction algorithm LM^{RHW} generated 7 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	0.8	more greedy 1
N/A	1.85e-01	3.12e-01	8.16e-01	2.43e-01
N/A	9.44e-01	6.69e-01	7.94e-01	9.44e-01
N/A	1	1	1	9.44e-01
N/A	1	1	1	6.64e-01
N/A	1	1	1	5.50e-01
N/A	1	1	1	9.44e-01
N/A	1	1	1	7.91e-01
N/A	1	1	1	7.91e-01
N/A	1	1	1	1
N/A	1	1	1	1

p-values

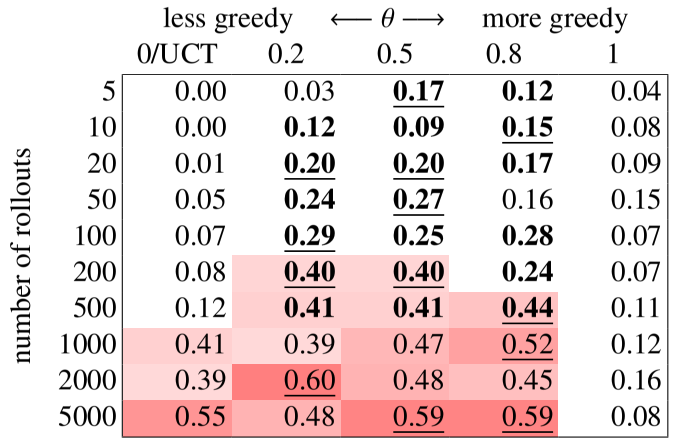
Figure B.1(b): `exploding_blocksworld` problem `p2`. The landmark extraction algorithm LM^{RHW} generated 4 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	0.8	more greedy 1
N/A	**5.69e-03**	**1.40e-03**	**1.40e-03**	**8.72e-03**
N/A	**8.45e-07**	**9.13e-05**	**4.98e-05**	**4.79e-03**
N/A	1.26e-02	2.81e-02	**1.19e-03**	1.26e-02
N/A	7.07e-01	2.47e-01	4.89e-01	4.89e-01
N/A	9.44e-01	7.80e-02	1.85e-01	4.05e-02
N/A	1	5.55e-01	3.08e-01	5.74e-01
N/A	9.44e-01	6.69e-01	3.65e-01	5.69e-03
N/A	3.65e-01	7.94e-01	2.89e-01	5.69e-03
N/A	5.60e-01	3.70e-01	3.70e-01	1.94e-02
N/A	9.44e-01	4.53e-01	6.69e-01	2.29e-01

p-values

Figure B.1(c): `exploding_blocksworld` problem `p3`. The landmark extraction algorithm LM^{RHW} generated 7 nontrivial landmarks for this problem.



Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	0.8	more greedy 1
N/A	4.37e-01	**9.15e-05**	**2.28e-03**	2.02e-01
N/A	**2.28e-03**	**9.73e-03**	**5.00e-04**	2.05e-02
N/A	**1.83e-04**	**1.83e-04**	**8.45e-04**	5.13e-02
N/A	**1.66e-03**	**5.00e-04**	5.13e-02	7.83e-02
N/A	**4.99e-04**	**2.56e-03**	**8.45e-04**	1
N/A	**7.11e-06**	**7.11e-06**	**1.10e-02**	9.44e-01
N/A	**9.15e-05**	**9.15e-05**	**2.66e-05**	9.44e-01
N/A	8.02e-01	5.77e-01	2.47e-01	9.15e-05
N/A	1.97e-02	3.04e-01	4.69e-01	3.63e-03
N/A	4.96e-01	7.11e-01	7.11e-01	1.59e-09

p-values

Figure B.1(d): `exploding_blocksworld` problem `p4`. The landmark extraction algorithm LM^{RHW} generated 9 nontrivial landmarks for this problem.

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	1.00	1.00	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00
200	1.00	1.00	1.00	1.00	1.00
500	1.00	1.00	1.00	1.00	1.00
1000	1.00	1.00	1.00	1.00	1.00
2000	1.00	1.00	1.00	1.00	1.00
5000	1.00	1.00	1.00	1.00	1.00

Average cost

0/UCT	less greedy ⟵ θ ⟶ more greedy			
	0.2	0.5	0.8	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.1(e): `exploding_blocksworld` problem `p5`. The landmark extraction algorithm LM^{RHW} generated 0 nontrivial landmarks for this problem. This problem instance is trivial—the initial state satisfies the goal condition.

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	0.00	0.04	0.03	0.03	0.03
10	0.00	0.01	0.04	0.03	0.00
20	0.00	0.05	0.03	**0.16**	0.05
50	0.00	0.08	**0.13**	0.07	**0.11**
100	0.00	0.08	0.07	0.05	**0.09**
200	0.01	0.07	0.07	0.09	0.09
500	0.00	**0.09**	**0.12**	**0.12**	**0.19**
1000	0.00	**0.20**	**0.11**	**0.19**	**0.13**
2000	0.07	**0.31**	0.13	0.20	**0.23**
5000	0.09	**0.27**	**0.28**	**0.37**	**0.28**

Average cost

0/UCT	less greedy ⟵ θ ⟶ more greedy			
	0.2	0.5	0.8	1
N/A	2.02e-01	4.37e-01	4.37e-01	4.37e-01
N/A	9.44e-01	2.02e-01	4.37e-01	1
N/A	1.04e-01	4.37e-01	**2.14e-04**	1.04e-01
N/A	2.05e-02	**9.83e-04**	4.43e-02	**4.63e-03**
N/A	2.05e-02	4.43e-02	1.04e-01	**9.73e-03**
N/A	1.79e-01	1.79e-01	5.13e-02	5.13e-02
N/A	**9.73e-03**	**2.28e-03**	**2.28e-03**	**4.98e-05**
N/A	**2.05e-05**	**4.63e-03**	**4.98e-05**	**9.83e-04**
N/A	**2.34e-04**	2.29e-01	2.32e-02	**8.72e-03**
N/A	**8.72e-03**	**5.69e-03**	**9.13e-05**	**5.69e-03**

p-values

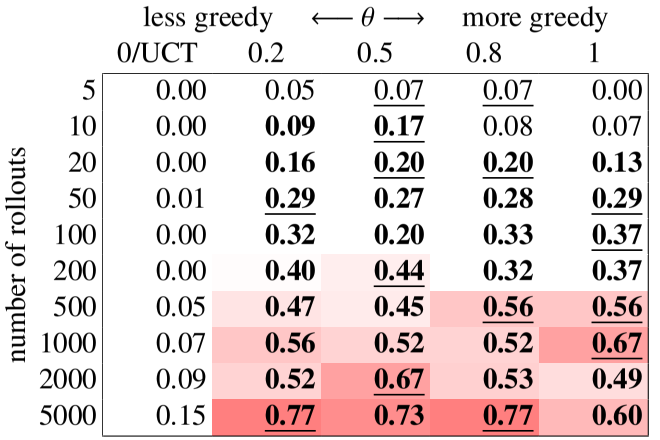
Figure B.1(f): `exploding_blocksworld` problem `p6`. The landmark extraction algorithm LM^{RHW} generated 16 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	9.44e-01	1	9.44e-01	1
N/A	9.44e-01	4.37e-01	1	4.37e-01
N/A	2.02e-01	1.04e-01	9.44e-01	4.37e-01
N/A	2.02e-01	4.37e-01	4.37e-01	2.02e-01
N/A	4.37e-01	4.37e-01	9.44e-01	1
N/A	1	9.44e-01	9.44e-01	9.44e-01
N/A	9.44e-01	1	1	1
N/A	1	9.44e-01	2.02e-01	9.44e-01
N/A	9.44e-01	1	1	4.37e-01
N/A	9.44e-01	9.44e-01	9.44e-01	4.37e-01

p-values

Figure B.1(g): `exploding_blocksworld` problem `p7`. The landmark extraction algorithm LM^{RHW} generated 17 nontrivial landmarks for this problem.

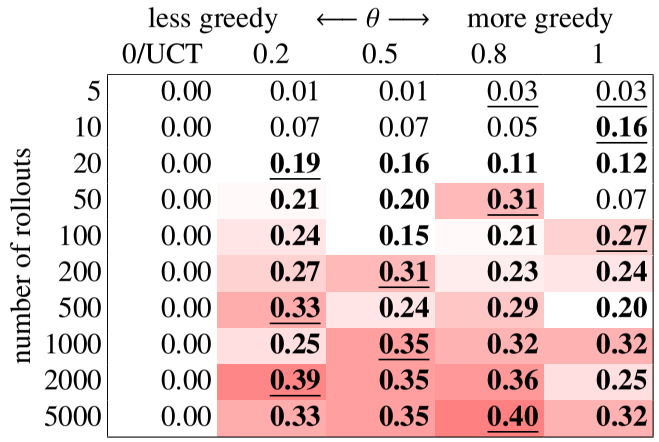


Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1.04e-01	4.43e-02	4.43e-02	1
N/A	**9.73e-03**	**9.15e-05**	2.05e-02	4.43e-02
N/A	**2.14e-04**	**2.05e-05**	**2.05e-05**	**9.83e-04**
N/A	**8.93e-07**	**4.39e-06**	**1.94e-06**	**8.93e-07**
N/A	**1.51e-08**	**2.05e-05**	**6.34e-09**	**4.04e-10**
N/A	**6.07e-11**	**4.27e-12**	**1.51e-08**	**4.04e-10**
N/A	**8.67e-09**	**1.95e-08**	**2.13e-11**	**2.13e-11**
N/A	**1.55e-10**	**1.72e-09**	**1.72e-09**	**8.90e-14**
N/A	**3.56e-08**	**2.91e-12**	**1.61e-08**	**1.80e-07**
N/A	**8.45e-13**	**1.18e-11**	**8.45e-13**	**5.02e-08**

p-values

Figure B.1(h): `exploding_blocksworld` problem `p8`. The landmark extraction algorithm LM^{RHW} generated 13 nontrivial landmarks for this problem.



Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	9.44e-01	9.44e-01	4.37e-01	4.37e-01
N/A	4.43e-02	4.43e-02	1.04e-01	**2.14e-04**
N/A	**4.98e-05**	**2.14e-04**	**4.63e-03**	**2.28e-03**
N/A	**8.80e-06**	**2.05e-05**	**3.52e-08**	4.43e-02
N/A	**1.78e-06**	**5.00e-04**	**8.80e-06**	**3.92e-07**
N/A	**3.92e-07**	**3.52e-08**	**3.79e-06**	**1.78e-06**
N/A	**6.34e-09**	**1.78e-06**	**8.05e-08**	**2.05e-05**
N/A	**8.37e-07**	**2.59e-09**	**1.51e-08**	**1.51e-08**
N/A	**1.55e-10**	**2.59e-09**	**1.03e-09**	**8.37e-07**
N/A	**6.34e-09**	**2.59e-09**	**6.07e-11**	**1.51e-08**

p-values

Figure B.1(i): `exploding_blocksworld` problem `p9`. The landmark extraction algorithm LM^{RHW} generated 17 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	1.04e-01	7.55e-01	2.39e-01	2.39e-01
N/A	**5.69e-03**	1.94e-02	**2.89e-04**	**3.63e-03**
N/A	2.14e-01	9.44e-01	4.16e-01	9.44e-01
N/A	1.66e-01	5.13e-02	2.14e-01	8.20e-01
N/A	9.44e-01	2.76e-01	9.77e-02	7.24e-02
N/A	7.24e-01	6.11e-01	8.29e-01	4.31e-01
N/A	4.19e-01	4.19e-01	1.28e-01	6.03e-01
N/A	9.44e-01	4.29e-01	4.29e-01	9.44e-01
N/A	4.29e-01	1.37e-01	7.36e-02	4.29e-01
N/A	9.44e-01	5.14e-01	8.26e-01	4.14e-01

p-values

Figure B.2(a): `tireworld` problem `p1`. The landmark extraction algorithm LM^{RHW} generated 5 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	2.29e-01	2.29e-01	2.29e-01	2.29e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(b): `tireworld` problem `p2`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1.38e-01	7.68e-02	7.68e-02	5.60e-02
N/A	1	6.73e-01	4.48e-01	5.55e-01
N/A	7.94e-01	6.64e-01	6.64e-01	6.64e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(c): `tireworld` problem `p3`. The landmark extraction algorithm LM^{RHW} generated 2 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	3.89e-01	3.89e-01	4.82e-01	3.08e-01
N/A	1	9.44e-01	1	7.97e-01
N/A	9.44e-01	9.44e-01	9.44e-01	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(d): `tireworld` problem `p4`. The landmark extraction algorithm LM^{RHW} generated 2 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	4.53e-01	9.44e-01	3.59e-01	5.60e-01
N/A	6.64e-01	9.44e-01	9.44e-01	6.64e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(e): `tireworld` problem `p5`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	5.50e-01	6.69e-01	5.50e-01	6.69e-01
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01
N/A	9.44e-01	1	9.44e-01	9.44e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(f): `tireworld` problem `p6`. The landmark extraction algorithm LM^{RHW} generated 2 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1.71e-01	1.63e-02	2.43e-02	3.54e-02
N/A	6.84e-01	8.07e-01	5.69e-01	2.89e-01
N/A	6.69e-01	5.50e-01	7.97e-01	5.50e-01
N/A	9.44e-01	1	1	1
N/A	9.44e-01	7.91e-01	7.91e-01	9.44e-01
N/A	6.64e-01	6.64e-01	6.64e-01	6.64e-01
N/A	7.91e-01	7.91e-01	7.91e-01	1
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01
N/A	1	9.44e-01	1	1
N/A	1	1	1	1

p-values

Figure B.2(g): `tireworld` problem `p7`. The landmark extraction algorithm LM^{RHW} generated 2 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	0.8	more greedy 1
N/A	2.95e-02	1.18e-01	1.18e-01	6.16e-02
N/A	9.44e-01	9.44e-01	8.02e-01	9.44e-01
N/A	8.00e-01	9.44e-01	4.48e-01	5.55e-01
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(h): `tireworld` problem `p8`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	0.8	more greedy 1
N/A	7.05e-01	**1.18e-02**	3.60e-02	1.64e-01
N/A	9.44e-01	9.44e-01	8.23e-01	3.24e-01
N/A	3.20e-01	4.01e-01	9.44e-01	1.92e-01
N/A	6.81e-01	6.81e-01	6.81e-01	9.44e-01
N/A	3.59e-01	3.59e-01	6.77e-01	3.59e-01
N/A	9.44e-01	5.50e-01	1	9.44e-01
N/A	9.44e-01	7.91e-01	7.91e-01	7.91e-01
N/A	9.44e-01	1	1	1
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01

p-values

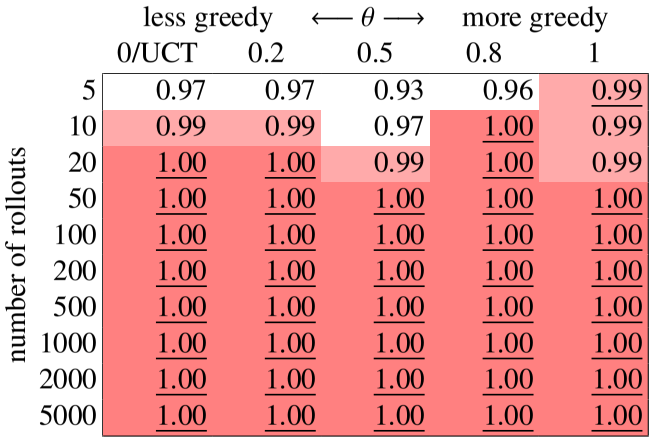
Figure B.2(i): `tireworld` problem `p9`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	4.00e-02	4.00e-02	4.00e-02	4.00e-02
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(j): `tireworld` problem `p10`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.



Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	1	6.73e-01	9.44e-01	9.44e-01
N/A	1	9.44e-01	9.44e-01	1
N/A	1	9.44e-01	1	9.44e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(k): `tireworld` problem `p11`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT	0.2	0.5	0.8	1
5	0.88	1.00	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00
200	1.00	1.00	1.00	1.00	1.00
500	1.00	1.00	1.00	1.00	1.00
1000	1.00	1.00	1.00	1.00	1.00
2000	1.00	1.00	1.00	1.00	1.00
5000	1.00	1.00	1.00	1.00	1.00

Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	1.79e-01	1.79e-01	1.79e-01	1.79e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(l): `tireworld` problem `p12`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

less greedy ⟵ θ ⟶ more greedy

number of rollouts	0/UCT	0.2	0.5	0.8	1
5	0.47	**0.73**	**0.73**	**0.75**	0.69
10	0.83	0.88	0.91	0.95	0.96
20	0.89	0.99	0.91	0.95	0.93
50	0.97	0.99	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00
200	1.00	1.00	1.00	1.00	1.00
500	1.00	1.00	1.00	1.00	1.00
1000	1.00	1.00	1.00	1.00	1.00
2000	1.00	1.00	1.00	1.00	1.00
5000	1.00	1.00	1.00	1.00	1.00

Average cost

less greedy ⟵ θ ⟶ more greedy

0/UCT	0.2	0.5	0.8	1
N/A	**4.77e-03**	**4.77e-03**	**3.06e-03**	1.63e-02
N/A	5.88e-01	3.94e-01	1.85e-01	1.38e-01
N/A	2.89e-01	9.44e-01	5.69e-01	6.84e-01
N/A	9.44e-01	7.91e-01	7.91e-01	7.91e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

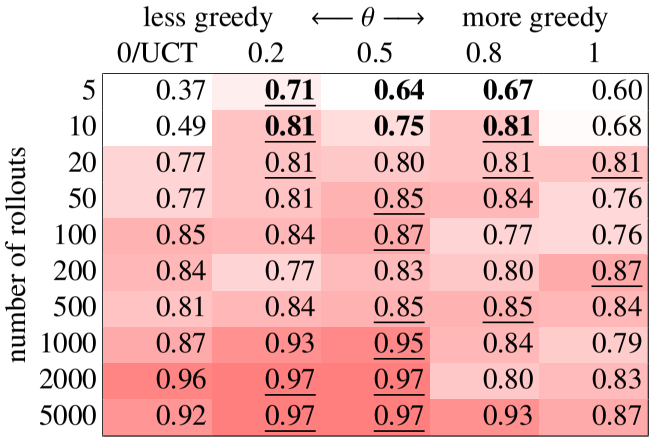
Figure B.2(m): `tireworld` problem `p13`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	**8.53e-04**	**2.47e-06**	**3.09e-05**	**8.05e-08**
N/A	1.62e-01	1.62e-01	1.22e-01	7.15e-01
N/A	6.91e-01	9.44e-01	4.74e-01	5.77e-01
N/A	2.29e-01	2.94e-01	2.29e-01	1.79e-01
N/A	9.44e-01	9.44e-01	9.44e-01	9.44e-01
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1
N/A	1	1	1	1

p-values

Figure B.2(n): `tireworld` problem `p14`. The landmark extraction algorithm LM^{RHW} generated 1 nontrivial landmarks for this problem.



Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	**3.15e-04**	**3.67e-03**	**1.45e-03**	1.31e-02
N/A	**8.05e-04**	**7.70e-03**	**8.05e-04**	4.96e-02
N/A	7.07e-01	8.21e-01	7.07e-01	7.07e-01
N/A	7.07e-01	4.08e-01	4.99e-01	9.44e-01
N/A	9.44e-01	9.44e-01	4.08e-01	3.30e-01
N/A	4.99e-01	9.44e-01	7.03e-01	8.14e-01
N/A	8.17e-01	7.01e-01	7.01e-01	8.17e-01
N/A	4.74e-01	3.80e-01	8.14e-01	4.05e-01
N/A	9.44e-01	9.44e-01	7.68e-02	1.38e-01
N/A	5.60e-01	5.60e-01	9.44e-01	5.77e-01

p-values

Figure B.2(o): `tireworld` problem `p15`. The landmark extraction algorithm LM^{RHW} generated 3 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	2.97e-03	1.86e-03	1.07e-02	7.92e-05
N/A	1.04e-02	1.54e-02	6.31e-02	4.49e-03
N/A	1.09e-05	6.88e-05	2.87e-06	1.45e-02
N/A	1.39e-05	9.46e-07	2.10e-08	2.23e-07
N/A	3.63e-03	1.39e-05	4.63e-07	2.66e-05
N/A	9.44e-01	1.89e-06	4.98e-05	1.31e-02
N/A	1	2.29e-01	2.66e-05	4.98e-05
N/A	1	1	7.18e-06	7.18e-06
N/A	1	1	9.46e-07	1.64e-04
N/A	1	1	9.14e-05	4.63e-07

p-values

Figure B.3(a): `triangle_tireworld` problem `p1`. The landmark extraction algorithm LM^{RHW} generated 2 nontrivial landmarks for this problem.

Average cost

0/UCT	less greedy 0.2	⟵ θ ⟶ 0.5	more greedy 0.8	1
N/A	2.50e-08	1.07e-06	5.67e-06	9.15e-05
N/A	2.50e-08	2.47e-06	1.07e-06	4.38e-07
N/A	9.03e-12	8.94e-18	2.18e-16	9.03e-12
N/A	4.79e-15	1.74e-17	9.96e-23	1.89e-13
N/A	2.51e-12	5.13e-18	1.19e-21	9.45e-21
N/A	9.28e-09	4.28e-23	2.35e-21	8.00e-14
N/A	2.51e-12	2.56e-17	2.35e-21	8.52e-26
N/A	2.88e-10	8.45e-13	6.05e-24	4.28e-23
N/A	7.06e-10	2.24e-14	6.81e-25	2.56e-17
N/A	9.28e-09	8.45e-13	5.38e-20	2.78e-19

p-values

Figure B.3(b): `triangle_tireworld` problem `p2`. The landmark extraction algorithm LM^{RHW} generated 4 nontrivial landmarks for this problem.

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	0.17	0.12	0.21	0.17	0.11
10	0.36	0.07	0.08	0.16	0.08
20	0.45	0.12	0.11	0.07	0.08
50	0.53	0.07	0.08	0.08	0.07
100	0.60	0.11	0.07	0.04	0.11
200	0.68	0.27	0.13	0.09	0.08
500	0.59	0.19	0.28	0.07	0.23
1000	0.61	0.35	0.17	0.15	0.08
2000	0.75	0.25	0.20	0.08	0.16
5000	0.60	0.27	0.24	0.15	0.09

Average cost

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	0.08	0.04	0.05	0.01	0.01
10	0.13	0.04	0.01	0.07	0.05
20	0.07	0.04	0.07	0.03	0.05
50	0.09	0.05	0.05	0.04	0.04
100	0.13	0.03	0.01	0.04	0.03
200	0.15	0.05	0.07	0.00	0.04
500	0.12	0.08	0.07	0.04	0.04
1000	0.07	0.09	0.09	0.04	0.04
2000	0.08	0.15	0.11	0.08	0.07
5000	0.08	0.03	0.08	0.03	0.04

p-values

Figure B.3(c): `triangle_tireworld` problem `p3`. The landmark extraction algorithm LM^{RHW} generated 4 nontrivial landmarks for this problem.

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	0.05	0.04	0.04	0.01	0.08
10	0.05	0.04	0.01	0.04	0.03
20	0.09	0.11	0.04	0.04	0.04
50	0.11	0.03	0.03	0.05	0.07
100	0.15	0.07	0.04	0.05	0.01
200	0.12	0.01	0.04	0.12	0.01
500	0.11	0.13	0.08	0.04	0.05
1000	0.15	0.08	0.07	0.11	0.07
2000	0.08	0.13	0.09	0.09	0.05
5000	0.12	0.13	0.13	0.04	0.07

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	9.44e-01	9.44e-01	3.04e-01	6.60e-01
N/A	9.44e-01	3.04e-01	9.44e-01	6.21e-01
N/A	9.44e-01	2.89e-01	2.89e-01	2.89e-01
N/A	7.68e-02	7.68e-02	3.12e-01	5.06e-01
N/A	1.53e-01	4.00e-02	7.83e-02	3.64e-03
N/A	1.31e-02	1.04e-01	1	1.31e-02
N/A	7.24e-01	7.24e-01	1.79e-01	3.12e-01
N/A	2.61e-01	1.53e-01	5.44e-01	1.53e-01
N/A	3.59e-01	9.44e-01	9.44e-01	6.60e-01
N/A	9.44e-01	9.44e-01	1.04e-01	3.58e-01

p-values

Figure B.3(d): `triangle_tireworld` problem `p4`. The landmark extraction algorithm LM^{RHW} generated 4 nontrivial landmarks for this problem.

number of rollouts	less greedy ⟵ θ ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	0.01	0.00	0.00	0.03	0.01
10	0.04	0.00	0.00	0.01	0.00
20	0.01	0.03	0.00	0.01	0.03
50	0.03	0.00	0.00	0.01	0.01
100	0.04	0.00	0.01	0.00	0.01
200	0.03	0.01	0.01	0.00	0.00
500	0.00	0.01	0.04	0.01	0.01
1000	0.03	0.01	0.00	0.01	0.01
2000	0.01	0.03	0.03	0.01	0.00
5000	0.01	0.03	0.03	0.01	0.01

Average cost

less greedy ⟵ θ ⟶ more greedy				
0/UCT	0.2	0.5	0.8	1
N/A	9.44e-01	9.44e-01	9.44e-01	1
N/A	2.02e-01	1.04e-01	3.04e-01	1.04e-01
N/A	9.44e-01	9.44e-01	1	9.44e-01
N/A	4.37e-01	4.37e-01	9.44e-01	9.44e-01
N/A	2.02e-01	5.31e-01	2.02e-01	5.31e-01
N/A	9.44e-01	9.44e-01	4.37e-01	4.37e-01
N/A	9.44e-01	2.02e-01	9.44e-01	9.44e-01
N/A	9.44e-01	4.37e-01	9.44e-01	9.44e-01
N/A	9.44e-01	9.44e-01	1	9.44e-01
N/A	9.44e-01	9.44e-01	1	1

p-values

Figure B.3(e): `triangle_tireworld` problem `p5`. The landmark extraction algorithm LM^{RHW} generated 4 nontrivial landmarks for this problem.

Appendix C: Papers and Presentations

- [26] E. Conway, D. Porfirio, **D. H. Chan**, M. Roberts, and L. M. Hiatt, "POrTAL: Plan-orchestrated tree assembly for lookahead," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2026)*, Pittsburgh, PA, USA, September 27–October 1, 2026, under review.

- [22] **D. H. Chan**, M. Roberts, and D. S. Nau, "Probabilistic hierarchical goal network planning with UCT," in *Proceedings of the 40th AAAI Conference on Artificial Intelligence (AAAI 2026)*, Singapore, January 20–27, 2026, pp. 36189–36196.

- [19] **D. H. Chan**, M. Roberts, and D. S. Nau, "Landmark-assisted Monte Carlo planning," presented at *the Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP 2025)*, Melbourne, Australia, November 11, 2025.

- [21] **D. H. Chan**, M. Roberts, and D. S. Nau, "Probabilistic hierarchical goal network planning: Preliminary results," in *Proceedings of the 8th ICAPS Workshop on Hierarchical Planning (HPlan 2025)*, Melbourne, Australia, November 10, 2025, pp. 1–5.

- [17] **D. H. Chan**, "Hierarchical goal decomposition for probabilistic planning," presented at *the ICAPS 2025 Doctoral Consortium*, Melbourne, Australia, November 9, 2025.

- [20] **D. H. Chan**, M. Roberts, and D. S. Nau, "Landmark-assisted Monte Carlo planning," presented at *the 35th International Conference on Automated Planning and Scheduling (ICAPS 2025)*, Melbourne, Australia, November 9–14, 2025.

- [18] **D. H. Chan**, M. Roberts, and D. S. Nau, "Landmark-assisted Monte Carlo planning," in *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, Bologna, Italy, October 25–30, 2025, pp. 4710–4717.

- [5] J. Baldwin, L. Birnbaum, **D. H. Chan**, N. Denisenko, D. S. Nau, J. N. Paredes, C. Pulice and G. I. Simari, V. S. Subrahmanian, and R. Waltzman, "The impact of strategic communication in coopetitive multiagent settings," *IEEE Transactions on Computational Social Systems*, vol. 12, no. 5, pp. 2915–2929, 2025.

- [25] E. Conway, D. Porfirio, **D. H. Chan**, M. Roberts, and L. M. Hiatt, "Lightweight probabilistic planning with macro actions," presented at *the 2nd AAAI Fall Symposium on Unifying Representations for Robot Application Development (UR-RAD 2024)*, Arlington, VA, USA, November 7–9, 2024.

- [62] T. Rabbani, J. Su, X. Liu, **D. H. Chan**, G. Sangston, and F. Huang, "Fast evaluation of multilinear operations in convolutional tensorial neural networks," presented at *the 3rd Workshop on Seeking Low-Dimensionality in Deep Neural Networks (SLow-DNN 2023)*, Abu Dhabi, United Arab Emirates, January 3–7, 2023.

Bibliography

[1] R. Alford, G. Behnke, D. Höller, P. Bercher, S. Biundo, and D. W. Aha, "Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems," in *Proc. 26th Int. Conf. Automated Planning and Scheduling (ICAPS 2016)*, London, UK, June 12–17, 2016, pp. 20–28.

[2] R. Alford, V. Shivashankar, M. Roberts, J. Frank, and D. W. Aha, "Hierarchical planning: Relating task and goal decomposition with task sharing," in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI 2016)*, New York, NY, USA, July 9–15, 2016, pp. 3022–3029.

[3] R. Alford, U. Kuter, and D. S. Nau, "Translating HTNs to PDDL: A small amount of domain knowledge can go a long way," in *Proc. 21st Int. Joint Conf. Artif. Intell. (IJCAI 2009)*, Pasadena, CA, USA, July 11–17, 2009, pp. 1629–1634.

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2–3, pp. 235–256, 2002.

[5] J. Baldwin et al., "The impact of strategic communication in coopetitive multiagent settings," *IEEE Transactions on Computational Social Systems*, vol. 12, no. 5, pp. 2915–2929, 2025.

[6] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artif. Intell.*, vol. 72, no. 1–2, pp. 81–138, 1995.

[7] G. Behnke, D. Höller, and S. Biundo, "totSAT - totally-ordered hierarchical planning through SAT," in *Proc. 32nd AAAI Conf. Artif. Intell. (AAAI 2018)*, New Orleans, LA, USA, February 2–7, 2018, pp. 6110–6118.

[8] G. Behnke, D. Höller, A. Schmid, P. Bercher, and S. Biundo, "On succinct groundings of HTN planning problems," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI 2020)*, New York, NY, USA, February 7–12, 2020, pp. 9775–9784.

[9] G. Behnke, F. Pollitt, D. Höller, P. Bercher, and R. Alford, "Making translations to classical planning competitive with other HTN planners," in *Proc. 36th AAAI Conf. Artif. Intell. (AAAI 2022)*, Virtual Event, February 22 – March 1, 2022, pp. 9687–9697.

[10] P. Bercher, G. Behnke, D. Höller, and S. Biundo, "An admissible HTN planning heuristic," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI 2017)*, Melbourne, Australia, August 19–25, 2017, pp. 480–488.

[11] A. Bit-Monnot, D. E. Smith, and M. Do, "Delete-free reachability analysis for temporal and hierarchical planning," in *Proc. 22nd Eur. Conf. Artif. Intell. (ECAI 2016)*, The Hague, The Netherlands, August 29–September 2, vol. 285, 2016, pp. 1698–1699.

[12] B. Bonet and H. Geffner, "Planning as heuristic search," *Artif. Intell.*, vol. 129, no. 1–2, pp. 5–33, 2001.

[13] B. Bonet, G. Loerincs, and H. Geffner, "A robust and fast action selection mechanism for planning," in *Proc. 14th Nat. Conf. Artif. Intell. (AAAI 1997)*, Providence, RI, USA, July 27–31, 1997, pp. 714–719.

[14] C. Browne et al., "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[15] C. Büchner, R. Christen, S. Eriksson, and T. Keller, "Hitting set heuristics for overlapping landmarks in satisficing planning," in *Proc. 17th Int. Symp. Combinatorial Search (SoCS 2024)*, Kananaskis, Canada, June 6–8, 2024, pp. 198–202.

[16] T. Bylander, "The computational complexity of propositional STRIPS planning," *Artif. Intell.*, vol. 69, no. 1–2, pp. 165–204, 1994.

[17] D. H. Chan, *Hierarchical goal decomposition for probabilistic planning*, presented at the ICAPS 2025 Doctoral Consortium, Melbourne, Australia, November 9, 2025.

[18] D. H. Chan, M. Roberts, and D. S. Nau, "Landmark-assisted monte carlo planning," in *Proc. 28th Eur. Conf. Artif. Intell. (ECAI 2025)*, Bologna, Italy, October 25–30, 2025, pp. 4710–4717.

[19] D. H. Chan, M. Roberts, and D. S. Nau, *Landmark-assisted Monte Carlo planning*, presented at the Workshop on Heuristics and Search for Domain-independent Planning (HSDIP 2025), Melbourne, Australia, November 11, 2025.

[20] D. H. Chan, M. Roberts, and D. S. Nau, *Landmark-assisted Monte Carlo planning*, presented at the 35th Int. Conf. Automated Planning and Scheduling (ICAPS 2025), Melbourne, Australia, November 9–14, 2025.

[21] D. H. Chan, M. Roberts, and D. S. Nau, "Probabilistic hierarchical goal network planning: Preliminary results," in *Proc. 8th ICAPS Workshop on Hierarchical Planning (HPlan 2025)*, Melbourne, Australia, November 10, 2025, pp. 1–5.

[22] D. H. Chan, M. Roberts, and D. S. Nau, "Probabilistic hierarchical goal network planning with uct," in *Proc. 40th AAAI Conf. Artif. Intell. (AAAI 2026)*, Singapore, January 20–27, 2026, pp. 36 189–36 196.

[23] D. Z. Chen and P. Bercher, "Fully observable nondeterministic HTN planning – formalisation and complexity results," in *Proc. 31st Int. Conf. Automated Planning and Scheduling (ICAPS 2021)*, Guangzhou, China (virtual), August 2–13, 2021, pp. 74–84.

[24] D. Z. Chen and P. Bercher, "Flexible FOND HTN planning: A complexity analysis," in *Proc. 32nd Int. Conf. Automated Planning and Scheduling (ICAPS 2022)*, Singapore (virtual), June 13–24, 2022, pp. 26–34.

[25] E. Conway, D. Porfirio, D. H. Chan, M. Roberts, and L. M. Hiatt, *Lightweight probabilistic planning with macro actions*, presented at the 2nd AAAI Fall Symp. on Unifying Representations for Robot Application Develop. (UR-RAD 2024), Arlington, VA, USA, November 7–9, 2024.

[26] E. Conway, D. Porfirio, D. H. Chan, M. Roberts, and L. M. Hiatt, "POrTAL: Plan-orchestrated tree assembly for lookahead," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst. (IROS 2026)*, Pittsburgh, PA, USA, September 27–October 1, 2026, under review.

[27] G. N. Crispino, V. Freire, and K. V. Delgado, "Monte carlo tree search algorithm for SSPs under the GUBS criterion," *CLEI Electron. J.*, vol. 27, no. 3, 5:1–5:18, 2024.

[28] A. Demir, E. Çilden, and F. Polat, "Landmark based guidance for reinforcement learning agents under partial observability," *Int. J. Mach. Learn. Cybern.*, vol. 14, no. 4, pp. 1543–1563, 2023.

[29] K. Erol, J. A. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *Proc. 12th Nat. Conf. Artif. Intell. (AAAI 1994)*, Seattle, WA, USA, July 31–August 4, 1994, pp. 1123–1128.

[30] K. Erol, J. A. Hendler, and D. S. Nau, "UMCP: A sound and complete procedure for hierarchical task-network planning," in *Proc. 2nd Int. Conf. Artif. Intell. Planning Systems (AIPS 1994)*, Chicago, IL, USA, June 13–15, 1994, pp. 249–254.

[31] Z. Feldman and C. Domshlak, "Simple regret optimization in online planning for markov decision processes," *J. Artif. Intell. Res.*, vol. 51, pp. 165–205, 2014.

[32] R. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, no. 3/4, pp. 189–208, 1971.

[33] V. Freire and K. V. Delgado, "GUBS: a utility-based semantic for goal-directed markov decision processes," in *Proc. 16th Conf. Auton. Agents and Multiagent Syst. (AAMAS 2017)*, São Paulo, Brazil, May 8–12, 2017, pp. 741–749.

[34] V. Freire, K. V. Delgado, and W. A. S. Reis, "An exact algorithm to make a trade-off between cost and probability in SSPs," in *Proc. 19th Int. Conf. Automated Planning and Scheduling (ICAPS 2019)*, Berkeley, CA, USA, July 11–15, 2019, pp. 146–154.

[35] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artif. Intell.*, vol. 175, no. 11, pp. 1856–1875, 2011.

[36] A. Gerevini, B. Bonet, and R. Givan, *The 5th international planning competition*, 2006.

[37] M. Ghallab, D. S. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge, U.K.: Cambridge Univ. Press, 2016.

[38] M. Ghallab, D. S. Nau, and P. Traverso, *Acting, Planning, and Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2025.

[39] E. A. Hansen and S. Zilberstein, "LAO*: A heuristic search algorithm that finds solutions with loops," *Artif. Intell.*, vol. 129, no. 1–2, pp. 35–62, 2001.

[40] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, *An Introduction to the Planning Domain Definition Language* (Synthesis Lectures on Artificial Intelligence and Machine Learning). San Rafael, CA, USA: Morgan & Claypool, 2019.

[41] M. Helmert and C. Domshlak, "Landmarks, critical paths and abstractions: What's the difference anyway?" In *Proc. 19th Int. Conf. Automated Planning and Scheduling (ICAPS 2019)*, Thessaloniki, Greece, September 19–23, 2009, pp. 162–169.

[42] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, 2001.

[43] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered landmarks in planning," *J. Artif. Intell. Res.*, vol. 22, pp. 215–278, 2004.

[44] D. Höller, "The TOAD system for totally ordered HTN planning," *J. Artif. Intell. Res.*, vol. 80, pp. 613–663, 2024.

[45] D. Höller, P. Bercher, G. Behnke, and S. Biundo, "HTN planning as heuristic progression search," *J. Artif. Intell. Res.*, vol. 67, pp. 835–880, 2020.

[46] T. Keller and P. Eyerich, "PROST: probabilistic planning based on UCT," in *Proc. 22nd Int. Conf. Automated Planning and Scheduling (ICAPS 2012)*, São Paulo, Brazil, June 25–29, 2012, pp. 119–127.

[47] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. ECML 2006*, 2006, pp. 282–293.

[48] A. Kolobov, Mausam, and D. S. Weld, "A theory of goal-oriented MDPs with dead ends," in *Proc. 28th Conf. Uncertainty in Artif. Intell. (UAI 2012)*, Catalina Island, CA, USA, August 14–18, 2012, pp. 438–447.

[49] R. Li, D. S. Nau, M. Roberts, and M. Fine-Morris, "Automatically learning HTN methods from landmarks," in *37th Int. FLAIRS Conf. Proc. (FLAIRS 2024)*, Miramar Beach, Florida, USA, May 19–21, 2024, pp. 146–154.

[50] I. Little and S. Thiebaux, "Probabilistic planning vs. replanning," in *Proc. 17th Int. Conf. Automated Planning and Scheduling (ICAPS 2007)*, Providence, RI, USA, September 22–26, 2007, pp. 1–10.

[51] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proc. 18th Int. Conf. on Mach. Learn. (ICML 2001)*, Williamstown, MA, USA, June 28–July 1, 2001, pp. 361–368.

[52] I. Menache, S. Mannor, and N. Shimkin, "Q-Cut - dynamic discovery of sub-goals in reinforcement learning," in *Proc. 13th Eur. Conf. Mach. Learn. (ECML 2002)*, Helsinki, Finland, August 19–23, 2002, pp. 295–306.

[53] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, "The SHOP planning system," *AI Mag.*, vol. 22, no. 3, pp. 91–64, 2001.

[54] D. S. Nau et al., "SHOP2: an HTN planning system," *J. Artif. Intell. Res.*, vol. 20, pp. 379–404, 2003.

[55] M. Painter, B. Lacerda, and N. Hawes, "Convex hull Monte-Carlo tree-search," in *Proc 30th Int. Conf. Automated Planning and Scheduling (ICAPS 2020)*, Nancy, France, October 26–30, 2020, pp. 217–225.

[56] S. Patra, J. Mason, M. Ghallab, D. S. Nau, and P. Traverso, "Deliberative acting, planning and learning with hierarchical operational models," *Artif. Intell.*, vol. 299, p. 103 523, 2021.

[57] S. Patra, J. Mason, A. Kumar, M. Ghallab, P. Traverso, and D. S. Nau, "Integrating acting, planning, and learning in hierarchical operational models," in *Proc 30th Int. Conf. Automated Planning and Scheduling (ICAPS 2020)*, Nancy, France, October 26–30, 2020, pp. 478–487.

[58] E. P. D. Pednault, "ADL: exploring the middle ground between STRIPS and the situation calculus," in *Proc. 1st Int. Conf. Princ. Knowl. Representation and Reasoning (KR 1989)*, Toronto, Canada, May 15–18, 1989, pp. 324–332.

[59] F. Pommerening and M. Helmert, "Incremental LM-Cut," in *Proc. 23rd Int. Conf. Automated Planning and Scheduling*, Rome, Italy, June 10–14, 2013, pp. 162–170.

[60] J. Porteous, L. Sebastia, and J. Hoffmann, "On the extraction, ordering, and usage of landmarks in planning," in *Proc. 6th Eur. Conf. Planning (ECP 2001)*, Toledo, Spain, September 12–14, 2001, pp. 37–48.

[61] G. Quenard, D. Pellier, and H. Fiorino, "SibylSat: Using SAT as an oracle to perform a greedy search on TOHTN planning," in *Proc. 27th Eur. Conf. Artif. Intell. (ECAI 2024)*, ser. Frontiers in Artif. Intell. and Applications, Santiago de Compostela, Spain, October 19–24, vol. 392, 2024, pp. 4157–4164.

[62] T. Rabbani, J. Su, X. Liu, D. H. Chan, G. Sangston, and F. Huang, *Fast evaluation of multilinear operations in convolutional tensorial neural networks*, presented at 3rd Workshop on Seeking Low-Dimensionality in Deep Neural Networks (SLow-DNN 2023), Abu Dhabi, United Arab Emirates, January 3–7, 2023.

[63] R. Ramesh, M. Tomar, and B. Ravindran, "Successor options: An option discovery framework for reinforcement learning," in *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI 2019)*, Macau, China, August 10–16, 2019, pp. 3304–3310.

[64] S. Richter, M. Helmert, and M. Westphal, "Landmarks revisited," in *Proc. 23rd AAAI Conf. Artif. Intell. (AAAI 2008)*, Chicago, IL, USA, July 13–17, 2008, pp. 975–982.

[65] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *J. Artif. Intell. Res.*, vol. 39, pp. 127–177, 2010.

[66] E. Scala, P. Haslum, D. Magazzeni, and S. Thiébaux, "Landmarks for numeric planning problems," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI 2017)*, Melbourne, Australia, August 19–25, 2017, pp. 4384–4390.

[67] D. Schreiber, "Lilotane: A lifted SAT-based approach to hierarchical planning," *J. Artif. Intell. Res.*, vol. 70, pp. 1117–1181, 2021.

[68] V. Shivashankar, R. Alford, and D. W. Aha, "Incorporating domain-independent planning heuristics in hierarchical planning," in *Proc. 31st AAAI Conf. Artif. Intell. (AAAI 2017)*, San Francisco, CA, USA, February 4–9, 2017, pp. 3658–3664.

[69] V. Shivashankar, R. Alford, M. Roberts, and D. W. Aha, "Cost-optimal algorithms for planning with procedural control knowledge," in *Proc. 22nd Eur. Conf. Artif. Intell. (ECAI 2016)*, The Hague, The Netherlands, August 29–September 2, vol. 285, 2016, pp. 1702–1703.

[70] V. Shivashankar, R. Alford, U. Kuter, and D. S. Nau, "The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning," in *Proc. 23rd Int. Joint Conf. Artif. Intell. (IJCAI 2013)*, Beijing, China, August 3–9, 2013, pp. 2380–2386.

[71] V. Shivashankar, U. Kuter, D. S. Nau, and R. Alford, "A hierarchical goal-based formalism and algorithm for single-agent planning," in *Proc. 11th Int. Conf. Auton. Agents and Multiagent Syst. (AAMAS 2012)*, Valencia, Spain, June 4–8, 2012, pp. 981–988.

[72] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nat.*, vol. 529, no. 7587, pp. 484–489, 2016.

[73] Ö. Simsek and A. G. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proc. 21st Int. Conf. Mach. Learn. (ICML 2004)*, Banff, Canada, July 4–8, 2004, p. 95.

[74] Ö. Simsek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proc. 22nd Int. Conf. Mach. Learn.*, Bonn, Germany, August 7–11, 2005, pp. 816–823.

[75] D. Speck, M. Ortlieb, and R. Mattmüller, "Necessary observations in nondeterministic planning," in *Proc. 38th Annual German Conf. Artif. Intell. (KI 2015)*, Dresden, Germany, September 21–25, 2015, pp. 181–193.

[76] S. Sreedharan, C. Muise, and S. Kambhampati, "Generalizing action justification and causal links to policies," in *Proc. 33rd Int. Conf. Automated Planning and Scheduling (ICAPS 2023)*, Prague, Czech Republic, July 8–13, 2023, pp. 417–426.

[77] S. Sreedharan, S. Srivastava, and S. Kambhampati, "TLdR: Policy summarization for factored SSP problems using temporal abstractions," in *Proc. 30th Int. Conf. Automated Planning and Scheduling (ICAPS 2020)*, Nancy, France, October 26–30, 2020, pp. 272–280.

[78] M. Stolle and D. Precup, "Learning options in reinforcement learning," in *Proc. Symp. Abstraction, Reformulation and Approximation (SARA 2002)*, Kananaskis, Canada, August 2–4, 2002, 2002, pp. 212–223.

[79] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1–2, pp. 181–211, 1999.

[80] F. Teichteil-Königsbuch, "Stochastic safest and shortest path problems," in *Proc. 26th AAAI Conf. Artif. Intell. (AAAI 2012)*, Toronto, Canada, July 22–26, 2012, pp. 1825–1831.

[81] S. Vernhes, G. Infantes, and V. Vidal, "Problem splitting using heuristic search in landmark orderings," in *Proc. 23rd Int. Joint Conf. Artif. Intell. (IJCAI 2013)*, Beijing, China, August 3–9, 2013, pp. 2401–2407.

[82] J. Wichlacz, D. Höller, and J. Hoffmann, "Landmark heuristics for lifted classical planning," in *Proc. 31st Int. Joint Conf. Artif. Intell. (IJCAI 2022)*, Vienna, Austria, July 23–29, 2022, pp. 4665–4671.

[83] J. Wichlacz, D. Höller, Á. Torralba, and J. Hoffmann, "Applying Monte-Carlo tree search in HTN planning," in *Proc. 13th Annual Symp. on Combinatorial Search (SoCS 2020)*, May 26–28, 2020, pp. 82–90.

[84] S. W. Yoon, A. Fern, and R. Givan, "FF-Replan: A baseline for probabilistic planning," in *Proc. 17th Int. Conf. Automated Planning and Scheduling (ICAPS 2007)*, Providence, RI, USA, September 22–26, 2007, p. 352.

[85] H. L. S. Younes, M. L. Littman, D. Weissman, and J. Asmuth, "The first probabilistic track of the International Planning Competition," *J. Artif. Intell. Res.*, vol. 24, pp. 851–887, 2005.

[86] M. Yousefi and P. Bercher, "Laying the foundations for solving FOND HTN problems: Grounding, search, heuristics (and benchmark problems)," in *Proc. 33rd Int. Joint Conf. Artif. Intell. (IJCAI 2024)*, Jeju, South Korea, August 3–9, 2024, pp. 6796–6804.

[87] M. Yousefi, J. Schmalz, P. Haslum, and P. Bercher, "Probabilistic HTN Planning: Formalization and Computational Complexity Analysis," in *Proc. 22nd Int. Conf. Princ. Knowl. Representation and Reasoning (KR 2025)*, Melbourne, Australia, November 11–17, 2025, pp. 869–879.

[88] L. Zhu and R. Givan, "Landmark extraction via planning graph propagation," in *ICAPS 2003 Doctoral Consortium*, Trento, Italy, June 9–13, 2003, pp. 156–160.