To appear in: ECAI-2025

Landmark-Assisted Monte Carlo Planning

David H. Chan[a,c,*], **Mark Roberts**[c] **and Dana S. Nau**[a,b]

[a]Department of Computer Science, University of Maryland, College Park, MD, USA
[b]Institute for Systems Research, University of Maryland, College Park, MD, USA
[c]Navy Center for Applied Research in Artificial Intelligence, U.S. Naval Research Laboratory, Washington, DC, USA

Abstract. Landmarks—conditions that must be satisfied at some point in every solution plan—have contributed to major advancements in classical planning, but they have seldom been used in stochastic domains. We formalize probabilistic landmarks and adapt the UCT algorithm to leverage them as subgoals to decompose MDPs; core to the adaptation is balancing between greedy landmark achievement and final goal achievement. Our results in benchmark domains show that well-chosen landmarks can significantly improve the performance of UCT in online probabilistic planning, while the best balance of greedy versus long-term goal achievement is problem-dependent. The results suggest that landmarks can provide helpful guidance for anytime algorithms solving MDPs.

1 Introduction

Landmarks are conditions that must be true in any solution to a planning problem. In classical planning, landmarks have been used to focus search [12] and for heuristic guidance [21]. Ongoing work has developed landmarks for more sophisticated heuristics (see §6).

In stochastic planning, however, landmarks have only been used in limited settings. Speck et al. [28] used a kind of landmark analysis to identify necessary observations. There is also some notable work on using landmarks, or critical states, for plan explanation: Sreedharan et al. [29] defined policy landmarks in MDPs to generate user-explainable policies. To our knowledge, there is scant work exploring the direct application of landmarks to MDP algorithms.

To address this gap, we develop an algorithm called LAMP—**L**andmark-**A**ssisted **M**onte Carlo **P**lanning—that uses landmarks as subgoals to solve stochastic planning problems, much like their original use in classical planning. During rollouts, exploiting landmarks is often more helpful with fewer rollouts and less helpful for a larger rollout budget. So, we develop a mechanism that balances a focus on achieving near-term landmarks with a focus on the long-term goal in a manner reminiscent of Weighted A* within LAMA [21].

The contributions in this paper include:

- formalizing probabilistic landmarks as a natural extension of classical landmarks, first defined by Porteous et al. [19];
- adapting UCT to use landmarks as subgoals during rollouts and developing a weighting parameter that balances between greedily focusing on the next landmark to achieve and the final goal;
- describing the LAMP algorithm, which uses landmark-assisted UCT rollouts to learn a landmark-sensitive Q-function; and,
- demonstrating significant performance improvements by LAMP over standard UCT in five of six probabilistic planning benchmark domains, with the optimal weighting varying across problems.

The results from LAMP highlight the effectiveness of landmarks for probabilistic planning. Similar to the finding that landmarks often resulted in better anytime search for LAMA [23], our results show that using landmarks early in search can yield better solutions.

* Corresponding Author. Email: dhchan@cs.umd.edu

2 Background: classical landmarks

Following the definitions of Ghallab et al. [10], a classical planning domain Σ_C is a tuple (S, A_C, γ_C), where S 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 $\text{Applicable}(s)$ 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 $P_C = (\Sigma_C, s_0, g)$ be a classical planning problem, where $s_0 \in S$ is the initial state and g is the goal condition. A solution to 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 [21], let $P_C = (\Sigma_C, s_0, g)$ 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 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 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** $\varphi_1 \to_{\text{n}} \varphi_2$ 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** $\varphi_1 \to_{\text{gn}} \varphi_2$ 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 [21], but such orderings are not mandatory. They are not used in LAMP, so we omit them here.

3 Probabilistic landmarks

A probabilistic planning domain Σ is a tuple $(S, A, \gamma, \Pr)$, where S and A are finite sets of states and actions, respectively; $\gamma : S \times A \to$

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 $\text{Applicable}(s)$ to denote the set of all applicable actions. A **policy** is a partial function $\pi : S \to A$ with domain $\text{Dom}(\pi) \subseteq S$ such that for all $s \in \text{Dom}(\pi)$, $\pi(s) \in \text{Applicable}(s)$. The policy π is total if $\text{Dom}(\pi) = S$. We will assume all actions have unit cost.

Let $P = (\Sigma, s_0, g)$ be a probabilistic planning problem, where s_0 is the initial state and g is the goal condition. Solutions to 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, $((s_i \models g) \vee (s_i \notin \text{Dom}(\pi))) \iff i = h$. 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 P if $\sum_{\sigma \in H(s_0, \pi, g)} \Pr(\sigma \mid \pi, s_0) = 1$.

We extend the definition of landmarks and landmark orderings to probabilistic planning domains. Let $P = (\Sigma, s_0, g)$ 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 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 P, there is a:

- **natural ordering** $\varphi_1 \to \varphi_2$ 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** $\varphi_1 \to_{\text{n}} \varphi_2$ 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** $\varphi_1 \to_{\text{gn}} \varphi_2$ 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 $P = (\Sigma, s_0, g)$ using the **all-outcomes determinization** of $\Sigma = (S, A, \gamma, \Pr)$, which is a classical domain, denoted $\Sigma_D = (S, A_D, \gamma_D)$, where each action in Σ is replaced by a set of deterministic actions, one for each possible outcome of the original action. More formally, let $s \in S$ and $a \in \text{Applicable}(s)$, and suppose $\gamma(s, a) = \{o_1, \ldots, o_k\}$. 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 Σ_D carry over to Σ as follows:

Lemma 1. *Let $P = (\Sigma, s_0, g)$ be a probabilistic planning problem, where $\Sigma = (S, A, \gamma, \Pr)$. Let $P_D = (\Sigma_D, s_0, g)$ be the classical planning problem where $\Sigma_D = (S, A_D, \gamma_D)$ is the all-outcomes determinization of Σ. Let π be a policy for P. For every history $\sigma = \langle s_1, \ldots, s_h \rangle \in H(s_0, \pi, g)$, there exists a plan $\pi_D = \langle a_1, \ldots, a_h \rangle$ for 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 $P = (\Sigma, s_0, g)$ be a probabilistic planning problem, where $\Sigma = (S, A, \gamma, \Pr)$. Let $P_D = (\Sigma_D, s_0, g)$ be the classical planning problem where Σ_D is the all-outcomes determinization of Σ. If φ is a landmark for P_D, then φ is a landmark for P.*

Proof. Let π be a policy for 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 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 $P = (\Sigma, s_0, g)$ be a probabilistic planning problem, where $\Sigma = (S, A, \gamma, \Pr)$. Let $P_D = (\Sigma_D, s_0, g)$ be the classical planning problem where $\Sigma_D = (S_D, A_D, \gamma_D)$ is the all-outcomes determinization of Σ. Let φ_1 and φ_2 be conditions.*

(1) If $\varphi_1 \to \varphi_2$ in P_D, then $\varphi_1 \to \varphi_2$ in P.
(2) If $\varphi_1 \to_{\text{n}} \varphi_2$ in P_D, then $\varphi_1 \to_{\text{n}} \varphi_2$ in P.
(3) If $\varphi_1 \to_{\text{gn}} \varphi_2$ in P_D, then $\varphi_1 \to_{\text{gn}} \varphi_2$ in 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 P_D, with $\varphi_1 \to \varphi_2$ (resp. $\varphi_1 \to_{\text{n}} \varphi_2$; $\varphi_1 \to_{\text{gn}} \varphi_2$). Let π be a policy for 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 $P = (\Sigma, s_0, g)$, we can use existing classical landmark extraction algorithms, like LM^{RHW} [22], to generate classical landmarks for the all-outcomes determinized planning problem $P_D = (\Sigma_D, s_0, g)$, and directly use those landmarks and their orderings in P.

3.1 Landmarks as subgoals

Before we introduce LAMP (§4), 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. [12]. 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 $P_C = (\Sigma_C, s_0, g)$ 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 $(\Sigma_C, s_{i-1}, \varphi_i)$, 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 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 $P = (\Sigma, s_0, g)$ be a probabilistic planning problem and let $\langle \varphi_1, \ldots, \varphi_k \rangle$ be a sequence of landmarks 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 P by running policies $\pi_1, \ldots, \pi_k$ in sequence (see Appendix A for more details [4]). This procedure again depends on the choice of sequence $\langle \varphi_1, \ldots, \varphi_k \rangle$ and policies π_i.

In the ideal 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 $\mathcal{O}(b^d)$ 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 $\mathcal{O}(b^{d/k})$ to achieve each landmark. With k landmarks, this reduces the total planning time to $\mathcal{O}(kb^{d/k})$, an exponential speed-up (see Appendix A Figure A1 [4]).

However, the existence of such 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 from which the final goal is unreachable [12] (see Appendix A Figure A3 for an example [4]).

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. [12] 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 (§4.2).

4 Landmark-assisted Monte Carlo planning

Intuitively, using landmarks during MDP planning is straightforward. Algorithm 1 shows a simple procedure, called LANDMARKPLAN, that first computes a collection of landmarks Φ (Line 2) and then iteratively selects a landmark (Lines 5–7) and actions to achieve that landmark (Lines 8–10). The behavior of LANDMARKPLAN depends largely on how selection on Lines 7 and 9 are implemented.

Algorithm 2 shows LAMP, the main algorithm of this paper, which is a specific implementation of LANDMARKPLAN based on Monte Carlo tree search. Like LANDMARKPLAN, it has stages for selecting landmarks (Lines 7–9) and actions (Lines 10–12). LAMP learns Q-functions (Lines 5–6) 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 α to achieve a balance between greedy actions to achieve the current landmark with actions that advance toward the top-level goal (Line 11). If a dead-end is encountered, the algorithm fails.

Section 5 will demonstrate that LAMP performs well in several benchmark problems and that the best α 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

Algorithm 1 A general procedure for landmark-guided planning.

```
1:  procedure LANDMARKPLAN(P)
2:      Φ ← LANDMARKGRAPH(P) ∪ {g}
3:      s ← s_0; φ ← NIL
4:      while Φ ≠ ∅ and s ⊭ g do
5:          if φ = NIL or s ⊨ φ then                    ▷ select landmark
6:              Φ ← Φ\{φ}
7:              select φ ∈ leaves(Φ)
8:          else                                        ▷ select action
9:              select a ∈ Applicable(s) to achieve φ
10:             s ← APPLY(s, a)
```

Algorithm 2 The LAMP algorithm, a particular implementation of LANDMARKPLAN. $P = (\Sigma, s_0, g)$ is a planning problem, $n\_rollouts$ is the number of rollouts, $budget$ is a cost limit, $depth$ is the maximum rollout depth, and α is the greediness parameter.

```
1:  procedure LAMP(P, n_rollouts, budget, depth, α)
2:      Φ ← LANDMARKGRAPH(P) ∪ {g}
3:      φ ← NIL; s ← s_0; cost ← 0
4:      while Φ ≠ ∅ and s ⊭ g and cost < budget do
5:          for i ← 1, ..., n_rollouts do     ▷ learn Q_LM, Q_φ, Q_g
6:              ROLLOUT(s, φ, Φ, depth, α, cost)
7:          if φ = NIL or s ⊨ φ then                    ▷ select landmark
8:              Φ ← Φ\{φ}
9:              φ ← argmax_{φ' ∈ leaves(Φ)} (Q_LM(Φ, φ'))
10:         else                                        ▷ select action
11:             a ← argmax_{a ∈ Applicable(s)} (αQ_φ(s, a) + (1 − α)Q_g(s, a))
12:             s ← APPLY(s, a)
13:         cost ← cost + 1
```

based on LANDMARKPLAN which uses UCT [14], 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 LANDMARKPLAN, including non-sampling-based approaches.

Next, we describe the key components of LAMP in determining: how to choose a landmark (§4.1), how to choose an action (§4.2), and how to learn the Q-functions (§4.3).

4.1 Choosing landmarks in LAMP

To learn an ordering of the landmarks, we frame the selection of landmarks as a planning problem. Let $P = (\Sigma, s_0, g)$ be a probabilistic planning problem and let Φ be a collection of landmarks for P. Let Σ_Φ 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 Σ 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 the state in Σ 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 Σ_Φ, 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

$$\text{leaves}(\Phi') = \{\varphi \in \Phi' \mid \varphi \text{ has no } \prec\text{-predecessors}\}. \quad (1)$$

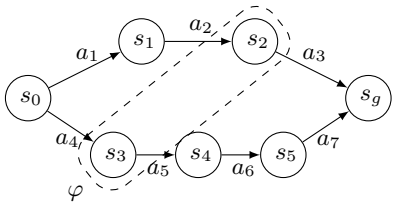


Figure 1. Suppose φ is a landmark such that $s_2 \models \varphi$ and $s_3 \models \varphi$. Starting at s_0, a greedy algorithm may achieve φ by performing action a_4, leading to a worse solution.

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 $P_\Phi = (\Sigma_\Phi, \Phi, g_\Phi)$, where $s \models g_\Phi$ iff $s = \varnothing$.

LAMP's landmark selection bears some similarity to the options framework in hierarchical reinforcement learning [32]. That is, Σ_Φ 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 $\langle \mathcal{I}, \dot{\pi}, \beta \rangle_a$ with initiation set $\mathcal{I} = S$, policy $\dot{\pi} = \mathbf{1}_{[\pi_a(s')=a']}(s', a')$, and termination condition $\beta = \mathbf{1}_{[s' \models a]}(s')$. Since options are known to be semi-Markov by Sutton et al. [32], we know that Σ_Φ is semi-Markov, meaning that landmarks offer a similar benefit to options in providing temporal abstractions to decompose a problem.

4.2 Choosing actions in LAMP

To solve P using the approach outlined in Algorithm 1, we must solve two planning problems concurrently: the landmark selection problem in Σ_Φ (Algorithm 2 Lines 7–9) and the action selection problem in Σ (Algorithm 2 Lines 10–12). 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 11 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 1. Suppose we want to achieve landmark φ with top-level goal g. Assuming actions are deterministic, we can achieve φ by either executing $\langle a_1, a_2 \rangle$ 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 $\alpha \in [0, 1]$, 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

$$\arg\max_{a \in \text{Applicable}(s)} (\alpha Q_\varphi(s, a) + (1 - \alpha) Q_g(s, a)) \tag{2}$$

(cf. Algorithm 2 Line 11). In Figure 1, 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 $\alpha > \frac{1}{5}$ selects action a_4, while $\alpha < \frac{1}{5}$ selects action a_1. Random selection is used to break ties.

4.3 Learning Q-functions for LAMP

To learn these Q-values, LAMP uses an approach based on Monte Carlo tree search (MCTS). MCTS is a general approach to Markov decision processes that has garnered considerable attention after its noteworthy success at computer Go [25]. 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 a tree search with Monte Carlo rollouts and uses the outcome of these rollouts to evaluate states in a search tree.

MCTS iteratively builds a search tree using rollouts from the current state. Within the family of MCTS algorithms, Upper Confidence Bound applied to Trees (UCT) is the most widely used in practice [14]. It uses a solution to the multi-armed bandit problem [1] that provides a principled balance between exploration and exploitation when selecting nodes to expand.

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 Semantic), a criterion for solving SSPs with dead-ends which provides a principled tradeoff between maximizing the probability of reaching the goal and minimizing expected cost with a utility-based model [8, 9]. The utility function U captures a smooth tradeoff between goal achievement and cost, where the utility of a history σ is

$$U(\sigma) = u(|\sigma|) + K_g \mathbf{1}_{[\sigma_{-1} \models g]}, \tag{3}$$

u 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 [6]. 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 as landmark values.

- $Q_g(s, a)$ estimates the expected utility of executing action a in state s for top-level goal g
- $Q_\varphi(s, a)$ estimates the expected utility of executing action a in state s for landmark φ
- $Q_{LM}(\Phi, \varphi)$ estimates the expected utility of first achieving landmark $\varphi \in \Phi$ for top-level goal g.

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). In standard UCT, during each rollout, actions are selected using UCB1 values for each state-action pair, where

$$\text{UCB1}(Q, N, s, a) = Q(s, a) + C\sqrt{\tfrac{\log(N(s))}{N(s,a)}} \tag{4}$$

and C is the exploration constant. In state s, the action that yields the highest UCB1-value is selected. However, in Algorithm 3 Line 9, we adjusted the algorithm to account for the greediness parameter α; actions are selected using a linear combination of UCB1 values corresponding to both the top-level goal and the current landmark (cf. Equation 2).

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,

Algorithm 3 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, α 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 ROLLOUT$(s, \varphi, \Phi, d, \alpha, c)$
  if $\Phi = \varnothing$ **return** $(0, \mathsf{true}, 0, \mathrm{true})$
  if $\varphi =$ NIL or $s \models \varphi$ **then**
    $\varphi' \leftarrow \underset{\varphi' \in \mathrm{leaves}(\Phi)\setminus\{\varphi\}}{\arg\max}$ UCB1$(Q_{LM}, N_{LM}, \Phi, \varphi')$
    $(\lambda_\varphi, \beta_\varphi, \lambda_g, \beta_g) \leftarrow$ ROLLOUT$(s, \varphi', \Phi\setminus\{\varphi\}, d, \alpha, c)$
    UCB-UPDATE$(Q_{LM}, N_{LM}, \Phi, \varphi, \lambda_g + c, \beta_g)$
    return $(0, \mathsf{true}, \lambda_g, \beta_g)$
  if $d = 0$ or Applicable$(s) = \varnothing$ **return** $(d, \mathsf{false}, d, \mathrm{false})$
  $a \leftarrow \underset{a \in \mathrm{Applicable}(s)}{\arg\max}$ $(\alpha$UCB1$(Q_\varphi, N_\varphi, s, a)+$ $(1-\alpha)$UCB1$(Q_g, N_g, s, a))$
  $s' \leftarrow$ SIMULATE(s, a)
  $(\lambda_\varphi, \beta_\varphi, \lambda_g, \beta_g) \leftarrow$ ROLLOUT$(s', \varphi, \Phi, d-1, \alpha, c+1)$
  $\lambda_\varphi \leftarrow \lambda_\varphi + 1$; $\lambda_g \leftarrow \lambda_g + 1$
  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$(Q, N, s, a, \lambda, \beta)$
  $Q(s,a) \leftarrow \frac{N(s,a)Q(s,a)+u(\lambda)+K_g \mathbf{1}_{[\beta]}}{1+N(s,a)}$
  $N(s) \leftarrow N(s)+1$; $N(s,a) \leftarrow N(s,a)+1$

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 $\varphi' \in \mathrm{leaves}(\Phi)$ 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 [13], 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 $\mathcal{O}(|\Phi|)$ 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.

5 Experiments using LAMP

We evaluated LAMP (Algorithm 2) 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 $u : cost \mapsto \exp(-\frac{1}{10} cost)$ was used, mirroring the default parameter settings of [6].

We also used Fast-Downward's implementation of the $\mathrm{LM}^{\mathrm{RHW}}$ landmark extraction algorithm [22] 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 $n\_rollouts \in \{5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$ and the greediness parameter $\alpha \in \{0, 0.2, 0.5, 0.8, 1\}$. When $\alpha = 0$, LAMP is equivalent to standard UCT and serves as the baseline. The code for our implementation is available [5].

5.1 Early benchmarks

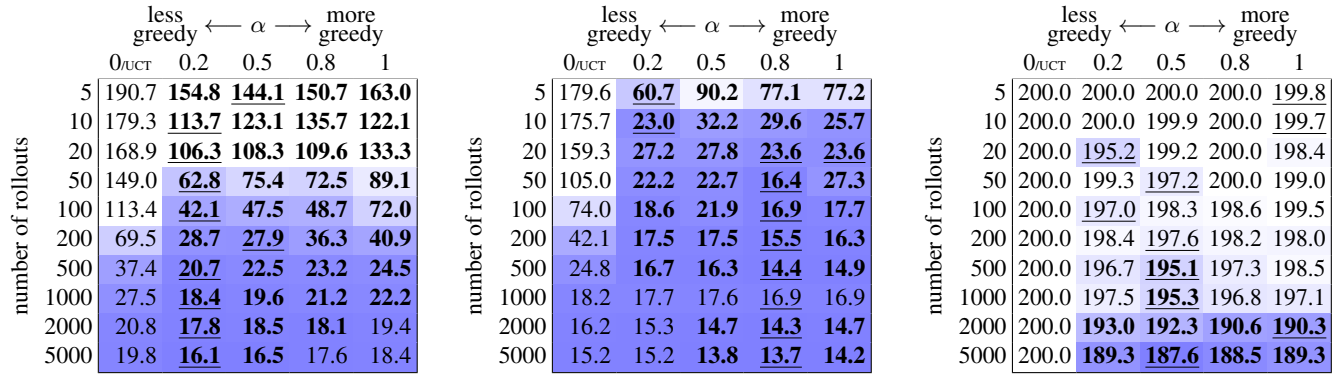
Our first set of experiments use domains and problem instances from the probabilistic track of the Fifth International Planning Competition, which are publicly available [2].

- `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 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.

In Figure 2, we selected one problem instance from each domain. For each problem, we reported the average cost of solutions generated by LAMP, averaged over 75 runs. Runs that exceeded the execution budget of 200 were halted and contributed 200 to the average. Additional results from another 26 problem instances in these domains were qualitatively similar, and are presented in Appendix B [4].

These results show that a greedy approach can significantly improve performance when the planner is constrained to fewer rollouts. Even though standard UCT ($\alpha = 0$) provably converges to an optimal solution [14], 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, standard UCT was never able to reach a goal within the cost budget while the greedy approach could, even with 5000 rollouts. However, we also observe that the optimal greediness value α depends not only on the problem, but also the number of rollouts performed. A greediness value of $\alpha \approx 0.2$ yielded the lowest average cost for the `prob_blocksworld` problem, except in the cases of 5 and 200 rollouts where $\alpha = 0.5$ performed best. In contrast, $\alpha = 0.8$ 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 t-test between LAMP and baseline UCT ($\alpha = 0$) at the same number of rollouts for each value of α. 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 B [4]). In the complete set of results, LAMP



(a) `prob_blocksworld` problem `p5` **(b)** `elevators` problem `p5` **(c)** `zenotravel` problem `p5`

Figure 2. 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 α-value in the row. Boldfaced values indicate where LAMP significantly ($p < 0.0125$) dominated standard UCT ($\alpha = 0$) at the same number of rollouts. Statistical analysis and results from other problem instances are in Appendix B [4].

significantly outperformed standard UCT in 25 of 29 early benchmark problems, and in 172 of 290 total rows.

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 [15]. There 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` [2]: a dead-end variation of standard blocksworld where blocks can detonate upon being put down.
- `tireworld` [2]: 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` [15]: a variation of `tireworld` with a 50% flat-tire rate and a triangular road network (Figure 4).

In Figure 3, we selected one problem instance from each 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. Additional results from another 26 problem instances in these domains are presented in Appendix C [4].

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 4, 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 not 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 (similar to a Fisher exact test) 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 `traingle_tireworld` problem. Full statistical analysis and results from other problems are in Appendix C [4]. 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.

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 ($\alpha > 0$) often outperforms standard UCT ($\alpha = 0$). 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.

6 Related work

Porteous et al. [19] first introduced landmarks for propositional planning. Although deciding whether a propositional formula is a landmark is PSPACE-complete, they proposed an algorithm, LM^{RPG}, that could efficiently extract landmarks and orderings using a delete-relaxed planning graph (RPG). Using the resulting landmark graph, their proposed method restricted the action space by disallowing actions that would achieve landmarks out of order, achieving limited

number of rollouts	less greedy ⟵ α ⟶ more greedy				
	0/UCT	0.2	0.5	0.8	1
5	0.01	0.11	0.05	0.09	0.08
10	0.04	0.17	**0.27**	0.09	0.12
20	0.01	**0.32**	**0.24**	0.12	**0.16**
50	0.03	**0.29**	**0.21**	0.13	0.11
100	0.08	**0.27**	**0.28**	0.21	0.12
200	0.08	**0.47**	**0.32**	**0.37**	0.08
500	0.32	0.37	0.48	0.36	0.09
1000	0.31	0.43	**0.59**	0.47	0.08
2000	0.41	0.49	0.57	0.59	0.03
5000	0.48	0.53	0.60	0.51	0.16

(a) `exploding_blocksworld` problem `p4`

number of rollouts	less greedy ⟵ α ⟶ 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 ⟵ α ⟶ 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. 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 α-value in the row. Boldfaced values indicate where LAMP significantly ($p < 0.0125$) dominated standard UCT ($\alpha = 0$) at the same number of rollouts. Statistical analysis and results from other problem instances are in Appendix C [4].

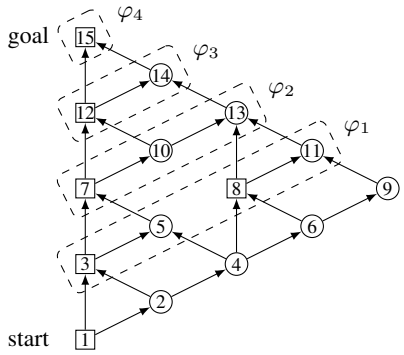


Figure 4. A depiction of `triangle_tireworld` problem `p2` of moving a car from node 1 to node 15. Each time the car moves between nodes, there is a 50% chance it gets a flat tire. There are spare tires at circular nodes. All roads are "one-way." The only safe solution is to move to node 9, then to node 15. The landmarks generated by LM^{RHW} are shown in the dashed boxes; e.g. $\varphi_2 \equiv (\text{car-at}(7) \vee \text{car-at}(10) \vee \text{car-at}(13))$.

results. Hoffmann et al. [12] 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, demonstrating a more substantial speed-up. Vernhes et al. [33] later revisited this problem-splitting approach with a search framework based on landmark orderings for a given landmark graph.

There have also been significant developments in using landmarks to generate heuristic values to guide planning [35, 22, 11, 18, 3]. Others have extended these approaches beyond classical planning to numeric [24] and lifted planning [34].

Landmarks have not been as widely used in probabilistic planning problems. Speck et al. [28] used landmarks 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. [29] defined policy landmarks for MDPs to identify subgoals for a given policy. These policy landmarks were then used to provide high-level explanations of policies and required actions [30].

Within reinforcement learning, the options framework [32] 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 [16, 31, 17, 26, 27], or prototypical states of well-connected regions of the state space [20]. 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 the value of visiting each landmark in the task [7].

7 Conclusion and future work

We extended classical landmarks to probabilistic settings and introduced LAMP, a particular implementation of LANDMARKPLAN 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 α depends on both the problem domain and the number of rollouts, future work could investigate the potential for an adaptive α-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 LANDMARKPLAN (Algorithm 1) could yield improvements similar to those achieved by our UCT-based approach.

Acknowledgements

Special thanks to Paul Zaidins for many helpful discussions. We also sincerely thank the anonymous reviewers for their helpful comments and suggestions. This research was supported by the U.S. Naval Research Laboratory (NRL).

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.

[2] B. Bonet and R. Givan. 5th international planning competition, 2006. URL https://ipc06.icaps-conference.org/probabilistic/.

[3] C. Büchner, R. Christen, S. Eriksson, and T. Keller. Hitting set heuristics for overlapping landmarks in satisficing planning. In *Proc. SoCS 2024*, pages 198–202, 2024.

[4] D. H. Chan, M. Roberts, and D. S. Nau. Landmark-assisted monte carlo planning, 2025. URL https://arxiv.org/abs/2508.11493. Full version of this paper.

[5] D. H. Chan, M. Roberts, and D. S. Nau. Landmark-Assisted Monte Carlo Planning. https://github.com/dhrchan/lamp-pddlgym, 2025.

[6] G. N. Crispino, V. Freire, and K. V. Delgado. Monte carlo tree search algorithm for SSPs under the GUBS criterion. *CLEI Electron. J.*, 27(3), 2024.

[7] A. Demir, E. Çilden, and F. Polat. Landmark based guidance for reinforcement learning agents under partial observability. *Int. J. Mach. Learn. Cybern.*, 14(4):1543–1563, 2023.

[8] V. Freire and K. V. Delgado. GUBS: a utility-based semantic for goal-directed markov decision processes. In *Proc. AAMAS 2017*, pages 741–749, 2017.

[9] 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. ICAPS 2019*, pages 146–154, 2019.

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

[11] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 2009.

[12] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *J. Artif. Intell. Res.*, 22:215–278, 2004.

[13] T. Keller and P. Eyerich. PROST: probabilistic planning based on UCT. In *Proc. ICAPS 2012*, 2012.

[14] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proc. ECML 2006*, pages 282–293, 2006.

[15] I. Little and S. Thiebaux. Probabilistic planning vs. replanning. In *Proc. ICAPS 2007*, pages 1–10, 2007.

[16] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. ICML 2001*, pages 361–368, 2001.

[17] I. Menache, S. Mannor, and N. Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proc. ECML 2002*, pages 295–306, 2002.

[18] F. Pommerening and M. Helmert. Incremental lm-cut. In *Proc. ICAPS 2013*, page 162–170, 2013.

[19] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proc. ECP 2001*, pages 37–48, 2001.

[20] R. Ramesh, M. Tomar, and B. Ravindran. Successor options: An option discovery framework for reinforcement learning. In *Proc. IJCAI 2019*, pages 3304–3310, 2019.

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

[22] S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proc. AAAI 2008*, pages 975–982, 2008.

[23] S. Richter, J. T. Thayer, and W. Ruml. The joy of forgetting: Faster anytime search via restarting. In *Proc. ICAPS 2010*, pages 137–144, 2010.

[24] E. Scala, P. Haslum, D. Magazzeni, and S. Thiébaux. Landmarks for numeric planning problems. In *Proc. IJCAI 2017*, pages 4384–4390, 2017.

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

[26] Ö. Simsek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proc. ICML 2004*, 2004.

[27] Ö. Simsek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proc. ICML 2005*, pages 816–823, 2005.

[28] D. Speck, M. Ortlieb, and R. Mattmüller. Necessary observations in nondeterministic planning. In *Proc. KI 2015*, volume 9324, pages 181–193, 2015.

[29] S. Sreedharan, S. Srivastava, and S. Kambhampati. Tldr: Policy summarization for factored SSP problems using temporal abstractions. In *Proc. ICAPS 2020*, pages 272–280, 2020.

[30] S. Sreedharan, C. Muise, and S. Kambhampati. Generalizing action justification and causal links to policies. In *Proc. ICAPS 2023*, pages 417–426, 2023.

[31] M. Stolle and D. Precup. Learning options in reinforcement learning. In *Proc. SARA 2002*, pages 212–223, 2002.

[32] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[33] S. Vernhes, G. Infantes, and V. Vidal. Problem splitting using heuristic search in landmark orderings. In *Proc. IJCAI 2013*, pages 2401–2407, 2013.

[34] J. Wichlacz, D. Höller, and J. Hoffmann. Landmark heuristics for lifted classical planning. In *Proc. IJCAI 2022*, pages 4665–4671, 2022.

[35] L. Zhu and R. Givan. Landmark extraction via planning graph propagation. In *ICAPS Doctoral Consortium 2003*, pages 156–160, 2003.