Submodular Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

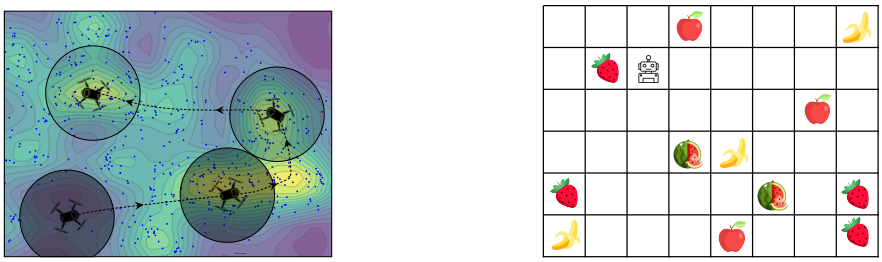
In reinforcement learning (RL), rewards of states are typically considered additive, and following the Markov assumption, they are *independent* of states visited previously. In many important applications, such as coverage control, experiment design and informative path planning, rewards naturally have diminishing returns, i.e., their value decreases in light of similar states visited previously. To tackle this, we propose *submodular RL* (SubRL), a paradigm which seeks to optimize more general, non-additive (and history-dependent) rewards modelled via submodular set functions which capture diminishing returns. Unfortunately, in general, even in tabular settings, we show that the resulting optimization problem is hard to approximate. On the other hand, motivated by the success of greedy algorithms in classical submodular optimization, we propose SubPO, a simple policy gradient-based algorithm for SubRL that handles non-additive rewards by greedily maximizing marginal gains. Indeed, under some assumptions on the underlying Markov Decision Process (MDP), SubPO recovers optimal constant factor approximations of submodular bandits. Moreover, we derive a natural policy gradient approach for locally optimizing SubRL instances even in large state- and action- spaces. We showcase the versatility of our approach by applying SubPO to several applications such as biodiversity monitoring, Bayesian experiment design, informative path planning, and coverage maximization. Our results demonstrate sample efficiency, as well as scalability to high-dimensional state-action spaces.

1 Introduction

In reinforcement learning (RL), the agent aims to learn a policy by interacting with its environment in order to maximize rewards obtained. Typically, in RL, the environments are modelled as a (controllable) Markov chain, and the rewards are considered *additive* and *independent of the trajectory*. In this well-understood setting, referred to as Markov Decision Processes (MDP), the Bellman optimality principle allows to find an optimal policy in polynomial time for finite Markov chains [1, 2]. However, many interesting problems cannot straight-forwardly be modelled via additive rewards. In this paper, we consider a rich and fundamental class of non-additive rewards, in particular *submodular reward functions*. Applications for planning under submodular rewards abound, from coverage control [3], entropy maximization, experiment design [4], informative path planning, orienteering problem [5], resource allocation to Mapping [6].

Submodular functions capture an intuitive diminishing returns property that naturally arises in these applications: *the reward obtained by visiting a state decreases in light of similar states visited previously*. E.g., in a biodiversity monitoring application (see Fig. 1), if the agent has covered a particular region, the information gathered from neighbouring regions becomes redundant and tends to diminish. To tackle such history-dependent, non-Markovian rewards, one could naively augment the state to include all the past states visited so far. This approach, however, exponentially increases the state-space size, leading to intractability. In this paper, we make the following contributions:

Submitted to 16th European Workshop on Reinforcement Learning (EWRL 2023). Do not distribute.



(a) Informative path planning over Gorilla nests (b) Item collection

Figure 1: In Fig. 1a, for monitoring biodiversity, a drone needs to plan a path with maximum coverage over critical areas, represented by lighter regions in a heatmap. Here, the additional information (coverage) provided by visiting a location (state) depends on which states have been visited before. We therefore must visit diverse locations that maximize coverage of important regions. In Fig. 1b, the environment contains a group of items (g_i) placed on a grid. The agent must find a trajectory (τ) that picks a fixed number of items d_i from each group g_i, i.e., $\max_\tau \sum_i \min(|\tau \cap g_i|, d_i)$. If the agent picks more than d_i, it is not rewarded – diminishing gain. Both of these tasks cannot be represented with additive rewards (in terms of locations) and serve as illustrative examples for this work.

Firstly, We introduce SubRL, a paradigm for reinforcement learning with submodular reward functions. To establish limits of the framework, we derive a lower bound that establishes hardness of approximation up to log factors (i.e., ruling out any constant factor approximation) in general (Section 3). **Secondly,** despite the hardness, we show that, in many important cases, SubRL instances can often be effectively solved in practice. In particular, we propose an algorithm, SubPO, motivated by the greedy algorithm in classic submodular optimization. It is a simple policy-gradient-based algorithm for SubRL that handles non-additive rewards by greedily maximizing marginal gains (Section 4). **Thirdly,** we show that in some restricted settings, SubPO performs *provably* well. In particular, under specific assumptions on the underlying MDP, SubRL reduces to an instance of constrained continuous DR-submodular optimization over the policy space. Even though the reduced problem is still NP-hard, we guarantee convergence to the information-theoretically optimal constant factor approximation of $1 - 1/e$, generalizing previous results on submodular bandits (Section 5). **Lastly,** we demonstrate the practical utility of SubPO in simulation as well as real-world applications. Namely, we showcase its use in biodiversity monitoring, Bayesian experiment design, informative path planning, building exploration, car racing and Mujoco robotics tasks. Our algorithm is sample efficient, discovers effective strategies, and scales well to high dimensional spaces (Section 7).

2 Submodular RL: Preliminaries and Problem Statement

Submodularity. Let $\mathcal{V}$ be a ground set. A set function $F : 2^{\mathcal{V}} \to \mathbb{R}$ is *submodular* if $\forall A \subseteq B \subseteq \mathcal{V}$, $v \in \mathcal{V} \backslash B$, we have, $F(A \cup \{v\}) - F(A) \geq F(B \cup \{v\}) - F(B)$. The property captures a notion of diminishing returns, i.e., adding an element v to A will help at least as much as adding it to the superset B. We denote the marginal gain of element v as $\Delta(v|A) \coloneqq F(A \cup \{v\}) - F(A)$. Functions for which $\Delta(v|A)$ is *independent* of A are called *modular*. F is said to be *monotone* if $\forall A \subseteq B \subseteq \mathcal{V}$, we have, $F(A) \leq F(B)$ (or, equiv., $\Delta(v \mid A) \geq 0$ for all v, A).

Controlled Markov Process (CMP). A CMP is a tuple formed by $\langle \mathcal{V}, \mathcal{A}, \mathcal{P}, \rho, H \rangle$, where $\mathcal{V}$ is the ground set, $v \in \mathcal{V}$ is a state, $\mathcal{A}$ is the action space with $a \in \mathcal{A}$. ρ denotes the initial state distribution and $\mathcal{P} = \{P_h\}_{h=0}^{H-1}$, where $P_h(v'|v, a)$ is the distribution of successive state v' after taking action a at state v at horizon h. We consider an episodic setting with finite horizon H.

Submodular MDP. We define a *submodular* MDP, or SMDP, as a CMP with a monotone submodular reward function, i.e., a tuple formed by $\langle \mathcal{V}, \mathcal{A}, \mathcal{P}, \rho, H, F \rangle$. Hereby, using $\mathcal{S} = H \times \mathcal{V}$ to designate time-augmented states (i.e., each $s = (h, v) \in \mathcal{S}$ is a state augmented with the current horizon step), we assume $F : 2^{\mathcal{S}} \to \mathbb{R}$[1] is a monotone submodular reward function. The non-stationary transition distribution $P_h(v'|v, a)$ of the CMP can be equivalently converted to $P((h+1, v')|(h, v), a) = P(s'|s, a)$ since s accounts for time. An episode starts at $s_0 \sim \rho$, and at each time step $h \geq 0$ at state s_h, the agent draws its action a_h according to a policy π (see below). The environment evolves to a new state s_{h+1} following the CMP. A realization of this stochastic process is a trajectory

[1]Without loss of generality, this can be extended to state-action based rewards

$\tau = \big((s_h, a_h)_{h=0}^{H-1}, s_H\big)$, an ordered sequence with a fixed horizon H. $\tau_{l:l'} = \big((s_h, a_h)_{h=l}^{l'-1}, s_{l'}\big)$ denotes the part from time step l to l'. Note that $\tau = \tau_{0:H}$. For each (partial) trajectory $\tau_{l:l'}$, we use the notation $F(\tau_{l:l'})$ to refer to the objective F evaluated on the set of (state,time)-pairs visited by $\tau_{l:l'}$.

Policies. The agent acts in the SMDP according to a *policy*, which in general maps histories $\tau_{0:h}$ to (distributions over) actions a_h. A policy $\pi(a_h|\tau_{0:h})$ is called *Markovian* if its actions only depend on the current state s, i.e., $\pi(a_h|\tau_{0:h}) = \pi(a_h|s_h)$. The set of all Markovian policies is denoted by Π_{M}. Similarly, we can consider Non-Markovian policies $\pi(a_h|\tau_{h-k:h})$ that only depend on the previous past k steps $\tau_{h-k:h}$. The set of all Non-Markovian policies conditioned on history up to past k steps is denoted by Π_{NM}^k, and we use $\Pi_{\text{NM}} := \Pi_{\text{NM}}^H$ to allow arbitrary history-dependence.

Problem Statement. For a given submodular MDP, we want to find a policy to maximize its expected reward. For a given policy π, let $f(\tau;\pi)$ denote the probability distribution over the random trajectory τ following agent's policy π,

$$f(\tau;\pi) = \rho(s_0) \prod_{h=0}^{H-1} \pi(a_h|\tau_{0:h}) P(s_{h+1}|s_h, a_h). \tag{1}$$

The performance measure $J(\pi)$ is defined as the expectation of the submodular set function over the trajectory distribution induced by the policy π and the goal is to find a policy that maximizes the performance measure within a given family of policies Π (e.g., Markovian ones). Precisely,

$$\pi^\star = \arg\max_{\pi\in\Pi} J(\pi), \text{ where } J(\pi) = \sum_\tau f(\tau;\pi) F(\tau). \tag{2}$$

Given the non-additive reward function F, the optimal policy on Markovian and non-Markovian policy classes can differ (since the reward depends on the history, the policy may need to take the history into account for taking optimal actions). In general, even representing arbitrary non-Markovian policies is intractable, as its description size would grow exponentially with the horizon. It is easy to see that for *deterministic* MDPs, the optimal policy is indeed attained by a deterministic Markovian policy:

Proposition 1. *For any deterministic* MDP *with a fixed initial state, the optimal Markovian policy achieves the same value as the optimal non-Markovian policy.*

While this is not true in general, in this paper we focus on optimizing over the family of Markovian policies. The following proposition guarantees that, even for stochastic transitions, there always exists an optimal deterministic policy among the family of Markovian policies Π_M. Therefore, we do not incur a loss compared to stochastic policies.

Proposition 2. *For any set function F, there exists an optimal Markovian policy that is deterministic.*

The proof is provided in Appendix A. The result directly extends to any non-Markovian policy class Π_{NM}^k for $k > 0$ as well, since one can group together the past k states and treat it as high-dimensional Markovian state space.

Examples of submodular rewards. We first observe that classical MDPs are a strict special case of submodular MDPs. Indeed, for some classical reward function $r : \mathcal{V} \to \mathbb{R}$, by setting $F((h_1, v_1), \ldots, (h_k, v_k)) := \sum_{l=1}^k \gamma^{h_l} r(v_l)$, $F(\tau)$ simply recovers the (discounted) sum of rewards of the states visited by trajectory τ (hereby, $\gamma \in [0, 1]$, i.e., use $\gamma = 1$ for the undiscounted setting).

Moreover, a generic way to construct submodular rewards is to take a submodular set function $F' : 2^{\mathcal{V}} \to \mathbb{R}$ defined on the ground set V, and define $F(\tau) := F'(\mathbf{T}\tau)$, using an operator $\mathbf{T} : 2^{H\times\mathcal{V}} \to 2^{\mathcal{V}}$ that drops the time indices. Thus, $F(\tau)$ measures the value of the set of states $\mathbf{T}\tau \subseteq \mathcal{V}$ visited by τ. Note that each state is counted only *once*, i.e., even if F' is a modular function, F exhibits diminishing returns. There are many practically relevant examples of F', such as (weighted) coverage functions, experimental design criteria such as mutual information and others that have found applications in machine learning tasks [cf. 7, 8]. Moreover, many operations preserve submodularity, which can be exploited to build complex submodular objectives from simpler ones ([7, section 1.2]) and can potentially be used in reward shaping. While submodularity is often considered for discrete $\mathcal{V}$, the concept naturally generalizes to continuous domains.

3 Submodular RL: Theoretical Limits

We first show that the SUBRL problem is hard to approximate in general. In particular, we establish a lower bound that implies SUBRL cannot be approximated up to any constant factor in polynomial time,

even for *deterministic* submodular MDPs. We prove this by reducing our problem to a known hard-to-approximate problem – the submodular orienteering problem (SOP) [9]. Since we focus on *deterministic* SMDP's, according to Proposition 1 and Proposition 2, it suffices to consider deterministic, Markovian policies. We now formally state the inapproximability result with proof given in Appendix B.

Theorem 1. *Let* OPT *be the optimal value for* SUBRL. *Even for deterministic* SMDP*'s, the* SUBRL *problem is hard to approximate within a factor of* $\Omega(\log^{1-\gamma} \text{OPT})$ *unless* NP $\subseteq$ ZTIME$(n^{polylog(n)})$.

Thus, under common assumptions in complexity theory [9, 10], the SUBRL problem cannot be approximated in general to better than logarithmic factors. The significance of this result extends beyond submodular RL. As SUBRL falls within the broader category of general non-Markovian reward functions, Theorem 1 implies that problems involving general set functions are similarly inapproximable, limited to logarithmic factors.

Since the inapproximability result we present is worst-case in nature, it does not rule out that interesting SUBRL problems remain practically solvable. In the next section, we introduce a general algorithm that can be efficiently implemented for any SUBRL problem. In Section 7, we demonstrate the effectiveness of the algorithm and its utility in an extensive experimental study involving submodular rewards. Additionally, in Section 5, we establish that our algorithm enjoys a constant factor approximation under stronger assumptions on the Markov chain.

4 General Algorithm: Submodular Policy Optimization SUBPO

We now propose a practical algorithm for SUBRL that can efficiently handle submodular rewards. The core of our approach follows a greedy gradient update on the policy π. As common in the modern RL literature, we make use of approximation techniques for the policies to derive a method applicable to large state-action spaces. This means that the policy $\pi_\theta(a|s)$ is parameterized by $\theta \in \Theta$ where $\Theta \subset \mathbb{R}^l$ is compact. In the case of tabular π, θ specifies an independent distribution over actions for each state.

Approach. The objective from Eq. (2) can be equivalently formulated as $\theta^\star \in \arg\max_{\theta\in\Theta} J(\theta)$ as θ indexes our policy class. Due to the nonlinearity of the parameterization, it is often not feasible to find a global optimum for the above problem. In practice, with appropriate initialization and hyperparameters, variants of gradient descent are known to empirically perform well for MDPs. Precisely,

$$\theta \leftarrow \theta + \underset{\delta\theta:\delta\theta+\theta\in\Theta}{\arg\max}\ \delta\theta^\top \nabla_\theta J(\theta) - \frac{1}{2\alpha}\|\delta\theta\|^2. \tag{3}$$

Various policy gradient methods arise by utilizing different techniques for estimating the gradient and applying regularization [11–13]. The key challenge to all of them is the computation of the gradient $\nabla J(\theta)$. Below, we devise an unbiased gradient estimator for general, non-additive functions.

Gradient Estimator. As common in the policy gradient literature, we can use the score function $g(\tau,\theta) \coloneqq \nabla_\theta(\log \prod_{i=0}^{H-1} \pi_\theta(a_i|s_i))$ to calculate the gradient $\nabla_\theta J$. Namely, Given an MDP and the policy parameters θ,

$$\nabla_\theta J(\pi_\theta) = \sum_\tau f(\tau;\theta) g(\tau,\theta) F(\tau). \tag{4}$$

As Eq. (4) shows, we do not require knowledge of the environment if sampled trajectories are available. It also does not require full observability of the states nor any structural assumption on the MDP. On the other hand, the score gradients suffer from high variance due to sparsity induced by trajectory rewards [2, 14, 15]. Next, we take the SMDP structure into account to develop efficient algorithms.

Marginal gain: We define the marginal gain for a state s in the trajectory $\tau_{0:j}$ up to horizon j as

$$F(s|\tau_{0:j}) = F(\tau_{0:j} \cup \{s\}) - F(\tau_{0:j}).$$

Our approach aims to maximize the marginal gain associated with each action instead of maximizing state rewards. This approach shares similarities with the greedy algorithm commonly used in submodular maximization, which maximizes marginal gains and is known for its effectiveness. Moreover, decomposing the trajectory return into marginal gains and incorporating it in the policy gradient with suitable baselines [16] removes sparsity and thus helps to reduce variance. Inspired by the policy gradient method for additive rewards, we have the following for SMDP:

Theorem 2. *Given an* SMDP *and the policy parameters θ, with any set function F,*

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)} \left[\sum_{i=0}^{H-1} \nabla_\theta \log \pi_\theta(a_i|s_i) \left(\sum_{j=i}^{H-1} F(s_{j+1}|\tau_{0:j}) - b(\tau_{0:j}) \right) \right] \tag{5}$$

We use an importance sampling estimator (log trick) to obtain Eq. (4). To reduce variance, we subtract a baseline $b(\tau_{0:j})$ from the score gradient, which can be a function of the past trajectory $\tau_{0:j}$. This incorporates the causality property in the estimator, ensuring that the action at timestep j cannot affect previously observed states. After simplifying and considering marginals, we obtain Theorem 2. This estimator assigns a higher weight to policies with high marginal gains and a lower weight to policies with low marginal gains. Our experiments show that this method performs well.

We can optimize this approach by using an appropriate baseline as a function of the history $\tau_{0:j}$, which leads to an actor-critic type method. The versatility of the approach is demonstrated by the fact that Theorem 2 holds for any choice of baseline critic. We explain later in the experiments how to choose a baseline. One can perform a Monte Carlo estimate [17] or generalized advantage function (GAE) [18] to estimate returns based on the marginal gain. To encourage exploration, similar to standard PG, we can employ a soft policy update based on entropy penalization, resulting in diverse trajectory samples. Entropy penalization in SUBRL can be thought of as the sum of modular and submodular rewards, which is a submodular function.

Algorithm. The outline of the steps is given in Algorithm 1. We represent the agent by a stochastic policy parameterized by a neural network. The algorithm operates in epochs and assumes a way to generate samples from the environment, e.g., via a simulator. In each epoch, the agent recursively samples actions from its stochastic policy and applies them in the environment leading to a *roll out* of the trajectory where it collects samples (Line 6). We execute multiple (B) batches in each epoch for accurate gradient estimation. To update the policy, we compute the estimator of the policy gradient as per Theorem 2, where we utilize marginal gains of the trajectory instead of immediate rewards as in standard RL (Line 7). Finally, we use stochastic gradient ascent to update the policy parameters.

5 Provable Guarantees in Simplified Settings

In general, Section 3 shows SUBRL is NP-hard to approximate. A natural question is: Is it possible to do better under additional structural assumptions? In this section, we present one interesting case under which SUBPO can be approximated to a constant factor via DR-submodular optimization.

Definition 1 (DR submodularity and DR-property, [19])**.** *For $\mathcal{X} \subseteq \mathbb{R}^d$, a function $f : \mathcal{X} \to \mathbb{R}$ is DR-submodular (has the DR property) if $\forall \mathbf{a} \leq \mathbf{b} \in \mathcal{X}$, $\forall i \in [d]$, $\forall k \in \mathbb{R}_+$ s.t. $(ke_i + \mathbf{a})$ and $(ke_i + \mathbf{b})$ are still in $\mathcal{X}$, it holds, $f(k\mathbf{e}_i + \mathbf{a}) - f(\mathbf{a}) \geq f(k\mathbf{e}_i + \mathbf{b}) - f(\mathbf{b})$. (Notation: $\mathbf{e}_i$ denotes i^{th} basis vector and for any two vectors $\mathbf{a}, \mathbf{b}, \mathbf{a} \leq \mathbf{b}$ means $a_i \leq b_i \forall i \in [d]$)*

Monotone DR-submodular functions form a family of generally non-convex functions, which can be approximately optimized. As discussed below, gradient-based algorithms find constant factor approximations over general convex, downward-closed polytopes.

Under the following condition on the Markov chain, we can show that as long as the policy is parametrized in a particular way, the objective is indeed monotone DR-submodular.

Definition 2 (ϵ-Bandit SMDP)**.** *An* SMDP *s.t. for any $v_j, v_k \in \mathcal{V}$, $j \neq k$, $\forall h \in [H]$ and $\forall v' \in \mathcal{V}$, $P_h(v_j|v', a_j) = 1 - \epsilon_h$, and $P_h(v_k|v', a_j) = \frac{\epsilon_h}{|\mathcal{V}|-1}$ for $\epsilon_h \in \left[0, \frac{|\mathcal{V}|}{|\mathcal{V}|+1}\right]$ is an ϵ-Bandit* SMDP.

This represents a "nearly deterministic" MDP where there is a unique action for each state in the MDP, which takes us to it with $1 - \epsilon$ probability and with the rest, we end up in any other state Fig. 2. While limiting, it generalizes the bandit scenario, which would occur when $\epsilon = 0$. In the following, we consider a class of state-independent policies that can change in each horizon, denoting the horizon dependence with $\pi^h(a)$. We now formally establish the connection between SUBRL and DR-submodularity,

Theorem 3. *For horizon dependent policy π parameterized as $\pi^h(a) \forall h \in [H]$ in an ϵ-Bandit* SMDP*, and $F(\tau)$ is a monotone submodular function, then $J(\pi)$ is monotone DR-submodular.*

The proof is in Appendix D. It builds on two steps; firstly, we use a reparameterization trick to handle policy simplex constraints. We relax the equality constraints on π to lie on a convex polytope $\mathcal{P} =$

Algorithm 1 Submodular Policy Optimization (SUBPO)

Input: SMDP$\mathcal{M}, \pi, N, B$
for epoch $k = 1 : N$ **do**
  for batch $b = 1 : B$ **do**
    for $h = 0 : H - 1$ **do**
      Sample $a_h \sim \pi(a_h|s_h)$, execute a_h
      $D \leftarrow \{s_h, a_h, F(\tau_{0:h+1})\}$
  Estimate $\nabla_\theta J(\pi)$ as per Theorem 2
  Update policy parameters (θ) using Eq. (3)

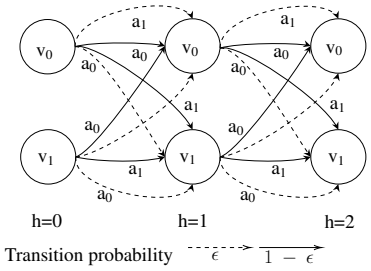


Figure 2: Transitions in ϵ-Bandit MDP

$\{\pi^h(a) \mid 0 \leq \pi^h(a) \leq 1, 0 \leq \sum_{j,j\neq k} \pi^h(a_j) \leq 1, \forall k \in [|\mathcal{A}|], \forall h \in [H]\}$ and enforce the equality constraints directly in the objective Eq. (2). Secondly, under the assumptions of Theorem 3, we show that the Hessian of $J(\pi)$ only has non-positive entries, which is an equivalent characterization of twice differentiable DR-submodular functions. Furthermore, the result can be generalized to accommodate state and action spaces that vary with horizons, although, for simplicity, we assumed fixed spaces.

The convex polytope $\mathcal{P}$ belongs to a class of down-closed convex constraints. Bian et al. [20] proposes a modified FRANK-WOLFE algorithm for DR-submodular maximization with down-closed constraints. This variant can achieve an $(1 - 1/e)$ approximation guarantee and has a sub-linear convergence rate. The algorithm proceeds as follows: the gradient oracle is the same as Theorem 2, while employing a tabular policy parameterization. The polytopic constraints $\mathcal{P}$ are ensured through a FRANK-WOLFE step, which involves solving a linear program over the policy domain. Finally, the policy is updated with a specific step size defined in [20]. Furthermore, Hassani et al. [21] shows that any stationary point in the optimization landscape of DR-submodular maximization under general convex constraints is guaranteed to be $1/2$ optimal. Therefore, any gradient-based optimizer can be used for the $\epsilon-$ Bandit SMDP class, and will result in an $1/2$-optimal policy. In Section 6, we elaborate on how this setting generalizes previous works on submodular bandits.

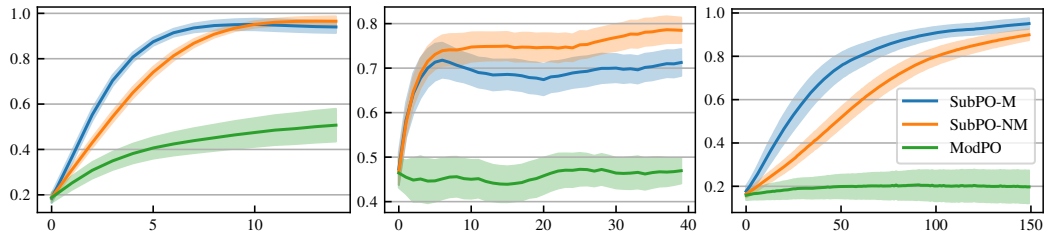
General SMDP. Although we cannot obtain a provable result for the general SMDP setting (Theorem 1), we can show that the SUBPO algorithm, under the assumption of modular rewards, reduces to standard PG, and hence recovers the guarantees and benefits of the modular PG. In particular, if initial state distribution is supported everywhere and under tabular policy parameterization, PG with FRANK-WOLFE provably converges to the global optimum [22].

6 Related Work

Beyond Markovian RL. Several prior works in RL identify the deficiency in the modelling ability of classical Markovian rewards. This manifests itself especially when exploration is desired, e.g., when the transition dynamics are not completely known [23, 24] or when the reward is not completely known [25, 26]. While all these addresses in some aspect the shortcomings of Markovian rewards, they tend to focus on a specific aspect instead of postulating a new class of reward functions as we do in this work.

Convex RL. Convex RL also seeks to optimize a family of non-additive rewards. The goal is to find a policy that optimizes a convex function over the state visitation distribution (which averages over the randomness in the MDP and the policies actions). This framework has applications, e.g., in exploration and experimental design [23, 24, 27–30]. While sharing some motivating applications, convex and submodular RL are rather different in nature. Beyond the high-level distinction that convex and submodular function classes are complimentary, our non-additive (submodular) rewards are defined over the *actual sequence* of states visited by the policy, not its *average behaviour*. Mutti et al. [31] points out that this results in substantial differences, noting the deficiency of convex RL in modelling expected utilities of the form as in Eq. (2), which we address in our work.

Submodular Maximization. Submodular functions are widely studied in combinatorial optimization and operations research and have found many applications in machine learning [7, 8, 32]. The seminal work of Nemhauser et al. [33] shows that greedy algorithms enjoy a constant factor $1 - 1/e$ approximation for maximizing monotone submodular functions under cardinality constraints, which



(a) Coverage over Gorilla nests (b) Item collection (c) Experiment design

Figure 3: Comparison of SubPO-M, SubPO-NM and ModPO. We observe that ModPO get stuck by repeatedly maximizing its modular reward, whereas SubPO-M achieves comparable performance to SubPO-NM while being more sample efficient. (Y-axis: normalized $J(\pi)$, X-axis: epochs)

is information- and complexity- theoretically optimal [34]. Beyond simpler cardinality (and matroid) constraints, more complex constraints have been considered: most relevant is the s-t-submodular orienteering problem [9], which aims to find an s-t-path in a graph of bounded length maximizing a submodular function of the visited nodes, and can be viewed as a special case of SubRL on deterministic SMDPs with deterministic starting state and hard constraint on the goal state. It has been used as an abstraction for informative path planning [35]. We generalize the setup and connect it with modern policy gradient techniques. Certain problems of the form considered in this work can satisfy a notion called *adaptive submodularity*, which generalizes the greedy approximation guarantee over a set of policies [36]. While adaptive submodularity allows capturing history-dependence, it fails to address complex constraints (such as those imposed by CMPs).

While submodularity is typically considered for discrete domains (i.e., for functions defined on $\{0,1\}^{|\mathcal{V}|}$, the concept can be generalized to continuous domains, e.g., $[0,1]^{|\mathcal{V}|}$ using notions such as DR-submodularity [37]. This notion forms a class of non-convex problems admitting provable approximation guarantees in polynomial time, which we exploit in Section 5.

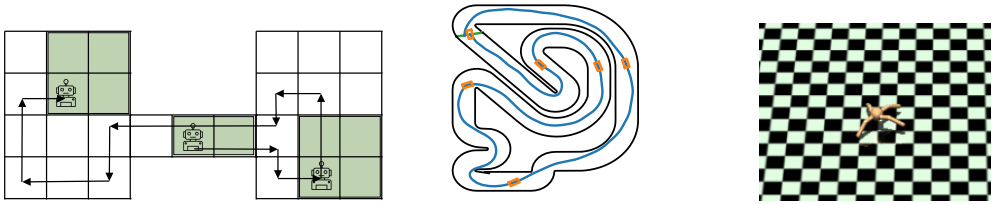
The problem of learning submodular functions has also been considered [38]. Dolhansky and Bilmes [39] introduce the class of deep submodular functions, neural network models guaranteed to yield functions that are submodular in their input. These may be relevant for our setting when learning unknown rewards using function approximation, which is an interesting direction for future work.

The submodular bandit problem is at the interface of learning and optimizing submodular functions [40–42]. Algorithms with no-regret (relative to the 1-1/e approximation) exist, whose performance can be improved by exploiting linearity [42] or smoothness [41] in the objective. Our results in Section 5 can be viewed as addressing (a generalization of) the submodular stochastic bandit problem. Exploiting further linearity or smoothness to improve sample complexity is an interesting direction for future work.

7 Experiments

We empirically study the performance of SubRL on multiple environments. They are i) Informative path planning, ii) Item collection, iii) Bayesian D-experimental design, iv) Building exploration, v) Car racing and vi) Mujoco-Ant. The environments involve discrete (i-iv) and continuous (v-vi) state-action spaces and capture a range of submodular rewards, illustrating the versatility of the framework.

The problem is challenging in two aspects: firstly, how to maximize submodular rewards, and secondly, how to maintain an effective state representation to enable history-dependent policies. Our experiments mainly focus on the first aspect and demonstrate that even with a simple Markovian state representation, by *greedily maximizing marginal gains*, one can achieve good performance similar to the ideal case of non-Markovian representation in many environments. However, we do not claim that a Markovian state representation is sufficient in general. For instance, in the building exploration and item collection environments, Markovian policies are insufficent, and a history-dependent approach is necessary for further optimization. Natural avenues are to augment the state representation to incorporate additional information, e.g., based on domain knowledge, or to use a non-Markovian parametric policy class such as RNNs. Exploring such representations is application specific, and beyond the scope of our work.



(a) Building exploration (b) Car racing (c) Mujoco Ant

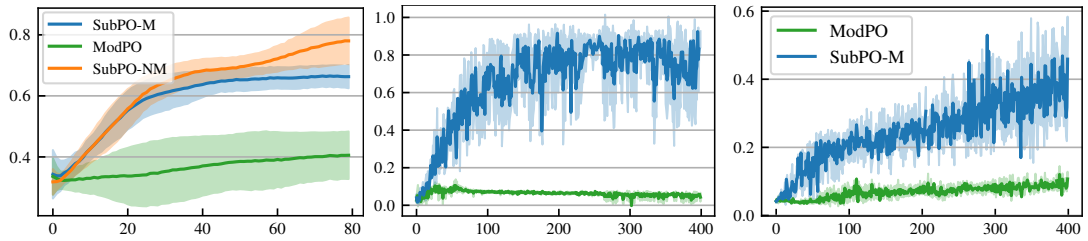
Figure 4: Challenging tasks modelled via submodular reward functions. Primarily, the agent at location s senses a region, D^s and seeks a policy to maximize submodular rewards $F(\tau) = |\cup_{s\in\tau} D^s|$. a) The agent, starting from the middle, must learn to explore both rooms. b) The car must learn to drive & finish the racing lap c) Ant must learn to walk to cover the maximum 2D space around itself.

We consider two variants of SubPO: SubPO-M and SubPO-NM, corresponding to Markovian and non-Markovian policies, respectively. SubPO-NM uses a stochastic policy that conditions an action on the history. We model the policy using a neural network that maps the history to a distribution over actions, whereas SubPO-M maps the state to a distribution over actions. Disregarding sample complexity, we expect SubPO-NM perform the best, since it can track the complete history. In our experiments, we always compare with *modular* RL (ModPO), a baseline that represents standard RL, where the additive reward for any state s is $F(\{s\})$. In ModPO, we maximize the undiscounted sum of additive rewards, whereas, in contrast, SubPO maximizes marginal gains. The rest of the process remains the same, i.e., we use the same policy gradient method. We implemented all algorithms in Pytorch and will make the code and the videos public. We deploy Pytorch's automatic differentiation package to compute an unbiased gradient estimator. Experiment details and extended empirical analysis are in Appendix E. Below we explain our observations for each environment:

Informative path planning. We simulate a bio-diversity monitoring task, where we aim to cover areas with a high density of gorilla nests with a quadrotor in the Kagwene Gorilla Sanctuary (Fig. 1a). The quadrotor at location s covers a limited sensing region around it, D^s. The quadrotor starts in a random initial state and follows deterministic dynamics. It is equipped with five discrete actions representing directions. Let $\rho : V \to \mathbb{R}$ be the nest density obtained by fitting a smooth rate function [43] over Gorilla nest counts [44]. The objective function is given by $F(\tau) = g(\sum_{s\in\tau} D^s)$, where $g(V) = \sum_{v\in V} \rho(v)$. As shown in Fig. 3a, we observe that ModPO repeatedly maximizes its modular reward and gets stuck at a high-density region, whereas SubPO achieves performance as good as SubPO-NM while being more sample efficient. To generalize the analysis, we replace the nest density with randomly generated synthetic multimodal functions and observe a similar trend (Appendix E).

Item collection. Fig. 1b, the environment consists of a grid with a group of items $\mathcal{G} = \{$*banana, apple, strawberries, watermelon*$\}$ located at $g_i \subseteq \mathcal{V}$, $i \in \mathcal{G}$. We consider stochastic dynamics such that with probability 0.9, the action we take is executed, and with probability 0.1, a random action is executed *(up, down, left, right, stay)*. The agent has to find a policy that generates trajectories τ, which picks d_i items from group g_i, for each i. Formally, the submodular reward function can be defined as $F(\tau) = \sum_{i\in\mathcal{G}} \min(|\tau \cap g_i|, d_i)$. We performed the experiment with 10 different randomly generated environments and 20 runs in each. In this environment, the agent must keep track of items collected so far to optimize for future items. Hence as shown in Fig. 3b, SubPO-NM based on non-Markovian policy achieves good performance, and SubPO-M achieves a slightly lower but yet comparable performance just by maximizing marginal gains.

Bayesian D-experimental design. In this experiment, we seek to estimate an a-priori unknown function f. The function f is assumed to be regular enough to be modelled using Gaussian Processes. Where should we sample f to estimate it as well as possible? Formally, our goal is to optimize over trajectories τ that provide maximum mutual information between f and the observations $y_\tau = f_\tau + \epsilon_\tau$ at the points τ. The mutual information is given by $I(y_\tau; f) = H(y_\tau) - H(y_\tau|f)$, representing the reduction in uncertainty of f after knowing y_τ, where $H(y_\tau)$ is entropy. We define the monotonic submodular function $F(\tau) = I(y_\tau; f)$. The gorilla nest density f is an a-priori unknown function. We generate 10 different environments by assuming random initialization and perform 20 runs on each to compute statistical confidence. In Fig. 3c, we observe a similar trend that ModPO gets stuck at a high uncertainty region and cannot effectively optimize the information gained by the entire trajectory, whereas SubPO-M achieves performance as good as SubPO-NM while being very sample efficient due to the smaller search space of the Markovian policy class.



(a) Building Exploration (b) Car Racing (c) Mujoco Ant

Figure 5: a) Coverage in building exploration, SubPO-NM tracks history and can explore the other room b) Car trained with SubPO-M learns to drive through the track (Y-axis: normalized [0-start & 1-finish]) c) Ant trained with SubPO-M learns to explore the domain (Y-axis: normalized with the domain area). Both b,c) show that SubPO scales very well to high dimensional continuous domains.

Building exploration. The environment consists of two rooms connected by a corridor. The agent at s covers a nearby region D^s around itself, marked as a green patch in Fig. 4a. The task is to find a trajectory τ that maximizes the submodular function $F(\tau) = |\cup_{s\in\tau} D^s|$. The agent starts in the corridor's middle and has deterministic dynamics. The horizon H is just enough to cover both rooms. Based on the time-augmented state space, there exists a deterministic Markovian policy that can solve the task. However, it is a challenging environment for exploration with Monte Carlo samples using Markovian policies. As shown in Fig. 5a, SubPO achieves a sub-optimal solution of exploring primarily one side, whereas SubPO-NM tracks the history and learns to explore the other room.

Car Racing is an interesting high-dimensional environment, with continuous state-action space, where a race car tries to finish the racing lap as fast as possible [15]. The environment is accompanied by an important challenge of learning a policy to maneuver the car at the limit of handling. The track is challenging, consisting of 13 turns with different curvature (Fig. 4b). The car has a six-dimensional state space representing position and velocities. The control commands are two-dimensional, representing throttle and steering. Detailed information is in the Appendix E. The car is equipped with a camera and observes a patch around its state s as D^s. The objective function is $F(\tau) = |\cup_{s\in\tau} D^s|$. We set a finite horizon of 700. The SubPO-NM will have a large state space of 700×6, which makes it difficult to train. For variance reduction, we use a baseline $b(s)$ in Eq. (5) that estimates the cumulative sum of marginal gains. As shown in Fig. 5b, under the coverage-based reward formulation, the agent trained with ModPO tries to explore a little bit but gets stuck with a stationary action at the beginning of the episode to get a maximum modular reward. However, the SubPO agent tries to maximise the marginal again at each timestep and hence learns to drive on the race track (https://youtu.be/jXp0QxIQ–E). Although it is possible to use alternative reward functions to train the car using standard RL, the main objective of this study is to demonstrate SubPO on the continuous domains and how submodular functions in RL can provide versatility to achieve surrogate goals.

MuJoCo. The MuJoCo ant task is a high-dimensional locomotion task, as depicted in Fig. 4c. The state space dimension is 30, containing information about the robot's global pose and the internal actuator's orientation. The control input dimension is 8, consisting of torque commands for each actuator. The Ant at any location s covers locations in 2D space, D^s and receives a reward based on it. The goal is to maximize $F(\tau) = |\cup_{s\in\tau} D^s|$. The results depicted in Fig. 5c demonstrate that SubPO maximizes marginal gain and learns to explore the environment, while ModPO learns to stay stationary, maximizing modular rewards. The environment carries the core challenge of continuous control and high-dimensional observation spaces. This experiment shows that SubPO can effectively scale to high-dimensional domains.

8 Conclusion

We introduced submodular reinforcement learning, a new modelling paradigm for sequential decision-making using submodular rewards. We presented an algorithm that exhibits strong empirical performance on this problem. We theoretically analysed the general problem, proving the first inapproximability result for SubRL. Despite the problem's NP-hardness, under simplified assumptions, we achieved a constant-factor approximation for the algorithm. We hope this work will expand the reach of the RL community to embrace the broad class of submodular objectives and the modelling opportunities they offer.

References

[1] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Manish Prajapat, Matteo Turchetta, Melanie Zeilinger, and Andreas Krause. Near-optimal multi-agent learning for safe coverage control. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 14998–15012. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/60dc26558762425a465cb0409fc3dc52-Paper-Conference.pdf`.

[4] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, jun 2008. ISSN 1532-4435.

[5] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2016.04.059. URL `https://www.sciencedirect.com/science/article/pii/S037722171630296X`.

[6] Miquel Kegeleirs, Giorgio Grisetti, and Mauro Birattari. Swarm slam: Challenges and perspectives. *Frontiers in Robotics and AI*, 8:618268, 2021.

[7] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3: 71–104, 2014.

[8] Jeff Bilmes. Submodularity in machine learning and artificial intelligence. *arXiv preprint arXiv:2202.00132*, 2022.

[9] Chandra Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 245–253, 2005. doi: 10.1109/SFCS.2005.9.

[10] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 585–594, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136749. doi: 10.1145/780542.780628. URL `https://doi.org/10.1145/780542.780628`.

[11] Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

[12] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[14] Michael C. Fu. Chapter 19 gradient estimation. In Shane G. Henderson and Barry L. Nelson, editors, *Simulation*, volume 13 of *Handbooks in Operations Research and Management Science*, pages 575–616. Elsevier, 2006. doi: https://doi.org/10.1016/S0927-0507(06)13019-4. URL `https://www.sciencedirect.com/science/article/pii/S0927050706130194`.

[15] Manish Prajapat, Kamyar Azizzadenesheli, Alexander Liniger, Yisong Yue, and Anima Anandkumar. Competitive policy optimization. In *Uncertainty in Artificial Intelligence*, pages 64–74. PMLR, 2021.

[16] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

[17] Leemon C Baird, III. Advantage updating. *WRIGHT LAB WRIGHT-PATTERSON AFB OH*, 1993.

[18] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR*, 2016.

[19] An Bian, Kfir Levy, Andreas Krause, and Joachim M Buhmann. Continuous dr-submodular maximization: Structure and algorithms. *Advances in Neural Information Processing Systems*, 30, 2017.

[20] Andrew An Bian, Baharan Mirzasoleiman, Joachim Buhmann, and Andreas Krause. Guaranteed non-convex optimization: Submodular maximization over continuous domains. In *Artificial Intelligence and Statistics*, pages 111–120. PMLR, 2017.

[21] Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. Gradient methods for submodular maximization. *Advances in Neural Information Processing Systems*, 30, 2017.

[22] Jalaj Bhandari and Daniel Russo. On the linear convergence of policy gradient methods for finite mdps. In *International Conference on Artificial Intelligence and Statistics*, pages 2386–2394. PMLR, 2021.

[23] Jean Tarbouriech and Alessandro Lazaric. Active exploration in markov decision processes. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 974–982. PMLR, 16–18 Apr 2019. URL `https://proceedings.mlr.press/v89/tarbouriech19a.html`.

[24] Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.

[25] David Lindner, Matteo Turchetta, Sebastian Tschiatschek, Kamil Ciosek, and Andreas Krause. Information directed reward learning for reinforcement learning. In *Proc. Neural Information Processing Systems (NeurIPS)*, December 2021.

[26] Stav Belogolovsky, Philip Korsunsky, Shie Mannor, Chen Tessler, and Tom Zahavy. Inverse reinforcement learning in contextual mdps. *Machine Learning*, 2021.

[27] Tom Zahavy, Brendan O'Donoghue, Guillaume Desjardins, and Satinder Singh. Reward is enough for convex mdps. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.

[28] Michael O'Gordon Duff. *Optimal learning: Computational procedures for Bayes -adaptive Markov decision processes*. PhD thesis, University of Massachusetts Amherst, 2002.

[29] Jean Tarbouriech, Shubhanshu Shekhar, Matteo Pirotta, Mohammad Ghavamzadeh, and Alessandro Lazaric. Active model estimation in markov decision processes. In *Conference on Uncertainty in Artificial Intelligence*, pages 1019–1028. PMLR, 2020.

[30] Mojmír Mutný, Tadeusz Janik, and Andreas Krause. Active exploration via experiment design in markov chains. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023. doi: 10.48550/ARXIV.2206.14332. URL `https://arxiv.org/abs/2206.14332`.

[31] Mirco Mutti, Riccardo De Santi, Piersilvio De Bartolomeis, and Marcello Restelli. Challenging common assumptions in convex reinforcement learning. *arXiv preprint arXiv:2202.01511*, 2022.

[32] Ehsan Tohidi, Rouhollah Amiri, Mario Coutino, David Gesbert, Geert Leus, and Amin Karbasi. Submodularity in action: From machine learning to signal processing applications. *IEEE Signal Processing Magazine*, 37(5):120–133, 2020. doi: 10.1109/MSP.2020.3003836.

[33] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[34] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, jul 1998. ISSN 0004-5411. doi: 10.1145/285055.285059. URL https://doi.org/10.1145/285055.285059.

[35] Amarjeet Singh, Andreas Krause, and William J. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, page 1843–1850, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[36] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42: 427–486, 2011.

[37] Andrew An Bian, Joachim M. Buhmann, Andreas Krause, and Sebastian Tschiatschek. Guarantees for greedy maximization of non-submodular functions with applications. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 498–507, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/bian17a.html.

[38] Maria-Florina Balcan and Nicholas JA Harvey. Learning submodular functions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 793–802, 2011.

[39] Brian W Dolhansky and Jeff A Bilmes. Deep submodular functions: Definitions and learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[40] Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. *Advances in Neural Information Processing Systems*, 21, 2008.

[41] Lin Chen, Andreas Krause, and Amin Karbasi. Interactive submodular bandit. In *NIPS*, 2017.

[42] Yisong Yue and Carlos Guestrin. Linear submodular bandits and their application to diversified retrieval. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/33ebd5b07dc7e407752fe773eed20635-Paper.pdf.

[43] Mojmír Mutný and Andreas Krause. Sensing cox processes via posterior sampling and positive bases. *CoRR*, abs/2110.11181, 2021. URL https://arxiv.org/abs/2110.11181.

[44] Neba Funwi-gabga and Jorge Mateu. Understanding the nesting spatial behaviour of gorillas in the kagwene sanctuary, cameroon. *Stochastic Environmental Research and Risk Assessment*, 26, 08 2011. doi: 10.1007/s00477-011-0541-1.

[45] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015.

[46] Egbert Bakker, Lars Nyborg, and Hans B. Pacejka. Tyre modelling for use in vehicle dynamics studies. In *SAE Technical Paper*. SAE International, 02 1987. doi: 10.4271/870421.

Part I

Appendix

A Proof for Propositions

In this section for completion, we briefly prove the following proposition about the MDP with any set function F.

Proposition 1. *For any deterministic* MDP *with a fixed initial state, the optimal Markovian policy achieves the same value as the optimal non-Markovian policy.*

Proof. Let $\mathcal{M}$ be a Markov Decision Process (MDP) with non-Markovian rewards. We can construct an extended MDP, denoted as $\mathcal{M}_e$, where the state space consists of trajectories $\tau_{0:h}$ for all time steps $h \in [H]$. In $\mathcal{M}_e$, the general trajectory rewards F are Markovian. Furthermore, there exists a deterministic optimal Markovian policy for $\mathcal{M}_e$ ([1]) that, when projected back to $\mathcal{M}$, corresponds to a non-Markovian policy.

Without loss of generality, let $\tau = ((s_0, a_0), (s_1, a_1), \ldots, s_H)$ be a trajectory induced by the deterministic optimal non-Markovian policy $\pi^\star_{\text{NM}} \in \Pi_{\text{NM}}$. If we consider non-stationary policies or augment the state space with time information, given the deterministic transitions and a fixed starting state s_0, a deterministic Markovian policy defined as $\pi_{\text{M}}(a_i|s_i) = 1$ and $\pi_{\text{M}}(a_j|s_i) = 0$ for $i \neq j$, $\forall i$, can induce the same trajectory τ and is also optimal (i.e., $J(\pi_{\text{M}}) = J(\pi^\star_{\text{NM}})$), as $\Pi_{\text{M}} \subset \Pi_{\text{NM}}$. □

Proposition 2. *For any set function F, there exists an optimal Markovian policy that is deterministic.*

Proof. Given any stochastic policy, we can pick a state and change the policy of that state to a deterministic action. The value of the new policy will be at least as good as the old policy.

Consider the objective (for simplicity we wrote only the trajectories affected with action at horizon k at state s_k), and wlog let's say there are two actions a and a' at state s_k (the argument stays the same for any finite number of actions).

$$
\begin{aligned}
J(\pi) = &\sum_{\tau \in \Gamma} \mu(s_0) \left[\prod_{i=0}^{H-1} p^i(s_{i+1}|s_i, a_{i+1}) \pi^i(a_{i+1}|s_i) \right] F(\{(s_0, a_1), \ldots (s_k, a_{k+1}) \ldots s_H\}) \\
= &\sum_{\tau \in \Gamma : (s_k, a_{k+1}) \in \tau} \mu(s_0) \left[\prod_{i=0, i \neq k}^{H-1} p^i(s_{i+1}|s_i, a_{i+1}) \pi^i(a_{i+1}|s_i) \right] p^k(s_{k+1}|s_k, a_{k+1}) \underbrace{\pi^k(a_{k+1}|s_k)}_{x} F(\{\tau\}) \\
&+ \sum_{\tau \in \Gamma : (s_k, a'_{k+1}) \in \tau} \mu(s_0) \left[\prod_{i=0, i \neq k}^{H-1} p^i(s_{i+1}|s_i, a_{i+1}) \pi^i(a_{i+1}|s_i) \right] p^k(s_{k+1}|s_k, a'_{k+1}) \underbrace{\pi^k(a'_{k+1}|s_k)}_{y} F(\{\tau\}) \\
&+ constant \qquad \text{(remaining terms, do not vary with } x \text{ and } y\text{)}
\end{aligned}
$$

Since J is linear in variables x, y that have to additional satisfy simplex constraints, an optimal solution lies in a vertex of these points . This corresponds to a deterministic decision for the state s_k. We can repeat this process for any state, starting with any stochastic optimal policy recursively which will result in a deterministic policy that is at least as good as the stochastic policy we started with. □

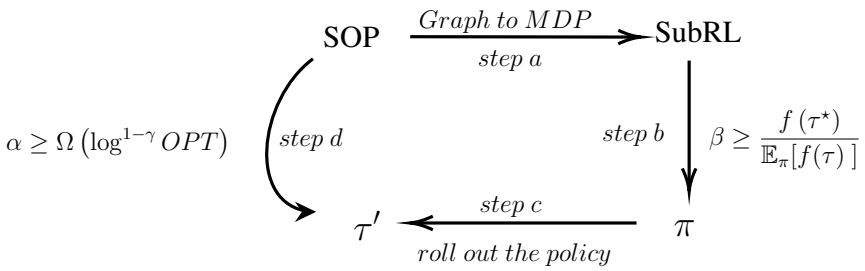


Figure 6: Reduction of SUBRL to submodular orienteering problem (SOP)

B Inapproximability Proof

First, we introduce a set of known hard problems that we will use to deduce the difficulty of SUBRL.

Group Steiner tree (GST). The input to the group Steiner problem is an edge-weighted graph $G = (V, E, l)$ and k subsets of nodes $g_1, g_2, \ldots, g_k$ called *groups*. Starting from a root node r, the goal in GST is to find a minimum weight tree $T^\star = (V(T^\star), E(T^\star))$ in G such that each group is visited at least once, i.e., $V(T^\star) \cap g_i \neq \emptyset, \forall i \in [k]$.

Covering Steiner problem (SCP). The input to SCP is an edge-weighted graph $G = (V, E, l)$ and k subsets of groups $g_1, g_2, \ldots, g_k$, each group has a positive integer d_i representing a minimum visiting requirement. Starting from a root node r, the goal in SCP is to find a minimum weight tree $T^\star = (V(T^\star), E(T^\star))$ in G such that the tree covers at least d_i nodes in group g_i, i.e., $|V(T^\star) \cap g_i| \geq d_i, \forall i \in [k]$. The SCP generalizes GST problem with the arbitrary number of d_i.

Submodular Orienteering Problem (SOP). In rooted SOP we are given a root node $r \in V(T)$, the goal is to find a *walk* τ of length at most B that maximizes some submodular function F defined on the nodes of the underlying graph.

Approximation ratio: Let x be an input instance of a maximization problem. The approximation ratio β is defined as $\beta \geq \frac{\text{OPT}(x)}{\text{ALG}(x)}$, $\beta \geq 1$, where OPT is the global optimal and ALG is the value attained by the algorithm. The hardness lower bound and approximation upper bounds refer to the lower and upper bound on the approximation ratio β.

Theorem 1. *Let* OPT *be the optimal value for* SUBRL. *Even for deterministic* SMDP*'s, the* SUBRL *problem is hard to approximate within a factor of* $\Omega(\log^{1-\gamma} \text{OPT})$ *unless* NP $\subseteq$ ZTIME$(n^{polylog(n)})$.

Proof. Using a reduction scheme similar to [9] (who consider the submodular s-t-orienteering problem), we establish the hardness of rooted SOP (Lemma 1). We reduce SUBRL to rooted SOP, demonstrating the inapproximability of SUBRL.

Given an instance of SOP with a graph $G = (V, E)$ and a root node $r \in V$, the goal is to find a walk τ that maximizes a submodular function $F(\tau)$, subject to a budget constraint $|\tau| \leq B$. This can be converted to a SMDP (input to SUBRL), $\langle \mathcal{S}, \mathcal{A}, P, \rho, F, H \rangle$ tuple in polynomial time as follows:

i) Set $\mathcal{S} \leftarrow H \times V$, $\mathcal{A} \leftarrow E$, $H \leftarrow B$, $\rho(0, r) = 1$ and the submodular function F remains unchanged.
ii) Iterate over the edges $e \in E$. Let $e = (v', v)$, set $P((h+1, v')|(h, v), a) = 1$, for all $h \in [H-1]$ where action $a = e$.

The solution to the SUBRL problem is a policy π that can be rolled out to obtain a solution for the SOP, which is also a polynomial time operation. By assuming the existence of a polynomial time algorithm for SUBRL with an approximation ratio $\beta = o(\log^{1-\gamma} \text{OPT})$, we can approximate SOP with $F(\tau) > \frac{F(\tau^\star)}{\beta}$ (Fig. 6). However, this contradicts the fact that rooted SOP cannot be approximated better than $\Omega(\log^{1-\gamma} \text{OPT})$ (Lemma 1). Proved by contradiction. □

For the sake of completeness, we include Theorem 4.1 from [9] and modify it for the rooted SOP. The reduction scheme is the same as [9]. It is given by,

Lemma 1 (Theorem 4.1 from [9])**.** *The rooted submodular orienteering problem (SOP) in undirected graphs is hard to approximate within a factor of* $\Omega(\log^{1-\gamma} \text{OPT})$ *unless* NP $\subseteq$ ZTIME$(n^{polylog(n)})$.

Proof. The group Steiner problem (GST) is hard to approximate to within a factor of $\Omega(\log^{2-\gamma} \text{OPT})$ unless NP has quasi-polynomial time Las Vegas algorithms [10]. We reduce the problem of rooted SOP to GST, proving the inapproximability of rooted SOP. This represents that if we have an efficient algorithm for SOP, then we can recover a solution for GST by using the same SOP algorithm.

Submodular function F. Given an SCP instance, define a submodular function $F(S) = \sum_{i=1}^{k} \min(d_i, |S \cap g_i|)$. F is a monotone submodular set function.

Consider an optimal solution of SCP as $T^\star$ of cost OPT. We can take an Euler tour of the tree $T^\star$ and obtain a tour from r of length at most 2OPT that covers all groups.

Reduction. We will reduce rooted SOP problem to the SCP (SCP generalises GST with any d_i>0). Let's say we have an algorithm $\mathcal{A}$ for SOP with $\Omega(\log \sum_i d_i)$. ($\sum_i d_i$ is optimal value for SOP). In a single iteration, $\mathcal{A}$ will generate a walk that covers $f(V(T^\star)/\log f(V(T^\star))$ of length B, which can be converted to a tour P of length at most $2B$. We can remove the nodes in P and reduce the coverage requirement of the groups that are partially covered and repeat the above procedure. Using Lemma 2, all groups will be covered up to the requisite amount in $\mathcal{O}(\log^2 \sum_i d_i)$ iterations. Combining all the tours yields a tree of length $\mathcal{O}(\log^2 \sum_i d_i)B$ that is a "feasible solution" of the SCP. B can be evaluated using binary search and is within a constant factor of OPT. When specialized to the GST, i.e., $d_i = 1$, the approximability ratio becomes $\mathcal{O}(\log^2 k)$.

Contradiction. Since $\log k$ approximation algorithm resulted $\mathcal{O}(\log^2 k)$ approximation ratio for GST. (assume $\alpha = o(\log k)$) An α approximation algorithm for SOP will give an approximation of $\mathcal{O}(\alpha \log k)$ for GST. But (GST) is hard to approximate to within a factor of $\Omega(\log^{2-\gamma} \text{OPT})$. Hence SOP is hard to approximate within a factor of $\Omega(\log^{1-\gamma} \text{OPT})$. □

Lemma 2. *A coverage algorithm $\mathcal{A}$ with approximability ratio $\Omega(\log k)$, can cover k nodes after $\mathcal{O}(\log^2 k)$ iterations.*

Proof. L_n be the nodes available after the nth iteration with a algorithm having $\beta \geq \Omega(\log k)$.

$$\begin{aligned} L_0 &\leftarrow k \\ L_1 &\leftarrow L_0 - \frac{L_0}{\log L_0} \\ &\vdots \\ L_{n+1} &\leftarrow \underbrace{L_n - \frac{L_n}{\log L_n}}_{x - \frac{x}{\log x}} \end{aligned}$$

In the first iteration with $x = k$

$$L_1 = x - \frac{x}{\log x} = x(1 - \frac{1}{\log x}) \leq x e^{\frac{-1}{\log x}} = k e^{\frac{-1}{\log k}}.$$

By definition, $L_1 \geq L_2 \geq \ldots L_n$, hence $L_i \leq k e^{-1/\log k} \forall i \in [1, n]$.
Nodes available after n^{th} iteration : $L_0(1 - \frac{1}{\log L_0})(1 - \frac{1}{\log L_1}) \ldots (1 - \frac{1}{\log L_n})$.

$$\begin{aligned} L_0(1 - \frac{1}{\log L_0})(1 - \frac{1}{\log L_1}) \ldots (1 - \frac{1}{\log L_n}) &\leq k \underbrace{e^{-1/\log k} \times e^{-1/\log k} \ldots e^{-1/\log k}}_{n \ \ times} \\ &= k e^{-n/\log k} \end{aligned}$$

for $n > \log^2 k$, Nodes available: $k e^{-n/\log k} < k e^{-\log^2 k/\log k} = k e^{-\log k} = 1$. □

C Submodular policy optimization, SUBPO's policy gradient proof

Theorem 2. *Given an* SMDP *and the policy parameters θ, with any set function F,*

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\sum_{i=0}^{H-1}\nabla_\theta \log \pi_\theta(a_i|s_i)\left(\sum_{j=i}^{H-1} F(s_{j+1}|\tau_{0:j}) - b(\tau_{0:j})\right)\right] \tag{5}$$

Proof. The performance measure is given by,

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi_\theta)}[F(\tau)] = \sum_\tau f(\tau;\pi_\theta)F(\tau)$$

thus, the gradient with respect to θ is given by

$$\nabla_\theta J(\pi_\theta) = \sum_\tau \nabla_\theta f(\tau;\pi_\theta)F(\tau)$$

for any $p_\theta(\tau) \neq 0$ using log trick, $\nabla_\theta \log p_\theta(\tau) = \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}$ from standard calculus and the definition of $f(\tau;\pi_\theta)$ in Eq. 1, we can compute the gradient of the objective. Lets define $g(\tau;\theta) = \nabla_\theta(\log\prod_{i=0}^{H-1}\pi_\theta(a_i|s_i))$ resulting in Eq. (4),

$$\nabla_\theta J(\pi_\theta) = \sum_\tau f(\tau;\pi_\theta)\nabla_\theta(\log\prod_{i=0}^{H-1}\pi_\theta(a_i|s_i))F(\tau) = \sum_\tau f(\tau;\pi_\theta)g(\tau;\theta)F(\tau)$$

For simplicity we abuse notation by $f(\tau;\pi) \coloneqq f(\tau;\pi_\theta)$

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\left(\sum_{i=0}^{H-1}\nabla_\theta \log \pi_\theta(a_i|s_i)\right)F(\tau)\right] \tag{6}$$

Using marginal gain $F(s|\tau_{0:j}) = F(\tau_{0:j}\cup\{s\}) - F(\tau_{0:j})$ and telescopic sum $\sum_{j=0}^{H-1}F(s_{j+1}|\tau_{0:j}) = F(\tau) - F(s_0)$

$$= \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\left(\sum_{i=0}^{H-1}\nabla_\theta \log \pi_\theta(a_i|s_i)\right)\left(\sum_{j=0}^{H-1}F(s_{j+1}|\tau_{0:j}) + F(s_0)\right)\right]$$

$$= \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\sum_{i=0}^{H-1}\nabla_\theta \log \pi_\theta(a_i|s_i)\left(\sum_{j=0}^{H-1}F(s_{j+1}|\tau_{0:j}) + F(s_0)\right)\right] \quad \text{(on simplification)}$$

For any function of partial trajectory up to i, $b'(\tau_{0:i})$, we have, $\sum_{a_i}\pi_\theta(a_i|s_i)\nabla_\theta \log \pi_\theta(a_i|s_i)b'(\tau_{0:i})$ $= \sum_{a_i}\nabla_\theta\pi_\theta(a_i|s_i)b'(\tau_{0:i}) = 0$. Thus one can subtract any history-dependent baseline without altering the gradient estimator,

$$= \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\sum_{i=0}^{H-1}\nabla_\theta \log \pi_\theta(a_i|s_i)\left(\sum_{j=0}^{H-1}F(s_{j+1}|\tau_{0:j}) + F(s_0) - b'(\tau_{0:i})\right)\right]$$

$$= \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\sum_{i=0}^{H-1}\nabla_\theta \log \pi_\theta(a_i|s_i)\left(\sum_{j=i}^{H-1}F(s_{j+1}|\tau_{0:j})\right)\right]$$

Finally, we can subtract a baseline again using a similar trick as above and we get the theorem statement:

$$= \mathop{\mathbb{E}}_{\tau\sim f(\tau;\pi)}\left[\sum_{i=0}^{H-1}\nabla_\pi \log \pi^i(a_i|s_i)\left(\sum_{j=i}^{H-1}F(s_{j+1}|\tau_{0:j}) - b(\tau_{0:i})\right)\right]$$

□

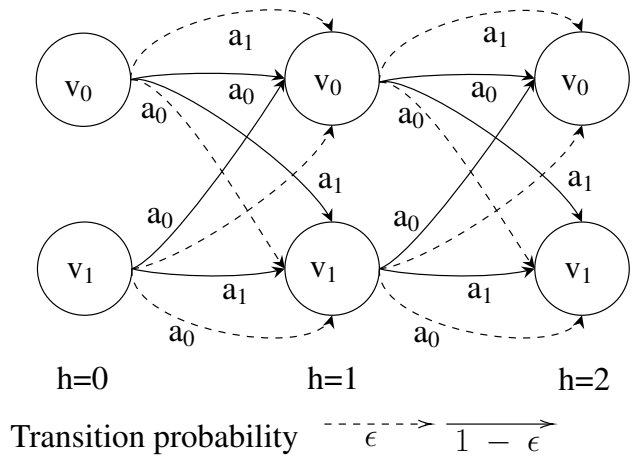


Figure 7: Nomenclature: With action, a_j from any state the agent jumps to state s_j with probability $1-\epsilon$ (solid) and with probability, ϵ it can jump to any other state uniformly (dashed)

D DR-Submodularity Proof

Theorem 3. *For horizon dependent policy π parameterized as $\pi^h(a)\forall h \in [H]$ in an ϵ-Bandit* SMDP*, and $F(\tau)$ is a monotone submodular function, then $J(\pi)$ is monotone DR-submodular.*

Proof. ϵ-Bandit MDP considers *state independent transitions* (only horizon dependent), i.e., $v_j, v_k \in \mathcal{V}$, $j \neq k$, $\forall h \in [H]$ and $\forall v' \in \mathcal{V}$, $P_h(v_j|v', a_j) = 1-\epsilon_h$, and $P_h(v_k|v', a_j) = \frac{\epsilon_h}{|\mathcal{V}|-1}$ for $\epsilon_h \in \left[0, \frac{|\mathcal{V}|}{|\mathcal{V}|+1}\right]$. For simplicity, we consider a fixed size $\mathcal{V}$, but it can also vary with the horizon. To denote explicit dependent on the horizon, we use P_h and v instead of directly s.

Similarly, policy parameterization in Theorem 3 considers *state independent policy* (only horizon dependent) $\pi^h(a|v') = \pi^h(a|v'')\forall h \in [H], \forall v', v'' \in \mathcal{V}$, in short notation we denote them as $\pi^h(a)$.

Note that $\pi^h(a_i|v_i)$ corresponds to the self-loop actions ("stay"). In the proof, we reparameterize the probability of self-loop actions with that of other actions (i.e., $\pi^h(a_i|v_i) = 1 - \sum_{a \neq a_i} \pi^h(a|v_i)$), resulting in relaxation of the simplex constraint $\left(\sum_a \pi^h(a|v) = 1, \forall h, v \rightarrow \sum_{a \neq a_i} \pi^h(a|v_i) \leq 1\right)$.

Consider the objective $J(\pi)$,

$$
\begin{aligned}
J(\pi) &= \sum_{\tau \in \Gamma} \mu(v_0) \prod_{h=0}^{H-1} p_h(v_{h+1}|v_h, a_h)\pi^h(a_h|v_h) F((v_i, a_i)_{i=0}^{H-1}, (H, v_H)) \\
&= \sum_{\tau \in \Gamma} \mu(v_0) \prod_{h=0}^{H-1} p_h(v_{h+1}|a_h)\pi^h(a_h) F((v_i, a_i)_{i=0}^{H-1}, v_H) \quad \text{(state independent assumptions)}
\end{aligned}
$$

We show DR-submodularity by showing $\forall \pi \in \mathcal{P}, \frac{\partial^2 J(\pi)}{\partial \pi' \partial \pi''} \leq 0, \forall \pi', \pi'' \in \mathcal{P}$. Consider $J(\pi)$ with all the self-loop actions (which bring back to the same state) being reparameterized as per $\pi^h(a_h^l|v_h^l) = 1 - \sum_{a \neq a_h^l} \pi^h(a|v_h^l)$. Here a^l is a looping action for state v^l at horizon h.

First, we prove monotonicity of $J(\pi)$, for that we show $\frac{\partial J(\pi)}{\partial \pi^h(a'_h)} \geq 0$. We consider the following for a state v^l but can be added over all the states to get $\frac{\partial J(\pi)}{\partial \pi^h(a'_h)} \geq 0$.

$$
\frac{\partial J(\pi)}{\partial \pi^h(a'_h)} =
$$

$$\sum_{\tau\in\Gamma:(v_h^l,a_h')\in\tau}\mu(v_0)\prod_{i=0}^{H-1}p_i(v_{i+1}|a_i)\left[\prod_{h\neq i}^{H-1}\pi^h(a_h)\right]F((v_0,a_0),\dots\underbrace{(v_h^l,a_h'),(v_{h+1}',a_{h+1}),(v_{h+2},a_{h+2})}_{A}\dots v_H)$$

$$-\sum_{\tau\in\Gamma:(v_h^l,a_h^l)\in\tau}\mu(v_0)\prod_{i=0}^{H-1}p_i(v_{i+1}|a_i)\left[\prod_{h\neq i}^{H-1}\pi^h(a_h)\right]F((v_0,a_0),\dots\underbrace{(v_h^l,a_h^l),(v_{h+1}^l,a_{h+1}),(v_{h+2},a_{h+2})}_{B}\dots v_H)$$

Note: For every trajectory in $E \coloneqq \{\tau\in\Gamma : (v_h^l, a_h')\in\tau\}$ (top summation), we have a trajectory in $L \coloneqq \{\tau\in\Gamma : (v_h^l, a_h)\in\tau\}$ (lower summation) that differ only in v_{h+1}' and v_{h+1}^l and all other states are exactly same. For every trajectory in L, there is a trajectory in E with a higher value. Let's introduce a short notation $f_h^- = \mu(v_0)\prod_{i\neq h}^{H-1}p_i(v_{i+1}|a_i)\pi^i(a_i)$, denotes trajectory distribution ignoring transition at v_h.

$$= \sum_{\tau\in\Gamma:(v_h^l,a_h')\in\tau}{f'}_h^- p_h(v_{h+1}'|a_h')F((v_0,a_0),\dots(v_h,a_h),(v_{h+1},a_{h+1}),(v_{h+2},a_{h+2})\dots v_H)$$

$$-\sum_{\tau\in\Gamma:(v_h^l,a_h^l)\in\tau}{f^l}_h^- p_h(v_{h+1}^l|a_h^l)F((v_0,a_0),\dots(v_h,a_h),(v_h,a_{h+2}),(v_{h+2},a_{h+3})\dots v_H) \tag{7}$$

Note ${f'}_h^-$ and ${f^l}_h^-$ are equal due to state independent transition and policy. Drop the actions (the function F depends on the states) and let $R = \tau\backslash v'$,

$$= \sum_{\tau\in\Gamma:(v_h^l,a_h')\in\tau}{f'}_h^-(1-\epsilon_h-\frac{\epsilon_h}{|\mathcal{V}|-1})\big(F(R\cup v_{h+1}')-F(R)\big)\geq 0 \qquad \text{(since, } \epsilon_h\leq\tfrac{|\mathcal{V}|-1}{|\mathcal{V}|}\text{)}$$

The reason for $(1-\epsilon_h-\frac{\epsilon_h}{|\mathcal{V}|-1}) : (1-\epsilon_h) = p_h(v_{h+1}'|a_h') = p_h(v_{h+1}^l|a_h^l)$ and $\frac{\epsilon_h}{|\mathcal{V}|-1} = p_h(v_{h+1}'|a_h^l) = p_h(v_{h+1}^l|a_h')$. In Eq. (7), the two terms (corresponding to set L and E) are subtracted, with probability $(1-\epsilon_h)$ the first term will be larger since $F(R\cup v_{h+1}')-F(R)\geq 0$ and with probability $\frac{\epsilon_h}{|\mathcal{V}|-1}$ the second term will be larger since the stochastic transition (e.g., looping action a_h^l can jump to next state v_{h+1}' and a_h' stays to the same state v_{h+1}^l. This can happen with probability $\frac{\epsilon_h}{|\mathcal{V}|-1}$). In the remaining stohastic transitions, in expectation, the two terms will sum to zero.

In the above, we have proved the monotonicity of $J(\pi)$ given $F(\cdot)$ is a monotone function.

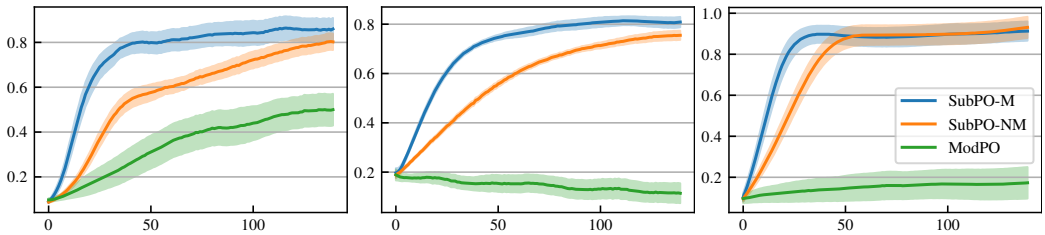
From the part above, we get to have,

$$\frac{\partial J(\pi)}{\partial\pi^h(a_h')} = \sum_\tau\mu(v_0)(1-\frac{|\mathcal{V}|\epsilon_h}{|\mathcal{V}|-1})\prod_{h=0}^{H-1}p_h(v_{h+1}|a_h)\left[\prod_{h\neq i}^{H-1}\pi^h(a_h)\right]\big(F(R\cup v_{h+1}')-F(R)\big)\geq 0$$

To obtain the hessian terms, we can follow the same process as above at some horizon g and state a''. Let $R = \tau\backslash(v', v'')$, ($v''$ is the state corresponding to action a''),

$$\begin{aligned}\frac{\partial^2 J(\pi)}{\partial\pi^g(a_g'')\partial\pi^h(a_h')} &= \sum_{\tau\in\Gamma}{f'}_{g,h}^-(1-\frac{|\mathcal{V}|\epsilon_g}{|\mathcal{V}|-1})(1-\frac{|\mathcal{V}|\epsilon_h}{|\mathcal{V}|-1})\\ &\qquad\Big(F(R\cup(v_{g+1}'',v_{h+1}'))-F(R\cup v_{h+1}')-(F(R\cup v_{g+1}''))-F(R))\Big)\\ &= \sum_{\tau\in\Gamma}{f'}_{g,h}^-(1-\frac{|\mathcal{V}|\epsilon_g}{|\mathcal{V}|-1})(1-\frac{|\mathcal{V}|\epsilon_h}{|\mathcal{V}|-1})\\ &\qquad\Big(F(\underbrace{R\cup v_{g+1}''}_{A}\cup v_{h+1}')-F(\underbrace{R\cup v_{g+1}''}_{A})-(F(R\cup v_{h+1}')-F(R))\Big)\\ &\leq 0 \qquad\qquad \text{(By submodularity of } F,\ R\subseteq A,\ \Delta_F(v'|A)\leq\Delta_F(v'|R))\end{aligned}$$

Hence $J(\pi)$ is DR-submodular. □



(a) Constant weights (b) Bimodal distribution (c) Gaussian Processes

Figure 8: Comparison of SubPO-M, SubPO-NM and ModPO. We observe that ModPO gets stuck by repeatedly maximizing its modular reward, whereas SubPO-M achieves comparable performance to SubPO-NM while being more sample efficient. (Y-axis: normalized $J(\pi)$, X-axis: epochs)

E Experiments

In this work, we examined state-action spaces that are both discrete (finite) and continuous. Here, we provide an overview of the hyperparameters and experiment specifics for each type.

E.1 Finite state action spaces

We consider a grid world environment with domain size 30×30, having a total of 900 states. Horizon $H = 40$, i.e., the agent has to find a path of length 40 that jointly maximize the objective. The action set comprised of $\{right, up, left, down, stay\}$. The agent's policy was parameterized by a two-layer multi-layer perceptron, consisting of 64 neurons in each layer. The non-linearity in the network was induced by employing the Rectified Linear Unit (ReLU) activation function. By employing a stochastic policy, the agent generated a categorical distribution over the action set for each state. Subsequently, this distribution was passed through a softmax probability function. We employed a batch size of $B = 500$ and a low entropy coefficient of $\alpha = 0$ or 0.005, depending on the specific characteristics of the environment.

We randomly generated ten different environments and conducted 20 runs for each environment, resulting in a total of 200 experiments. These experiments were executed concurrently on a server with 400 cores, allocating two cores for each job. It takes less than 20 minutes to complete 150 epochs and obtain a saturated policy, indicating no further improvement. All our plot shows the training curve (objective vs epochs).

For instances where we utilized randomly sampled environments, such as coverage with GP samples, gorilla nest density, or item collection environment, we have included the corresponding environment files in the attached code for easy reference.

To conduct additional analysis using the *Informative Path Planning Environment*, we made modifications to the underlying function previously based on gorilla nest density. Specifically, we introduced three variations: i) a constant function, ii) a bimodal distribution, and iii) a multi-modal distribution randomly sampled from a Gaussian Process (GP). As depicted in Figure 8, we noticed a consistent trend across all three variations. The ModPO algorithm repeatedly maximized its modular reward but became trapped in high-density regions. In contrast, the SubPO-M algorithm demonstrated performance comparable to SubPO-NM while exhibiting greater sample efficiency.

E.2 continuous state-action spaces

Car Racing. In the car racing environment, our objective is to achieve the fastest completion of a one-lap race. To accomplish this, we aim to learn a policy that effectively controls a car operating at its handling limits. Our simulation study closely emulates the experimental platform employed at ETH Zurich, which utilizes miniature autonomous race cars. Building upon Liniger et al. [45], we model the dynamics of each car using a dynamic bicycle model augmented with Pacejka tire models [46]. However, we deviate from the approach presented in [45] by formulating the dynamics in curvilinear coordinates, where the car's position and orientation are represented relative to a reference path. This coordinate transformation significantly simplifies the reward definition and facilitates policy learning. The state representation of an individual car is denoted as $z = [\rho, d, \mu, V_x, V_y, \psi]^T$. Here, ρ measures the progress along the reference path, d quantifies the deviation from the reference path, μ

characterizes the local heading relative to the reference path, V_x and V_y represent the longitudinal and lateral velocities in the car's frame, respectively, and ψ represents the car's yaw rate. The car's inputs are represented as $[D, \delta]^T$, where $D \in [-1, 1]$ represents the duty cycle input to the electric motor, ranging from full braking at -1 to full acceleration at 1, and $\delta \in [-1, 1]$ corresponds to the steering angle.

The car is equipped with a camera and observes a patch around its state s as D^s. The objective function is $F(\tau) = | \cup_{s \in \tau} D^s|$. For simplicity, we define the observation patch of some 5 m and spanning the entire width of the race track. We add additive reward penalization to avoid hitting the walls. If a narrow-width patch is used, then one can eliminate the use of reward penalization (used to avoid hitting the boundaries) as coverage near the boundaries is low and the agent learns to drive in the middle or go to the extreme if they gain due to going fast. The test track, depicted in Figure 4b, consists of 13 turns with varying curvatures. We utilize an optimized X-Y path as the reference path obtained through a lap time optimization tool. It is worth noting that using a pre-optimized reference path is not obligatory, but we observed improved algorithm convergence when employing this approach. To convert the continuous-time dynamics into a discrete-time Markov Decision Process (MDP), we discretize the dynamics using an RK4 integrator with a sampling time of 0.03 seconds.

For the training, we started the players on the start line ($\rho = 0$) and randomly assigned $d \in \{0.1, -0.1\}$. We limit one episode to 700 horizon. For each policy gradient step, we generated a batch of 8 game trajectories and ran the training for roughly 6000 epochs until the player consistently finished the lap. This takes roughly 1 hour of training for a single-core CPU. We use Adam optimizer with a slow learning rate of 10^{-3}. For our experiments, we ran 20 different random runs and reported the mean of all the seeds. As a policy, we use a multi-layer perceptron with two hidden layers, each with 128 neurons and used ReLU activation functions and a Tanh output layer to enforce the input constraints. For variance reduction, we use a baseline $b(s)$ in Eq. (5) that estimates the cumulative sum of marginal gains. One can think of this as a heuristic to estimate marginal gains. The same setting was also used for the competing algorithm MODPO.

MuJoCo.: It is a physics engine for simulating high-dimensional continuous control tasks in robotics. In this experiment, we consider the Ant environment, which is depictive of the core challenges that arise in the Mujoco environment. For detailed information about the Mujoco Ant environment, we refer the reader to https://gymnasium.farama.org/environments/mujoco/ant/.

For the training, we use the default random initialization of Mujoco-Ant. We consider a bounded domain of $[-20, 20]^2$. The agent covers a discrete grid of 5x5 around its location in the 2D space (only for efficient reward computation, we discretize the domain into a 400×400 grid, dynamics is continuous) and receives a reward based on the coverage. To train the agent faster, one can also couple MuJoCo's inbuilt additive rewards (tuned for faster walking) with the submodular coverage rewards, which results in submodular rewards.

We limit one episode to a horizon of 400. We use a vectorized environment that samples a batch of 15 trajectories at once. We trained for roughly 20,000 epochs until the agent consistently walks and explores the domain. This takes roughly 6 hours of training for a single-core CPU. We use Adam optimizer with a learning rate of 10^{-2}. For our experiments, we ran 20 different random runs and reported the mean of all the seeds. As a policy, we use a multi-layer perceptron with two hidden layers, each with 128 neurons and used ReLU activation functions and a Tanh output layer to enforce the input constraints. Similar to the car racing environment, we use a baseline $b(s)$ in Eq. (5) that estimates the cumulative sum of marginal gains. The same setting was also used for the competing algorithm MODPO.