Autonomous Robots (2020) 44:485–503
https://doi.org/10.1007/s10514-019-09871-2

Hierarchical reinforcement learning via dynamic subspace search for multi-agent planning

Aaron Ma[1] · Michael Ouimet[2] · Jorge Cortés[1]

Received: 24 April 2018 / Accepted: 22 June 2019 / Published online: 12 July 2019

Abstract
We consider scenarios where a swarm of unmanned vehicles (UxVs) seek to satisfy a number of diverse, spatially distributed objectives. The UxVs strive to determine an efficient plan to service the objectives while operating in a coordinated fashion. We focus on developing autonomous high-level planning, where low-level controls are leveraged from previous work in distributed motion, target tracking, localization, and communication. We rely on the use of state and action abstractions in a Markov decision processes framework to introduce a hierarchical algorithm, *Dynamic Domain Reduction for Multi-Agent Planning*, that enables multi-agent planning for large multi-objective environments. Our analysis establishes the correctness of our search procedure within specific subsets of the environments, termed 'sub-environment' and characterizes the algorithm performance with respect to the optimal trajectories in single-agent and sequential multi-agent deployment scenarios using tools from submodularity. Simulated results show significant improvement over using a standard *Monte Carlo tree search* in an environment with large state and action spaces.

Keywords Reinforcement learning · Multi-agent planning · Distributed robotics · Semi-Markov decision processes · Markov decision processes · Upper confidence bound tree search · Hierarchical planning · Hierarchical Markov decision processes · Model-based reinforcement learning · Swarm robotics · Dynamic domain reduction · Submodularity

1 Introduction

Recent technology has enabled the deployment of UxVs in a wide range of applications involving intelligence gathering, surveillance and reconnaissance, disaster response, exploration, and surveying for agriculture. In many scenarios, these unmanned vehicles are controlled by one or, more often than not, multiple human operators. Reducing UxV dependence on human effort enhances their capability in scenarios where communication is expensive, low bandwidth, delayed, or contested, as agents can make smart and safe choices on their own. In this paper we design a framework for enabling multi-agent autonomy within a swarm in order to satisfy arbitrary spatially distributed objectives. Planning presents a challenge because the computational complexity of determining optimal sequences of actions becomes expensive as the size of the swarm, environment, and objectives increase.

1.1 Literature review

Recent algorithms for decentralized methods of multi-agent deployment and path planning enable agents to use local information to satisfy some global objective. A variety of decentralized methods can be used for deployment of robotic swarms with the ability to monitor spaces, see e.g., Gerkey and Mataric (2004), Bullo et al. (2009), Mesbahi and Egerstedt (2010), Dunbabin and Marques (2012), Das et al. (2015), Cortés and Egerstedt (2017) and references therein. We gather motivation from these decentralized methods because

A preliminary version of this work appeared as Ma et al. (2017) at the International Symposium on Multi-Robot and Multi-Agent Systems.

This is one of the several papers published in *Autonomous Robots* comprising the Special Issue on Multi-Robot and Multi-Agent Systems.

✉ Aaron Ma
aam021@ucsd.edu

Michael Ouimet
ouimet@spawar.navy.mil

Jorge Cortés
cortes@ucsd.edu

[1] Department of Mechanical and Aerospace Engineering, University of California, San Diego, La Jolla, USA

[2] Naval Information Warfare Center Pacific, San Diego, USA

they enable centralized goals to be realized with decentralized computation and autonomy. In general, these decentralized methods provide approaches for low-level autonomy in multi-agent systems, so we look to common approaches used for high-level planning and scheduling algorithms.

Reinforcement learning is relevant to this goal because it enables generalized planning. Reinforcement learning algorithms commonly use Markov decision processes (MDP) as the standard framework for temporal planning. Variations of MDPs exist, such as semi-Markov decision processes (SMDP) and partially-observable MDPs (POMDP). These frameworks are invaluable for planning under uncertainty, see e.g., Sutton et al. (1999), Broz et al. (2008), Puterman (2014); Howard (1960). Given a (finite or infinite) time horizon, the MDP framework is conducive to constructing a policy for optimally executing actions in an environment (Papadimitriou and Tsitsiklis 1987; Lovejoy 1991; Bellman 1966). Reinforcement learning contains a large ecosystem of approaches. We separate them into three classes with respect to their flexibility of problem application and their ability to plan online vs the need to be trained offline prior to use.

The first class of approaches are capable of running online, but are tailored to solve specific domain of objectives, such as navigation. The work Bai et al. (2016) introduces an algorithm that allows an agent to simultaneously optimize hierarchical levels by learning policies from primitive actions to solve an abstract state space with a chosen abstraction function. Although this algorithm is implemented for navigational purposes, it can be tailored for other objective domains. In contrast, our framework reasons using higher levels of abstraction over different types of multiple objectives. In our formulation, we assume agents have the ability to use pre-existing algorithms, such as Bai et al. (2016), LaValle and Kuffner (2000), Prentice and Roy (2010), Agha-mohammadi et al. (2011), as actions that an agent utilizes in a massive environment. The dec-POMDP framework Oliehoek and Amato (2016) incorporates joint decision making and collaboration of multiple agents under uncertain and high-dimensional environments. *Masked Monte Carlo Search* is a dec-POMDP algorithm Omidshafiei et al. (2015) that determines joint abstracted actions in a centralized way for multiple agents that plan their trajectories in a decentralized POMDP. Belief states are used to contract the expanding history and curse of dimensionality found in POMDPs. Inspired by Rapidly-Exploring Randomized Trees (LaValle and Kuffner 2000), the Belief Roadmap (Prentice and Roy 2010) allows an agent to find minimum cost paths efficiently by finding a trajectory through belief spaces. Similarly, the algorithm in Agha-mohammadi et al. (2011) creates Gaussian belief states and exploits feedback controllers to reduce POMDPs to MDPs for tractability in order to find a trajectory. Most of the algorithms in this class are not necessarily comparable to each other due to the specific context of their problem statements and type of objective. For that reason, we are motivated to find an online method that is still flexible and can be utilized for a large class of objectives.

The second class of approaches have flexible use cases and are most often computed offline. These formulations include reinforcement learning algorithms for value or policy iteration. In general, these algorithms rely on MDPs to examine convergence, although the model is considered hidden or unknown in the algorithm. An example of a state-of-the-art reinforcement model-free learner is Deep Q-network (DQN), which uses deep neural networks and reinforcement learning to approximate the value function of a high-dimensional state space to indirectly determine a policy afterwards (Mnih et al. 2015). Policy optimization reinforcement algorithms focus on directly optimizing a policy of an agent in an environment. Trust Region Policy Optimization (TRPO) (Schulman et al. 2015) enforces constraints on the KL-divergence between the new and old policy after each update to produce more incremental, stable policy improvement. Actor-Critic using Kronecker-factored Trust Region (ACKTR) (Wu et al. 2017) is a hybrid of policy optimization and Q-learning which alternates between policy improvement and policy evaluation to better guide the policy optimization. These techniques were successfully applied to a range of Atari 2600 games, with results similar to advanced human players. Offline, model-free reinforcement algorithms are attractive because they can reason over abstract objectives and problem statements, however, they do not take advantage of inherent problem model structure. Because of this, model-free learning algorithms usually produce good policies more slowly than model-based algorithms, and often require offline computation.

The third class of algorithms are flexible in application and can be used online. Many of these algorithms require a model in the form of a MDP, or other variations. Standard algorithms include Monte Carlo tree searches (MCTS) (Bertsekas 1995) and modifications such as the upper confidence bound tree search (Kocsis and Szepesvári 2006). Many works under this category attempt to address the curse of dimensionality by lowering the state space through either abstracting the state space (Hansen and Feng 2000), the history in POMDPs (McCallum and Ballard 1996), or the action space (Theocharous and Kaelbling 2004). These algorithms most closely fit our problem statement, because we are interested in online techniques for optimizing across a large domain of objectives.

In the analysis of our algorithm, we rely on the notion of submodular set functions and the characterization of the performance of greedy algorithms, see e.g., Clark et al. (2016), Bian et al. (2017). Even though some processes of our algorithm are not completely submodular, we are able to invoke these results by resorting to the concept of submodularity ratio (Das and Kempe 2011), that quantifies how far a set function is from being submodular, using tools from scenario optimization (Campi et al. 2009).

1.2 Statement of contributions

Our approach seeks to bridge the gap between the MDP-based approaches described above. We provide a framework that remains general enough to reason over multiple objective domains, while taking advantage of the inherent spatial structure and known vehicle model of most robotic applications to efficiently plan. Our goal is to synthesize a multi-agent algorithm that enables agents to abstract and plan over large, complex environments taking advantage of the benefits resulting from coordinating their actions. We determine meaningful ways to represent the environment and develop an algorithm that reduces the computational burden on an agent to determine a plan. We introduce methods of generalizing positions of agents, and objectives with respect to proximity. We rely on the concept of 'sub-environment', which is a subset of the environment with respect to proximity-based generalizations, and use high-level actions, with the help of low-level controllers, designed to reduce the action space and plan in the sub-environments. The main contribution of the paper is an algorithm for splitting the work of an agent between dynamically constructing and evaluating sub-environments and learning how to best act in that sub-environment, cf. Figure 1. We also introduce modifications that enable multi-agent deployment by allowing agents to interact with the plans of other team members. We provide convergence guarantees on key components of our algorithm design and identify metrics to evaluate the performance of the sub-environment selection and sequential multi-agent deployment. Using tools from submodularity and scenario optimization, we establish formal guarantees on the suboptimality gap of these procedures. We illustrate the effectiveness of dynamically constructing sub-environments for planning in environments with large state spaces through simulation and compare our proposed algorithm against Monte Carlo tree search techniques.

1.3 Organization

The paper is organized as follows. Section 2 presents preliminaries on Markov decision processes. Section 3 introduces the problem of interest. Section 4 describes our approach to abstract states and actions with respect to spatial proximity, and Sect. 5 builds on these models to design our hierarchical planning algorithm. Section 6 presents analysis on algorithm convergence and performance, and Sect. 7 shows our simulation results. We gather our conclusions and ideas for future work in Sect. 8.

1.4 Notation

We use $\mathbb{Z}$, $\mathbb{Z}_{\geq 1}$, $\mathbb{R}$, and $\mathbb{R}_{>0}$ to denote integer, positive integer, real, and positive real numbers, respectively. We let $|Y|$ denote the cardinality of an arbitrary set Y. We employ object-oriented notation throughout the paper; $b.c$, means that c belongs to b, for arbitrary objects b and c. For reference, "Appendix C" presents a list of commonly used symbols.

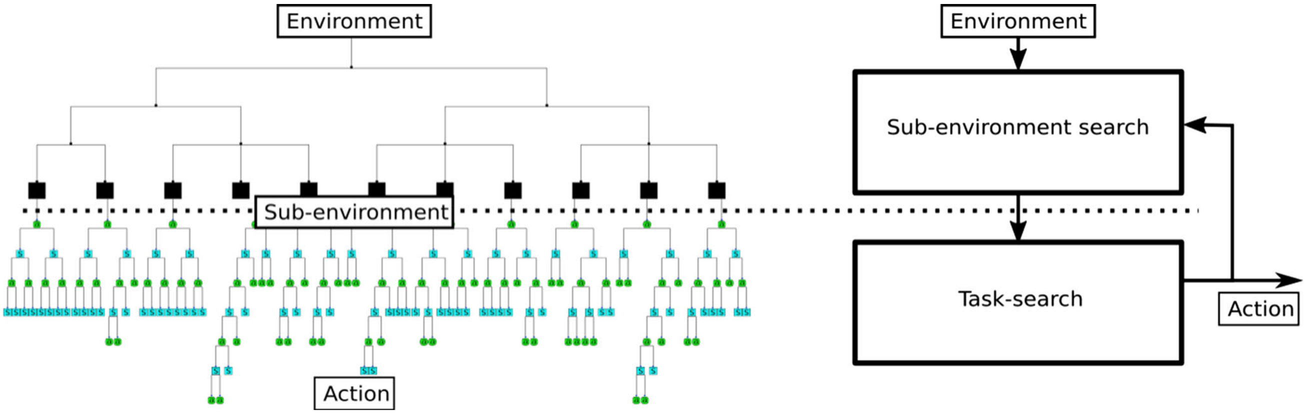

Fig. 1 Workflow of the proposed hierarchical algorithm. A *sub-environment* is dynamically constructed as series of spatial-based state abstractions in the environment in a process called `SubEnvSearch`. Given this sub-environment, an agent performs *tasks*, which are abstracted actions constrained to properties of the sub-environment, in order to satisfy objectives. Once a sub-environment is created, the process `TaskSearch` uses a semi-Markov decision process to model the sub-environment and determine an optimal 'task' to perform. Past experiences are recycled for similar looking sub-environments allowing the agent to quickly converge to an optimal policy. The agent dedicates time to both finding the best sub-environment and evaluating that sub-environment by cycling through `SubEnvSearch` and `TaskSearch`

2 Preliminaries on Markov decision processes

We follow the exposition from Bai et al. (2016) to introduce basic notions on Markov decision processes (MDPs). A MDP is a tuple $\langle S, A, \Pr^s, R\rangle$, where S and A are the state and action spaces, respectively; $\Pr^s(s'|a, s)$ is a transition function that returns the probability that state $s \in S$ becomes state s' after taking action $a \in A$; and $R(s, a, s')$ is the reward that an agent gets after taking action a from state s to reach state s'. A policy is a feedback control that maps each state to an action, $\pi : s \rightarrow a$, for each s. The value of a state under a given policy is

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} \Pr^s(s'|\pi(s), s) V^{\pi}(s'),$$

where $\gamma \in (0, 1)$ is the discount factor. The value function takes finite values. The solution to the MDP is an optimal policy that maximizes the value function, $\pi^*(s) = \operatorname{argmax}_{\pi} V^{\pi}(s)$ for all s. The value of taking an action at a given state under a given policy is

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} \Pr^s(s'|a, s) V^{\pi}(s').$$

Usual methods for obtaining π^* require a tree search of the possible states that can be reached by taking a series of actions. The rate at which the tree of states grows is called the *branching factor*. This search is a challenge for solving MDPs with large state spaces and actions with low-likelihood probabilistic state transitions. A technique often used to decrease the size of the state space is *state abstractions*, where a collection of states are clustered into one in some meaningful way. This can be formalized with a state abstraction function of the form $\phi_s : s \rightarrow s_{\phi}$. Similarly, actions can be abstracted with an action abstraction function $\phi_a : a \rightarrow a_{\phi}$. Abstracting actions is used to decrease the action space, which can make π^* easier to calculate. In MDPs, actions take one time step per action. However, abstracted actions may take a probabilistic amount of time to complete, $\Pr^t(t|a_{\phi}, s)$. When considering the problem using abstracted actions $a_{\phi} \in A_{\phi}$ in $\langle S, A_{\phi}, \Pr^s, R\rangle$, the process becomes a semi-Markov Decision Process (SMDP), which allows for probabilistic time per abstracted action. The loss of precision in the abstracted actions means that an optimal policy for an SMDP with abstracted modifications may not be optimal with respect to the original MDP.

Determining π^* often involves constructing a tree of reachable MDP/SMDP states determined through simulating actions from an initial state. Dynamic programming is commonly used for approximating π^* by using a Monte Carlo tree search to explore the MDP for the initial state. The action with the maximum upper confidence bound (UCB) (Kocsis and Szepesvári 2006) of the approximated expected value for taking the action at a given state,

$$\operatorname*{argmax}_{a \in A} \left\{ \hat{Q}(s, a) + C\sqrt{\frac{\ln(N_s)}{N_{s,a}}} \right\},$$

is often used to efficiently explore the MDP. Here, N_s is the number of times a state has been visited, $N_{s,a}$ is the number of times that action a has been taken at state s and C is a constant. Taking the action that maximizes this quantity balances between exploiting actions that previously had high reward and exploring actions with uncertain but potentially higher reward.

3 Problem statement

Consider a set of agents $\mathcal{A}$ indexed by $\alpha \in \mathcal{A}$. We assume agents are able to communicate with each other and have access to each other's locations. An agent occupies a point in $\mathbb{Z}^d$, and has computational, communication, and mobile capabilities. A *waypoint* $o \in \mathbb{Z}^d$ is a point that an agent must visit in order to serve an objective. Every waypoint then belongs to a class of objective of the form $\mathcal{O}^b = \{o_1, \ldots, o_{|\mathcal{O}^b|}\}$. Agents are able to satisfy objectives when waypoints are visited such that $o \in \mathcal{O}^b$ is removed from $\mathcal{O}^b$. When $\mathcal{O}^b = \emptyset$ the objective is considered 'complete'. An agent receives a reward $r^b \in \mathbb{R}$ for visiting a waypoint $o \in \mathcal{O}^b$. Define the set of objectives to be $\mathcal{O} = \{\mathcal{O}^1, \ldots, \mathcal{O}^{|\mathcal{O}|}\}$, and assume agents are only able to service one $\mathcal{O}^b \in \mathcal{O}$ at a time. We consider the environment to be $\mathcal{E} = \mathcal{O} \times \mathcal{A}$, which contains information about all objectives and agents. The state space of $\mathcal{E}$ increases exponentially with the number of objectives $|\mathcal{O}|$, the cardinality of each objective $|\mathcal{O}^b|$, for each b, and the number of agents $|\mathcal{A}|$.

We strive to design a decentralized algorithm that allows the agents in $\mathcal{A}$ to individually approximate the policy π^* that optimally services objectives in $\mathcal{O}$ in scenarios where $\mathcal{E}$ is very large. To tackle this problem, we rely on abstractions that reduce the stochastic branching factor to find a good policy in the environment. We begin our strategy by spatially abstracting objectives in the environment into convex sets termed 'regions'. We dynamically create and search subsets of the environment to reduce dimensions of the state that individual agents reason over. Then we structure a plan of high-level actions with respect to the subset of the environment. Finally, we specify a limited, tunable number of interactions that must be considered in the multi-agent joint planning problem, leading up to the *Dynamic Domain Reduction for Multi-Agent Planning* algorithm to approximate the optimal policy.

4 Abstractions

In order to leverage dynamic programming solutions to approximate a good policy, we reduce the state space and action space. We begin by introducing methods of abstraction with respect to spatial proximity for a single agent, then move on to the multi-agent case.

4.1 Single-agent abstractions

To tackle the fact that the number of states in the environment grows exponentially with respect to the number of agents, the number of objectives, and their cardinality, we cluster waypoints and agent locations into convex sets in space, a process we term region abstraction. Then, we construct abstracted actions that agents are allowed to execute with respect to the region abstractions.

4.1.1 Region abstraction

We define a *region* to be a convex set $x \subseteq \mathbb{R}^d$. Let Ω_x be a set of disjoint regions where the union of all regions in Ω_x is the entire space that agents reason over in the environment. We consider regions to be equal in size, shape, and orientation, so that the set of regions creates a tessellation of the environment. This makes the presentation simpler, albeit our framework can be extended to handle regions that are non-regular by including region definitions for each unique region in consideration.

Furthermore, let $\mathcal{O}_i^b$ be the set of waypoints of objective $\mathcal{O}^b$ in region x_i, i.e., such that $\mathcal{O}_i^b \subseteq x_i$. We use an abstraction function, $\phi : \mathcal{O}_i^b \rightarrow s_i^b$, to get the *abstracted objective state*, s_i^b, which enables us to generalize the states of an objective in a region. In general, the abstraction function is designed by a human user that distinguishes importance of an objective in a region. We define a *regional state*, $s_i = (s_i^1, \ldots, s_i^{|\mathcal{O}|})$, to be the Cartesian product of s_i^b for all objectives, $\mathcal{O}^b \in \mathcal{O}$, with respect to x_i.

4.1.2 Action abstraction

We assume that low-level feedback controllers allowing for servicing of waypoints are available. We next describe how we use low-level controllers as components of a 'task' to reason over. We define a *task* to be $\tau = \langle s_i^b, s_i'^b, x_i, x_j, b\rangle$, where s_i^b and $s_i'^b$ are abstracted objective states associated to x_i, x_j is a target region, and b is the index of a target objective. Assume that low-level controllers satisfy the following requirements:

- Objective transition: low-level controller executing τ drives the state transition, $\tau.s_i^b \rightarrow \tau.s_i'^b$.
- Regional transition: low-level controller executing τ drives the agent's location to x_j after the objective transition is complete.

Candidates for low-level controllers include policies determined using the approaches in Bai et al. (2016), Kocsis and Szepesvári (2006) after setting up the region as a MDP, modified traveling salesperson (Blum et al. 2007), or path planning-based algorithms interfaced with the dynamics of the agent. Agents that start a task are required to continue working on the task until requirements are met. Because the tasks are dependent on abstracted objectives states, the agent completes a task in a probabilistic time, given by $\Pr^t(t|\tau)$, that is determined heuristically. The set of all possible tasks is given by Γ. If an agent begins a task such that the following properties are not satisfied, then $\Pr^t(\infty|\tau) = 1$ and the agent never completes it.

4.1.3 Sub-environment

In order to further alleviate the curse of dimensionality, we introduce sub-environments, a subset of the environment, in an effort to only utilize relevant regions. A sub-environment is composed of a sequence of regions and a state that encodes proximity and regional states of those regions. Formally, we let the *sub-environment region sequence*, $\overrightarrow{x}$, be a sequence of regions of length up to $N_\epsilon \in \mathbb{Z}_{\geq 1}$. The k^{th} region in $\overrightarrow{x}$ is denoted with $\overrightarrow{x}_k$. The regional state of $\overrightarrow{x}_k$ is given by $s_{\overrightarrow{x}_k}$. For example, $\overrightarrow{x} = [x_2, x_1, x_3]$ is a valid sub-environment region sequence with $N_\epsilon \geq 3$, the first region in $\overrightarrow{x}$ is $\overrightarrow{x}_1 = x_2$, and the regional state of $\overrightarrow{x}_1$ is $s_{\overrightarrow{x}_1} = s_{x_2}$. In order to simulate the sub-environment, the agent must know if there are repeated regions in $\overrightarrow{x}$. Let $\xi(k, \overrightarrow{x})$, return the first index h of $\overrightarrow{x}$ such that $\overrightarrow{x}_h = \overrightarrow{x}_k$. Define the repeated region list to be $\xi_{\overrightarrow{x}} = [\xi(1, \overrightarrow{x}), \ldots, \xi(N_\epsilon, \overrightarrow{x})]$. Let $\phi_t(x_i, x_j) : x_i, x_j \rightarrow \mathbb{Z}$ be an abstracted amount of time it takes for an agent to move from x_i to x_j, or ∞ if no path exists. Let $s = [s_{\overrightarrow{x}_1}, \ldots, s_{\overrightarrow{x}_{N_\epsilon}}] \times \xi_{\overrightarrow{x}} \times [\phi_t(\overrightarrow{x}_1, \overrightarrow{x}_2), \ldots, \phi_t(\overrightarrow{x}_{N_\epsilon - 1,}, \overrightarrow{x}_{N_\epsilon})]$ be the *sub-environment state* for a given $\overrightarrow{x}$, and let S^ϵ be the set of all possible sub-environment states. We define a *sub-environment* to be $\epsilon = \langle \overrightarrow{x}, s\rangle$.

In general, we allow a sub-environment to contain any region that is reachable in finite time. However, in practice, we only allow agents to choose sub-environments that they can traverse within some reasonable time in order to reduce the number of possible sub-environments and save onboard memory. In what follows, we use the notation $\epsilon.s$ to denote the sub-environment state of sub-environment ϵ.

4.1.4 Task trajectory

We also define an ordered list of tasks that the agents execute with respect to a sub-environment ϵ. Let $\overrightarrow{\tau} = [\overrightarrow{\tau}_1, \ldots, \overrightarrow{\tau}_{N_\epsilon - 1}]$ be an ordered list of feasible tasks such that $\overrightarrow{\tau}_k \cdot x_i = \epsilon \cdot \overrightarrow{x}_k$, and $\overrightarrow{\tau}_k \cdot x_j = \epsilon \cdot \overrightarrow{x}_{k+1}$ for all $k \in \{1, \ldots, N_\epsilon - 1\}$, where x_i and x_j are the regions in the definition of task $\overrightarrow{\tau}_k$. Agents generate this ordered list of tasks assuming that they will execute each of them in order. The probability distribution on the time of completing the k^{th} task in $\overrightarrow{\tau}$ (after completing all previous tasks in $\overrightarrow{\tau}$) is given by $\overrightarrow{\Pr^t}_k$. We define $\overrightarrow{\Pr^t} = [\overrightarrow{\Pr^t}_1, \ldots, \overrightarrow{\Pr^t}_{N_\epsilon - 1}]$ to be the ordered list of probability distributions. We construct the *task trajectory* to be $\vartheta = \langle \overrightarrow{\tau}, \overrightarrow{\Pr^t} \rangle$, which is used to determine the finite time reward for a sub-environment.

4.1.5 Sub-environment SMDP

As tasks are completed, the environment evolves, so we denote $\mathcal{E}'$ as the environment after an agent has performed a task. Because the agents perform tasks that reason over abstracted objective states, there are many possible initial, and outcome environments. The exact reward that an agent receives when acting on the environment is a function of $\mathcal{E}$ and $\mathcal{E}'$, which is complex to determine by our use of abstracted objective states. We determine the reward an agent receives for completing τ as a probabilistic function Pr^r that is determined heuristically. Let r^ϵ be the *abstracted reward function*, determined by

$$r^\epsilon(\tau) = \sum_{r \in \mathbb{R}} Pr^r(r|\tau)r, \tag{4.1}$$

which is the expected reward for completing τ given the state of the sub-environment. Next we define the *sub-environment evolution procedure*. Note that agents must begin in the region $\epsilon.\overrightarrow{x}_1$ and end up in region $\epsilon.\overrightarrow{x}_2$ by definition of task. Evolving a sub-environment consists of 2 steps. First, the first element of the sub-environment region sequence is removed. The length of the sub-environment sequence $\epsilon.\overrightarrow{x}$ is reduced by 1. Next, the sub-environment state $\epsilon.s$ is recalculated with the new sub-environment region sequence. To do this, we draw the sub-environment state from a probability distribution and we determine the sub-environment after completing the task

$$\epsilon' = \langle \overrightarrow{x} = [\overrightarrow{x}_2, \ldots, \overrightarrow{x}_{N_\epsilon}],\ s = \Pr^s(\epsilon'.s|\epsilon.s, \tau) \rangle. \tag{4.2}$$

Finally, we can represent the process of executing tasks in sub-environments as the *sub-environment SMDP* $\mathcal{M} = \langle S^\epsilon, \Gamma, Pr^{s^\epsilon}, r^\epsilon, \Pr^t \rangle$. The goal of the agent is to determine a policy $\pi_\epsilon : \epsilon.s \to \tau$ that yields the greatest rewards in $\mathcal{M}$. The state value under policy π_ϵ is given by

$$V^{\pi_\epsilon}(\epsilon.s) = r^\epsilon + \sum_{t^\epsilon \in \mathbb{R}} Pr^{t^\epsilon} \gamma^{t^\epsilon} \sum_{\epsilon.s \in S^\epsilon} Pr^s V^{\pi_\epsilon}(\epsilon.s') \tag{4.3}$$

We strive to generate a policy that yields optimal state value

$$\pi^*_\epsilon(\epsilon.s) = \underset{\pi_\epsilon}{\operatorname{argmax}}\ V^{\pi_\epsilon}(\epsilon.s),$$

with associated optimal value $V^{\pi_\epsilon *}(\epsilon.s) = \max_{\pi_\epsilon} V^{\pi_\epsilon}(\epsilon.s)$.

Remark 4.1 (Extension to heterogeneous swarms) The framework described above can also handle UxV agents with heterogeneous capabilities. In order to do this, one can consider the possibility of any given agent having a unique set of controls which allow it to complete some tasks more quickly than others. The agents use our framework to develop a policy that maximizes their rewards with respect to their own capability, which is implicitly encoded in the reward function. For example, if an agent chooses to serve some objective and has no low level control policy that can achieve it, $\overrightarrow{\Pr^t}_k(\infty) = 1$, and the agent will never complete it. In this case, the agent would naturally receive a reward of 0 for the remainder of the trajectory. □

4.2 Multi-agent abstractions

Due to the large environment induced by the action coupling of multi-agent joint planning, determining the optimal policy is computationally unfeasible. To reduce the computational burden on any given agent, we restrict the number of coupled interactions. In this section, we modify the sub-environment, task trajectory, and rewards to allow for multi-agent coupled interactions. The following discussion is written from the perspective of an arbitrary agent labeled α in the swarm, where other agents are indexed with $\beta \in \mathcal{A}$.

4.2.1 Sub-environment with interaction set

Agent α may choose to interact with other agents in its *interaction set* $\mathcal{I}_\alpha \subseteq \mathcal{A}$ while executing $\overrightarrow{\tau}_\alpha$ in order to more effectively complete the tasks. The interaction set is constructed as a parameter of the sub-environment and indicates to the agent which tasks should be avoided based on the other agents' trajectories. Let $\mathcal{N}$ be a (user specified) maximum number of agents that an agent can interact with (hence $|\mathcal{I}_\alpha| \leq \mathcal{N}$ at all times). The interaction set is updated by adding another agent β and their interaction set, $\mathcal{I}_\alpha = \mathcal{I}_\alpha \cup \{\beta\} \cup \mathcal{I}_\beta$. If $|\mathcal{I}_\alpha \cup \{\beta\} \cup \mathcal{I}_\beta| > \mathcal{N}$, then we consider agent β to be an invalid candidate. Adding β's interaction set is necessary because tasks that affect the task trajectory ϑ_β may also affect all agents in $\mathcal{I}_\beta$. Constraining the maximum interaction set size reduces the large state size that occurs when agents' actions are coupled. To avoid interacting with agents not in the interaction set, we create a set of waypoints that are off-limits when creating a trajectory.

We define a *claimed regional objective* as $\theta = \langle \mathcal{O}^b, x_i \rangle$. The agent creates a set of claimed region objectives $\Theta_\alpha = \{\theta_1, \ldots, \theta_{N_\epsilon - 1}\}$ that contains a claimed region objective for every task in its trajectory and describes a waypoint in $\mathcal{O}^b$ in a region that the agent is planning to service. We define the *global claimed objective set* to be $\Theta^{\mathcal{A}} = \{\Theta_1, \ldots, \Theta_{|\mathcal{A}|}\}$, which contains the claimed region objective set for all agents. Lastly, let $\Theta'_\alpha = \Theta^{\mathcal{A}} \backslash \big\{ \bigcup_{\beta \in \mathcal{I}_\alpha} \Theta_\beta \big\}$ be the complete set of claimed objectives an agent must avoid when planning a trajectory. The agent uses Θ'_α to modify its perception of the environment. As shown in the following function, the agent sets the state of claimed objectives in Θ'_α to 0, removing appropriate tasks from the feasible task set.

$$s^b_{\Theta'_\alpha, \vec{x}_k} = \begin{cases} 0 & \text{if } \langle \mathcal{O}^b, \epsilon_\alpha . \vec{x}_k \rangle \in \Theta'_\alpha, \\ s^b_{\vec{x}_k} & \text{otherwise.} \end{cases} \tag{4.4}$$

Let $\vec{\mathcal{S}}_{\Theta'_\alpha, \vec{x}} = \vec{\mathcal{S}}_{\Theta'_\alpha, \vec{x}_1} \times \cdots \times \vec{\mathcal{S}}_{\Theta'_\alpha, \vec{x}_{N_\epsilon}}$, where $\vec{\mathcal{S}}_{\Theta'_\alpha, \vec{x}_k} = (s^{\mathcal{O}^1}_{\Theta'_\alpha, \vec{x}_k}, \ldots, s^{\mathcal{O}^{|\mathcal{O}|}}_{\Theta'_\alpha, \vec{x}_k})$. In addition to the modified sub-environment state, we include the partial trajectories of other agents being interacted with. Consider β's trajectory ϑ_β and an arbitrary ϵ_α. Let $\vartheta^p_{\beta,k} = \langle \vartheta_\beta . \vec{\tau}_k . s^b_i, \ \vartheta_\beta . \vec{\tau}_k . b \rangle$. The *partial trajectory*, $\vartheta^p_\beta = [\vartheta^p_{\beta,1}, \ldots, \vartheta^p_{\beta,|\vartheta_\beta|}]$ describes β's trajectory with respect to $\epsilon_\alpha . \vec{x}$. Let $\xi(k, \epsilon_\alpha, \epsilon_\beta)$ return the first index of $\epsilon_\alpha . \vec{x}$, h, such that $\epsilon_\beta . \vec{x}_h = \epsilon_\alpha . \vec{x}_k$, or 0 if there is no match. Each agent in the interaction set creates a matrix Ξ of elements, $\xi(k, \epsilon_\alpha, \epsilon_\beta)$, for $k \in \{1, \ldots, N_\epsilon\}$ and $\beta \in \{1, \ldots, |\mathcal{I}_\alpha|\}$. We finally determine the *complete multi-agent state*, $s = \vec{\mathcal{S}}_{\Theta'_\alpha, \vec{x}} \times \big\{ \big\langle \vartheta^p_1, \Xi_1 \big\rangle, \ldots, \langle \vartheta^p_{|\mathcal{I}_\alpha|}, \Xi_{|\mathcal{I}_\alpha|} \rangle \big\} \times [\phi_t(\vec{x}_1, \vec{x}_2), \ldots, \phi_t(\vec{x}_{N_\epsilon - 1}, \vec{x}_{N_\epsilon})]$. With these modifications to the sub-environment state, we define the (multi-agent) *sub-environment* as $\epsilon_\alpha = \langle \vec{x}, s, \mathcal{I}_\alpha \rangle$.

4.2.2 Multi-agent action abstraction

We consider the effect that an agent α has on another agent $\beta \in \mathcal{A}$ when executing a task that affects s^b_i in β's trajectory. Some tasks will be completed sooner with two or more agents working on them, for instance. For all $\beta \in \mathcal{I}_\alpha$, let t_β be the time that β begins a task that transitions $s^b_i \rightarrow s'^b_i$. If agent β does not contain such a task in its trajectory, then $t_\beta = \infty$. Let $T^{\mathcal{A}}_{\mathcal{I}_\alpha} = [t_1, \ldots, t_{|\mathcal{I}_\alpha|}]$. We denote by $\text{Pr}^t_{\mathcal{I}_\alpha}(t | \tau, T^{\mathcal{A}}_{\mathcal{I}_\alpha})$ the probability that τ is completed at exactly time t if other agents work on transitioning $s^b_i \rightarrow s'^b_i$. We redefine here the definition of $\overrightarrow{\text{Pr}^t}$ in Sect. 4.1.4, as $\overrightarrow{\text{Pr}^t} = [\overrightarrow{\text{Pr}^t}_{1,\mathcal{I}_\alpha}, \ldots, \overrightarrow{\text{Pr}^t}_{N_\epsilon - 1, \mathcal{I}_\alpha}]$, which is the probability time set of α modified by accounting for other agents trajectories. Furthermore, if an agent chooses a task that modifies agent α's trajectory, we define the probability time set to be $\overrightarrow{\text{Pr}^{t}}' = [\overrightarrow{\text{Pr}^{t}}'_{1,\mathcal{I}_\alpha}, \ldots, \overrightarrow{\text{Pr}^{t}}'_{N_\epsilon - 1, \mathcal{I}_\alpha}]$. With these modifications, we redefine the *task trajectory* to be $\vartheta = \Big\langle \vec{\tau}, \overrightarrow{\text{Pr}^t} \Big\rangle$. Finally, we designate X_ϑ to be the set that contains all trajectories of the agents.

4.2.3 Multi-agent sub-environment SMDP

We modify the reward abstraction so that each agent takes into account agents that it may interact with. When α interacts with other agents, it modifies the expected discounted reward gained by those agents. We define the *interaction reward function*, which returns a reward based on whether the agent executes a task that interacts with one or more other agents. The interaction reward function is defined as

$$r^\phi(\tau, X_\vartheta) = \begin{cases} \mathfrak{R}(\tau, X_\vartheta) & \text{if } \ \tau \in \vartheta^p_\beta \text{ for any } \beta, \\ r^\epsilon(\tau) & \text{otherwise.} \end{cases} \tag{4.5}$$

Here, the term $\mathfrak{R}$ represents a designed reward that that the agent receives for completing τ when it is shared by by other agents. This expression quantifies the effect that an interacting task has on an existing task. If a task helps another agent trajectory in a significant way, the agent may choose a task that aids the global expected reward amongst the agents. Let the *multi-agent sub-environment SMDP* be defined as the tuple $\mathcal{M} = \langle S^\epsilon, \Gamma, \text{Pr}^s, r^\phi, \text{Pr}^t_{\mathcal{I}_\alpha} \rangle$. The state value from (4.3) is updated using (4.5)

$$V^{\pi_\epsilon}(\epsilon.s) = r^\phi + \sum_{t^\epsilon \in \mathbb{R}} \text{Pr}^{t^\epsilon} \gamma^{t^\epsilon} \sum_{\epsilon.s \in S'^\epsilon} \text{Pr}^s V^{\pi_\epsilon}(\epsilon.s'). \tag{4.6}$$

We strive to generate a policy that yields optimal state value

$$\pi^*_\epsilon(\epsilon.s) = \underset{\pi_\epsilon}{\text{argmax}} \ V^{\pi_\epsilon}(\epsilon.s),$$

with associated optimal value $V^{\pi_\epsilon *}(\epsilon.s) = \max\limits_{\pi_\epsilon} V^{\pi_\epsilon}(\epsilon.s)$. Our next section introduces an algorithm for approximating this optimal state value.

5 Dynamic domain reduction for multi-agent planning

This section describes our algorithmic solution to approximate the optimal policy π^*_ϵ. The *Dynamic Domain Reduction for Multi-Agent Planning* algorithm consists of three main functions: `DDRMAP`, `TaskSearch`, and `SubEnvSearch`.[1]

[1] pseudocode of functions denoted with † is omitted but described in detail.

Algorithm 1 presents a formal description in the multi-agent case, where each agent can interact with a maximum of $\mathcal{N}$ other agents for planning purposes. In the case of a single agent, we take $\mathcal{N} = 0$ and refer to our algorithm as *Dynamic Domain Reduction Planning* (`DDRP`). In what follows, we first describe the variables and parameters employed in the algorithm and then discuss each of the main functions and the role of the support functions.

Algorithm 1: : Dynamic Domain Reduction for Multi-Agent Planning

```
1  $\Omega_x = \bigcup \{x\}\ \forall x$
2  $\mathcal{E} \leftarrow$ current environment
3  $\Theta^{\mathcal{A}} \leftarrow \bigcup \Theta_\beta,\ \ \forall \beta \in \mathcal{A}$
4  $\hat{Q}, V_x, N_{\epsilon.s}, N_b \leftarrow$ loaded from previous trials
5  DDRMAP ($\Omega_x, N_\epsilon, \mathcal{E}, \Theta^{\mathcal{A}}, \hat{Q}, V_x, N_{\epsilon.s}, N_b, \mathcal{M}$):
6      $Y_\vartheta = \emptyset$
7      while run time $<$ step time:
8          $\epsilon \leftarrow$SubEnvSearch ($\Omega_x, \mathcal{E}, \Theta^{\mathcal{A}}, V_x$)
9          TaskSearch ($\hat{Q}, N_{\epsilon.s}, N_b, \epsilon$)
10         $Y_\vartheta = Y_\vartheta \cup \{$MaxTrajectory$(\epsilon)\}$
11     return $Y_\vartheta$

12 TaskSearch ($\hat{Q}, N_{\epsilon.s}, N_b, \epsilon$)
13     if $\epsilon.\overrightarrow{x}$ is empty
14         return 0
15     $\tau \leftarrow \max_{\tau \in \mathcal{M}.\Gamma} \left\{ \hat{Q}[\epsilon.s][\tau.b] + 2C_p \sqrt{\frac{ln N_{\epsilon.s}[\epsilon.s]}{N_b[\epsilon.s][\tau.b]}} \right\}$
16     $t \leftarrow$ Sample$^\dagger$ $\mathcal{M}.\mathrm{Pr}^t(t|\tau, \epsilon)$
17     $\epsilon' = \langle [\epsilon.\overrightarrow{x}_2 \ldots, \epsilon \overrightarrow{x}_{N_\epsilon}]$,Sample$^\dagger$ $\mathcal{M} \cdot Pr^s(\epsilon.s, \tau) \rangle$
18     $r = \mathcal{M}.r^\epsilon(\tau, \epsilon) + \gamma^t$TaskSearch ($\hat{Q}, N_{\epsilon.s}, N_b, \epsilon'$)
19     TaskValueUpdate ($\hat{Q}, N_{\epsilon.s}, N_b, \epsilon.s, \tau.b, r$)
20     return $r$

21 TaskValueUpdate ($N_{\epsilon.s}, N_b, \hat{Q}, \epsilon.s, \tau.b, r$)
22     $N_{\epsilon.s}[\epsilon.s] = N_{\epsilon.s}[\epsilon.s] + 1$
23     $N_b[\epsilon.s][\tau.b] = N_b[\epsilon.s][\tau.b] + 1$
24     $\hat{Q}[\epsilon.s][\tau.b] = \hat{Q}[\epsilon.s][\tau.b] + \frac{1}{N_b[\epsilon.s][\tau.b]}(r - \hat{Q}[\epsilon.s][\tau.b])$

25 SubEnvSearch ($\Omega_\epsilon, \mathcal{E}, \Theta^{\mathcal{A}}, V_x$)
26     $X_x = \emptyset$
27     while $|X_x| < N_\epsilon$:
28         for $x \in \Omega_\epsilon$:
29             if $V_x[X_x \cup \{x\}]$ is empty:
30                 $V_x[X_x \cup \{x\}] =$
                   $\left\{ \max_{\tau \in \mathcal{M}.\Gamma} \hat{Q}[$InitSubEnv$(X_x, \mathcal{E}, \Theta^{\mathcal{A}}).S][\tau] \right\}$
31         $x = \mathrm{argmax}_{x \in \Omega_\epsilon}\ V_x[X_x \cup \{x\}]$
32         $X_x = X_x \cup \{x\}$
33     return InitSubEnv$(X_x, \mathcal{E}, \Theta^{\mathcal{A}})$

34 InitSubEnv ($X_x, \mathcal{E}, \Theta^{\mathcal{A}}$)
35     $\overrightarrow{x} =$
       $\mathrm{argmax}_{\overrightarrow{x} \in \binom{X_x}{|X_x|} \setminus \mathbb{V}_\epsilon} \left\{ \max_{\tau \in \mathcal{M}.\Gamma} \hat{Q}[\text{GetSubEnvState}(\overrightarrow{x}, \Theta^{\mathcal{A}})][\tau] \right.$
36     $\epsilon = \langle \overrightarrow{x}$, GetSubEnvState$(\overrightarrow{x}, \Theta^{\mathcal{A}}) \rangle$
37     return $\epsilon$
```

The following variables are common across the multi-agent system: the set of regions Ω_x, the current environment $\mathcal{E}$, the claimed objective set $\Theta^{\mathcal{A}}$, and the multi-agent sub-environment SMDP $\mathcal{M}$. Some variables can be loaded, or initialized to zero such as the number of times an agent has visited a state $N_{\epsilon.s}$, the number of times an agent has taken an action in a state N_b, the estimated value of taking an action in a state $\hat{Q}$, and the estimated value of selecting a region in the sub-environment search process V_x. The set $\mathbb{V}_\epsilon$ contains sub-environments as they are explored by the agent.

The main function `DDRMAP` structures Q-learning with domain reduction of the environment. In essence, `DDRMAP` maps the environment into a sub-environment where it can use a pre-constructed SMDP and upper confidence bound tree search to determine the value of the sub-environment. `DDRMAP` begins by initializing the set of constructed trajectories Y_ϑ as an empty set. The function uses `SubEnvSearch` to find a suitable sub-environment from the given environment, then `TaskSearch` is used to evaluate that sub-environment. `MaxTrajectory` constructs a trajectory using the sub-environment, which is added to Y_ϑ. This process is repeated for an allotted amount of time. The function returns the set of constructed trajectories Y_ϑ.

`TaskSearch` is a modification on Monte Carlo tree search. Given sub-environment ϵ, the function finds an appropriate task ϑ to exploit and explore the SMDP. On line 15 we select a task based on the upper confidence bound of $\hat{Q}$. We simulate executing the task by sampling Pr^t for the amount of time it takes to complete the task. We then evolve the sub-environment to get ϵ' by following the sub-environment evolution procedure (4.2). On line 18, we get the discounted reward of the sub-environment by summing the reward for the current task using (4.1,4.5) and the reward returned by recursively calling `TaskSearch` with the sampled evolution of the sub-environment ϵ' at a discount. The recursive process is terminated at line 13 when the sub-environment no longer contains regions in $\epsilon.\overrightarrow{x}$. `TaskValueUpdate` is called after each recursion and updates $N_{\epsilon.s}$, N_b, and $\hat{Q}$. On line 24, $\hat{Q}$ is updated by recalculating the average reward over all experiences given the task and sub-environment state, which is done by using N_b^{-1} as the learning rate. The time complexity of `TaskSearch` is $O(|\Gamma|N_\epsilon)$ due to the task selection on Line 15 and the recursive depth of the size of the sub-environment N_ϵ.

We employ `SubEnvSearch` to explore and find the value of possible sub-environments in the environment. The function `InitSubEnv` maps a set of regions $X_x \subseteq \Omega_x$ (we use subindex 'x' to emphasize that this set contains regions) to the sub-environment with the highest expected reward. We do this by finding the sequence of regions $\overrightarrow{x}$ given a X_x that maximizes $\max_{\tau \in \Gamma} \hat{Q}[\epsilon.s][\tau]$ on line 35. We keep track of the expected value of choosing X_x and the sub-environment that is returned by `InitSubEnv` with V_x. `SubEnvSearch` begins by initializing X_x to empty. The region that increases the value V_x the most when appended to X_x is then appended to X_x. This process is repeated until the length of X_x is N_ϵ. Finally, the best sub-environment given X_x is returned with `InitSubEnv`. The time complexity of `InitSubEnv` and `SubEnvSearch` is $O(N!)$ and $O(|\Omega_x| N_\epsilon \log(N_\epsilon!))$, respectively. `InitSubEnv` requires iteration over all possible permutations of X_x, however in practice we use heuristics to reduce the time of computation.

6 Convergence and performance analysis

In this section, we look at the performance of individual elements of our algorithm. First, we establish the convergence of the estimated value of performing a task determined over time using `TaskSearch`. We build on this result to characterize the performance of the `SubEnvSearch` and of sequential multi-agent deployment.

6.1 `TaskSearch` estimated value convergence

We start by making the following assumption about the sub-environment SMDP.

Assumption 6.1 There always exists a task that an agent can complete in finite time. Furthermore, no task can be completed in zero time steps.

Assumption 6.1 is reasonable because if not true, then the agent's actions are irrelevant and the scenario is trivial. The following result characterizes long term performance of the function `TaskSearch` which is necessary for the analysis of other elements of the algorithm.

Theorem 6.1 *Let $\hat{Q}$ be the estimated value of performing a task determined over time using `TaskSearch`. Under Assumption 6.1, $\hat{Q}$ converges to the optimal state value $V^{*\epsilon}$ of the sub-environment SMDP with probability 1.*

Proof SMDP Q-learning converges to the optimal value under the following conditions (Parr and Russell 1998), rewritten here to match our notation:

(i) State and action spaces are finite;
(ii) $\mathrm{Var}\{r^\epsilon\}$ is finite;
(iii) $\sum_{p=1}^{\infty} \alpha_p(\epsilon.s, \tau) = \infty$ and $\sum_{p=1}^{\infty} \alpha_p^2(\epsilon.s, \tau) < \infty$ uniformly over $\epsilon.s, \tau$;
(iv) $0 < \mathcal{B}^{\max} = \max_{\epsilon.s \in S^\epsilon, \tau \in \Gamma} \sum \Pr^t(t|\epsilon.s\tau)\gamma^t < 1$.

In the construction of the sub-environment SMDP, we assume that sub-environment lengths and number of objectives are finite, satisfying (i). We reason over the expected value of the reward in $\mathbb{R}$, that is determined heuristically, which implies that $\mathrm{Var}\{r^\epsilon\} = 0$ satisfying (ii). From `TaskValueUpdate` on line 24, we have that $\alpha_p(\epsilon.s, \tau) = 1/p$ if we substitute p for N_b. Therefore, (iii) is satisfied because $\sum_{N_b=1}^{\infty} \frac{1}{N_b} = \infty$ and $\sum_{N_b=1}^{\infty} (\frac{1}{N_b})^2 = \pi^2/6$ (finite). Lastly, to satisfy (iv), we use Assumption 6.1 (there always exists some τ such that $\Pr^t(\infty|\epsilon.s, \tau) < 1$) and the fact that $\gamma \in (0, 1)$ to ensure that $\mathcal{B}^{\max}$ will always be greater than 0. We use Assumption 6.1 (for all $\epsilon.s, \tau$ $\Pr^t(t > 0|\epsilon.s\tau) = 1$) to ensure that $\mathcal{B}^{\max}$ is always less than 1. □

In the following, we consider a version of our algorithm that is trained offline called *Dynamic Domain Reduction Planning:Online+Offline* (`DDRP-OO`). `DDRP-OO` utilizes data that it learned from previous experiments in similar environments. In order to do this, we train `DDRP` offline and save the state values for online use. We use the result of Theorem 6.1 as justification for the following assumption.

Assumption 6.2 Agents using `DDRP-OO` are well-trained, i.e., $\hat{Q} = V^{*\epsilon}$.

In practice, we accomplish this by running `DDRP` on randomized environments until $\hat{Q}$ remains unchanged for a substantial amount of time, an indication that it has nearly converged to $V^{*\epsilon}$. The study of `DDRP-OO` gives insight on the tree search aspect of finding a sub-environment in `DDRP` and gives intuition on its long-term performance.

6.2 Sub-environment search by a single agent

We are interested in how well the sub-environments are chosen with respect to the best possible sub-environment. In our study, we make the following assumption.

Assumption 6.3 Rewards are positive. Objectives are *uncoupled*, meaning that the reward for completing one objective is independent of the completion of any other objective. Furthermore, objectives only require one agent's service for completion.

Our technical approach builds on the submodularity framework, cf. "Appendix A", to establish analytical guarantees of the sub-environment search. The basic idea is to show that the algorithmic components of this procedure can be cast as a greedy search with respect to a conveniently defined set function. Let Ω_x be a finite set of regions. `InitSubEnv` takes a set of regions $X_x \subseteq \Omega_x$ and returns the sub-environment made up of regions in X_x in optimal order. We define the power set function $f_x : 2^{\Omega_x} \to \mathbb{R}$, mapping each set of regions to the discounted reward that an agent expects to receive for choosing the corresponding sub-environment

$$f_x(X_x) = \max_{\tau \in \Gamma} \hat{Q}[\texttt{InitSubEnv}(X_x).s][\tau]. \qquad (6.1)$$

For convenience, let $X_x^* = \operatorname{argmax}_{X_x \in \Omega_x} f_x(X_x)$ denote the set of regions that yields the sub-environment with the highest expected reward amongst all possible sub-environments. The following counterexample shows that f_x is, in general, not submodular.

Lemma 6.1 *Let f_x be the discounted reward that an agent expects to receive for choosing the corresponding sub-environment given a set of regions, as defined in (6.1). Under Assumptions 6.2–6.3, f_x is not submodular.*

Proof We provide a counterexample to show that f_x is not submodular in general. Consider a 1-dimensional environment. For simplicity, let regions be singletons, where $\Omega_x = \{x_0 = \{0\}, x_1 = \{-1\}, x_2 = \{1\}, x_3 = \{2\}\}$. Assume that only one objective exists, which is to enter a region. For this objective, agents only get rewarded for entering a region the first time. Let the time required to complete a task be directly proportional to the distance between region, $t = |\tau.x_i - \tau.x_j|$. Let $X_x = \{x_1\} \subset Y_x = (x_1, x_3) \subset \Omega_x$. f_x is submodular only if the marginal gain including $\{x_2\}$ is greater for X_x than Y_x. Assuming that the agent begins in x_0, one can verify that the region sequences of the sub-environments returned by `InitSubEnv` are as follows:

$$\begin{aligned}
&\texttt{InitSubEnv}(X_x) \rightarrow \overrightarrow{x} = [x_1]\\
&\texttt{InitSubEnv}(X_x \cup \{x_2\}) \rightarrow \overrightarrow{x} = [x_1, x_2]\\
&\texttt{InitSubEnv}(Y_x) \rightarrow \overrightarrow{x} = [x_1, x_3]\\
&\texttt{InitSubEnv}(Y_x \cup \{x_2\}) \rightarrow \overrightarrow{x} = [x_2, x_3, x_1]
\end{aligned}$$

Assuming that satisfying each task yields the same reward r we can calculate the marginal gains as

$$\begin{aligned}
&f_x(X_x \cup \{x_2\}) - f_x(X_x)\\
&\quad = (\gamma^{t_1} r + \gamma^{t_1}\gamma^{t_2} r) - (\gamma^{t_1} r) \approx .73r,
\end{aligned}$$

evaluated at $t_1 = x_1 - x_0 = 1$ and $t_2 = x_2 - x_1 = 2$. The marginal gains for appending $\{x_2\}$ to Y_x is

$$\begin{aligned}
&f_x(Y_x \cup \{x_2\}) - f_x(Y_x)\\
&\quad = (\gamma^{t_3} r + \gamma^{t_3}\gamma^{t_4} r + \gamma^{t_3}\gamma^{t_4}\gamma^{t_5} r)\\
&\qquad - (\gamma^{t_1} r + \gamma^{t_1}\gamma^{t_2} r) \approx .74r,
\end{aligned}$$

evaluated at $t_1 = x_1 - x_0 = 1$, $t_2 = x_3 - x_1 = 3$, $t_3 = x_2 - x_0 = 1$, $t_4 = x_3 - x_2 = 1$, and $t_5 = x_1 - x_3 = 3$, showing that the marginal gain for including $\{x_2\}$ is greater for Y_x than X_x. Hence, f_x is not submodular. □

Even though f_x is not submodular in general, one can invoke the notion of submodularity ratio to provide a guaranteed lower bound on the performance of the sub-environment search. According to (A.4), the submodularity ratio of a function f_x is the largest scalar $\lambda \in [0, 1]$ such that

$$\lambda \leq \frac{\sum_{z \in Z_x} f_x(X_x \cup \{z\}) - f_x(X_x)}{f_x(X_x \cup Z_x) - f_x(X_x)} \tag{6.2}$$

for all $X_x, Z_x \subseteq \Omega_x$. This ratio measures how far the function is from being submodular. The following result provides a guarantee on the expected reward with respect to the optimal sub-environment choice in terms of the submodularity ratio.

Theorem 6.2 *Let X_x be region set returned by the sub-environment search algorithm in `DDRP-OO`. Under Assumptions 6.2–6.3, it holds that $f_x(X_x) \geq (1 - e^{-\lambda}) f_x(X_x^*)$.*

Proof According to Theorem A.3, we need to show that f_x is a monotone set function, that $f_x(\emptyset) = 0$, and that the sub-environment search algorithm has greedy characteristics. Because of Assumption 6.3, adding regions to X_x monotonically increases the expected reward, hence Eq. (A.2) is satisfied. Next, we note that if $X_x = \emptyset$, then `InitSubEnv` returns an empty sub-environment, which implies from Eq. (6.1) that $f_x(\emptyset) = 0$. Lastly, by construction, the first iteration of the sub-environment search adds regions to the sub-environment one at a time in a greedy fashion. Because the sub-environment search keeps in memory the sub-environment with the highest expected reward, the entire algorithm is lower bounded by the first iteration of the sub-environment search. The result now follows from Theorem A.3. □

Algorithm 2: : Submodularity ratio estimation

```
$\Delta \leftarrow$ set of all possible pairs of $X_x, Z_x$.
$Pr^\delta \leftarrow$ probability distribution of sampling $\delta$ from $\Delta$.

submodulariyRatioEstimation($\varpi, \varepsilon, \Omega_x, \Delta, Pr^\delta$)
    $\lambda^+ = 1, d = 1, h = \emptyset$
    $N_{\text{SCP}} = \frac{2}{\varepsilon}(ln\frac{1}{\varpi} + d)$
    for $n = 0; n{+}{+}; n < N_{\text{SCP}}$
        $X_x, Z_x =$Sample$^\dagger$ $(Pr^\delta)$
        $\lambda^\delta = \frac{\sum_{z \in Z_x} f_x(X_x \cup \{z\}) - f_x(X_x)}{f_x(X_x \cup Z_x) - f_x(X_x)}$
        if $\lambda^\delta < \lambda^+$
            $\lambda^+ = \lambda^\delta$
            $h$.append$(\lambda^\delta, n)$
    $a, b = \operatorname{argmin}_{a,b \in \mathbb{R}} \sum_{\lambda^\delta, n \in h} (\lambda^+ - a - bn)^2$
    $\hat{\lambda} = a + b|\Delta|$
    return $\hat{\lambda}$
```

Given the generality of our proposed framework, the submodularity ratio of f_x is in general difficult to determine. To deal with this, we resort to tools from scenario optimization, cf. "Appendix B", to obtain an estimate of the submodularity ratio. The basic observation is that, from its definition, the computation of the submodularity ratio can be cast as a robust convex optimization problem. Solving this optimization problem is difficult given the large number of constraints that need to be considered. Instead, the procedure for estimating the submodularity ratio samples the current environment that an agent is in, randomizing optimization parameters. The human supervisor chooses confidence and violation parameters, ϖ and ε, that are satisfactory. `submodulariyRatioEstimation`, cf. Algorithm 2, iterates through randomly sampled parameters, while maintaining the maximum submodularity ratio that does not violate the sampled constraints. Once the agent has completed N_{SCP} number of sample iterations, we extrapolate the history of evolution of the submodularity ratio to get $\hat{\lambda}$, the approximate submodularity ratio. We do this by using a simple linear regression in lines 12-13 and evaluate the expression at $n = |\Delta|$, the cardinality of the constraint parameter set, to determine an estimate for the robust convex optimization problem.

The following result justifies to what extent the obtained ratio is a good approximation of the actual submodularity ratio.

Lemma 6.2 *Let $\hat{\lambda}$ be the approximate submodularity ratio returned by `submodulariyRatioEstimation` with inputs ϖ and ε. With probability, $1 - \varpi$, up to ε-fraction of constraints will be violated with respect to the robust convex optimization problem* (B.1).

Proof First, we show `submodulariyRatioEstimation` can be formulated as a scenario convex program and that it satisfies the convex constraint condition in Theorem B.1. Lines 6-11 provide a solution, λ^+, to the following scenario convex program.

$$\begin{aligned}\lambda^+ = \underset{\lambda^- \in \mathbb{R}}{\operatorname{argmin}} \quad & -\lambda^- \\ \text{s.t.} \quad & f_{\delta^i}(\lambda^-) \le 0, \quad i = 1, \ldots, N_{\mathrm{SCP}},\end{aligned}$$

where line 8 is a convex function that comes from Eq. (6.2). Since f_{δ^i} is a convex function, we can apply Theorem B.1; with probability, $1 - \varpi$, λ^+ violates at most ε-fraction of constraints in Δ.

The simple linear regression portion of the algorithm, lines 12-13, uses data points λ^δ, n that are only included in h when $\lambda^\delta < \lambda^+$. Therefore, the slope of the linear regression b is strictly negative. On line 13, $\hat{\lambda}$ is evaluated at $n = N_{\mathrm{SCP}}$ which implies that $\hat{\lambda} \le \lambda^+$ and that with probability, $1 - \varpi$, $\hat{\lambda}$ violates at most ε-fraction of constraints in Δ. □

Note that ϖ and ε can be chosen as small as desired to ensure that $\hat{\lambda}$ is a good approximation of λ. As $\hat{\lambda}$ approaches λ, our approximation of the lower bound performance of the algorithm with respect to f_x becomes more accurate. We conclude this section by studying whether the submodularity ratio is strictly positive. First, we prove it is always positive in non-degenerate cases.

Theorem 6.3 *Under Assumptions 6.1–6.3, f_x is a weakly submodular function.*

Proof We need to establish that the submodularity ratio of f_x is positive. We reason by contradiction, i.e., assume that the submodularity ratio is 0. This means that there exist X_x and Z_x such that the righthand side of expression (6.2) is zero (this rules out, in particular, the possibility of either X_x or Z_x being empty). In particular, this implies that $f_x(X_x \cup \{z\}) - f_x(X_x) = 0$ for every $z \in Z_x$ and that $f_x(X_x \cup Z_x) - f_x(X_x) > 0$. Assume that $\texttt{InitSubEnv}(X_x \cup \{z\})$ yields an ordered region list $[x_1, x_2, \ldots, z]$ for each z. Let r_x and t_x denote the reward and time for completing a task in a region x conditioned by the generated sub-environment $\texttt{InitSubEnv}(X_x \cup \{z\})$. Then,

$$\begin{aligned}f_x(X_x) &= \gamma^{t_{x_1}} r_{x_1} + \gamma^{t_{x_2}} r_{x_2} + \ldots, \\ f_x(X_x \cup \{z\}) &= \gamma^{t_{x_1}} r_{x_1} + \gamma^{t_{x_2}} r_{x_2} + \ldots + \gamma^{t_z} r_z, \\ f_x(X_x \cup \{z\}) - f_x(X_x) &= \gamma^{t_z} r_z,\end{aligned}$$

for each $z \in Z_x$ conditioned by the generated sub-environment $\texttt{InitSubEnv}(X_x \cup \{z\})$. Under Assumption 6.3, the term $f_x(X_x \cup \{z\}) - f_x(X_x)$ equals 0 when t_z is infinite. On the other hand, let r'_x and t'_x denote the reward and time for completing a task in region x conditioned by the generated sub-environment $\texttt{InitSubEnv}(X_x \cup Z_z)$. The denominator is nonzero when t'_z is finite. This cannot hold when t_z is infinite for each $z \in Z_x$ without contradicting Assumption 6.3, concluding the proof. □

The next remark discusses the challenge of determining an explicit lower bound on the submodularity ratio.

Remark 6.1 Beyond the result in Theorem 6.3, it is of interest to determine an explicit positive lower bound on the submodularity ratio. In general, obtaining such a bound for arbitrary scenarios is challenging and likely would yield overly conservative results. To counter this, we believe that restricting the attention to specific families of scenarios may instead lead to informative bounds. Our simulation results in Sect. 7.1 later suggest, for instance, that the submodularity ratio is approximately 1 in common scenarios related to scheduling spatially distributed tasks. However, formally establishing this fact remains an open problem. □

6.3 Sequential multi-agent deployment

We explore the performance of `DDRP-OO` in an environment with multiple agents. We consider the following assumption for the rest of this section.

Assumption 6.4 If an agent chooses a task that was already selected in X_ϑ, none of the completion time probability distributions are modified. Furthermore, the expected discounted reward for completing task τ given the set of trajectories X_ϑ is

$$T(\tau, X_\vartheta) = \max_{\vartheta \in X_\vartheta} \vartheta.\overrightarrow{\Pr^t}_k \gamma^t r^\phi = \max_{\vartheta \in X_\vartheta} T(\tau, \{\vartheta\}), \tag{6.3}$$

given that $\vartheta.\overrightarrow{\tau}_k = \tau$.

This assumption is utilized in the following *sequential multi-agent deployment* algorithm.

Algorithm 3: : Sequential multi-agent deployment

```
1  $\Omega_x = \bigcup \{x\}\ \forall x$
2  $\mathcal{E} \leftarrow$ current environment
3  $\hat{Q}, V_x, N_{\epsilon.s}, N_b \leftarrow$ loaded from previous trials

4  Evaluate ($\vartheta, X_\vartheta$):
5      $val = 0$
6      for $\overrightarrow{\tau}_k$ in $\vartheta.\overrightarrow{\tau}$:
7          if $T(\overrightarrow{\tau}_k, X_\vartheta) < \vartheta.\overrightarrow{Pr^t}_k(t)\gamma^t r^\phi$:
8              $val = val + \vartheta.\overrightarrow{Pr^t}_k(t)\gamma^t r^\phi - T(\overrightarrow{\tau}_k, X_\vartheta)$
9      return $val$

10 SequentialMultiAgentDeployment ($\mathcal{E}, \mathcal{A}, \Omega_x$)
11     $\Theta_\mathcal{A} = \emptyset$
12     $X_\vartheta = \emptyset$
13     for $\beta \in \mathcal{A}$
14         $\Theta_\mathcal{A} = \Theta_\mathcal{A} \cup \{\Theta^\beta\}$
15         $Z_\vartheta =$ DDRMAP ($\Omega_x, N_\epsilon, \mathcal{E}, \Theta^\mathcal{A}, \hat{Q}, V_x, N_{\epsilon.s}, N_b, \mathcal{M}$)
16         $\vartheta_\alpha = \underset{\vartheta \in Z_\vartheta}{\text{argmax}}$ Evaluate($\vartheta, X_\vartheta$) $X_\vartheta = X_\vartheta \cup \{\vartheta_\alpha\}$
17     return $X_\vartheta$
```

In this algorithm, agents plan their sub-environments and task search to determine a task trajectory ϑ one at a time. The function Evaluate returns the *marginal gain* of including ϑ, which is the added benefit of including ϑ in X_ϑ. We define the set function $f_\vartheta : 2^{\Omega_\vartheta} \to \mathbb{R}$ to be a metric for measuring the performance of SequentialMultiAgentDeployment as follows:

$$f_\vartheta(X_\vartheta) = \sum_{\forall \tau} T(\tau, X_\vartheta).$$

This function is interpreted as the sum of discounted rewards given all a set of trajectories X_ϑ. The definition of T from Assumption 6.4 allows us to state the following result.

Lemma 6.3 *Under Assumptions 6.3–6.4, f_ϑ is a submodular, monotone set function.*

Proof With (6.3) and the fact that rewards are non-negative (Assumption 6.3), we have that the marginal gain is never negative, therefore the function is monotone. For the function to be submodular, we show that it satisfies the condition of diminishing returns, meaning that $f_\vartheta(X_\vartheta \cup \{\vartheta_\alpha\}) - f_\vartheta(X_\vartheta) \geq f_\vartheta(Y_\vartheta \cup \{\vartheta_\alpha\}) - f_\vartheta(Y_\vartheta)$ for any $X_\vartheta \subseteq Y_\vartheta \subseteq \Omega_\vartheta$ and $\vartheta_\alpha \in \Omega_\vartheta \setminus Y_\vartheta$. Let

$$\mathcal{G}(X_\vartheta, \vartheta_\alpha) = T(\tau, X_\vartheta \cup \{\vartheta_\alpha\}) - T(\tau, X_\vartheta) = \max_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\}) - \max_{\vartheta \in X_\vartheta} T(\tau, \{\vartheta\})$$

be the marginal gain of including ϑ_α in X_ϑ. The maximum marginal gain of occurs when no $\vartheta \in X_\vartheta$ share the same tasks as ϑ_α. We determine the marginal gains $\mathcal{G}(X_\vartheta, \vartheta_\alpha)$ and $\mathcal{G}(Y_\vartheta, \vartheta_\alpha)$ for every possible case and show that $\mathcal{G}(X_\vartheta, \vartheta_\alpha) \geq \mathcal{G}(Y_\vartheta, \vartheta_\alpha)$.

Case 1: $\vartheta_\alpha = \text{argmax}_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\}) = \text{argmax}_{\vartheta \in Y_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\})$. This implies that $T(\tau, X_\vartheta \cup \{\vartheta_\alpha\}) = T(\tau, Y_\vartheta \cup \{\vartheta_\alpha\})$.

Case 2: $\vartheta = \text{argmax}_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\}) = \text{argmax}_{\vartheta \in Y_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\})$ such that $\vartheta \in X_\vartheta$. This implies that $T(\tau, X_\vartheta \cup \{\vartheta_\alpha\}) = T(\tau, Y_\vartheta \cup \{\vartheta_\alpha\})$.

Case 3: $\vartheta_\alpha = \text{argmax}_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\})$, and $\vartheta = \text{argmax}_{\vartheta \in Y_\vartheta \cup \{\vartheta_\alpha\}} T(\tau, \{\vartheta\})$ such that $\vartheta \in Y_\vartheta \setminus X_\vartheta$. Thus

$$\mathcal{G}(X_\vartheta, \vartheta_\alpha) \geq 0 \quad \text{and} \quad \mathcal{G}(Y_\vartheta, \vartheta_\alpha) = 0.$$

For both cases 1 and 2, since the function is monotone, we have $T(\tau, Y_\vartheta) \geq T(\tau, X_\vartheta)$. Therefore, the marginal gain $\mathcal{G}(X_\vartheta, \vartheta_\alpha) \geq \mathcal{G}(Y_\vartheta, \vartheta_\alpha)$ for all cases. □

Having established that f_ϑ is a submodular and monotone function, our next step is to provide conditions that allow us to cast SequentialMultiAgentDeployment as a greedy algorithm with respect to this function. This would enable us to employ Theorem A.1 to guarantee lower bounds on $f_\vartheta(X_\vartheta)$, with X_ϑ being the output of SequentialMultiAgentDeployment.

First, when picking trajectories from line 16 in SequentialMultiAgentDeployment, all trajectories must be chosen from the same set of possible trajectories Ω_ϑ. We satisfy this requirement with the following assumption.

Assumption 6.5 Agents begin at the same location in the environment, share the same SMDP, and are capable of interacting with as many other agents as needed, i.e., $\mathcal{N} = |\mathcal{A}|$. Agents choose their sub-environment trajectories one at a time. Furthermore, agents are given sufficient time for DDRP (line 15) to have visited all possible trajectory sets Ω_ϑ.

The next assumption we make allows agents to pick trajectories regardless of order and repetition, which is needed to mimic the set function properties of f_ϑ.

Assumption 6.6 $\mathfrak{R}$ is set to the single agent reward r^ϵ. As a result, the multi-agent interaction reward function is $r^\phi = r^\epsilon$.

This assumption is necessary because, if we instead consider a reward $\mathfrak{R}$ that is dependent on the number of agents acting on τ, the order of which the agents choose their trajectories would affect their decision making. Furthermore, this restriction satisfies the condition $Var(\mathfrak{R})$ to be finite in Theorem 6.1. We are now ready to characterize the lower bound performance of `SequentialMultiAgentDeployment` with respect to the optimal set of task trajectories. For convenience, let $X_\vartheta^* = \operatorname*{argmax}_{X_\vartheta \in \Omega_\vartheta} f_\vartheta(X_\vartheta)$ denote the optimal set of task trajectories.

Theorem 6.4 *Let X_ϑ be the trajectory set returned by* `SequentialMultiAgentDeployment`*. Under Assumptions 6.2–6.6, it holds that $f_\vartheta(X_\vartheta) \geq (1 - e^{-1}) f_\vartheta(X_\vartheta^*)$.*

Proof Our strategy relies on making sure we can invoke Theorem A.1 for `SequentialMultiAgentDeployment`. From Lemma 6.3, we know f_ϑ is submodular and monotone. What is left is to show that `SequentialMultiAgentDeployment` chooses trajectories from Ω_ϑ which maximize the marginal gain with respect to f_ϑ. First, we have that the agents all choose from the set Ω_ϑ as a direct result of Assumptions 6.2 and 6.5. This is because $\hat{Q}$ and SMDP are equivalent for all agents and they all begin with the same initial conditions. Now we show that `SequentialMultiAgentDeployment` chooses agents which locally maximizes the marginal gain of f_ϑ. Given any set X_ϑ and $\vartheta_\alpha \in \Omega_\vartheta \setminus X_\vartheta$, the marginal gain is

$$\begin{aligned} & f_\vartheta(X_\vartheta \cup \{\vartheta_\alpha\}) - f_\vartheta(X_\vartheta) \\ & \quad = \sum_{\forall \tau} T(\tau, X_\vartheta \cup \{\vartheta_\alpha\}) - \sum_{\forall \tau} T(\tau, X_\vartheta) \\ & \quad = \sum_{\forall \tau} (T(\tau, X_\vartheta \cup \{\vartheta_\alpha\}) - T(\tau, X_\vartheta)). \end{aligned}$$

The function `Evaluate` on lines 7 and 8 has the agent calculate the marginal gain for a particular task, given ϑ and X_ϑ. `Evaluate` calculates the marginal gain for all tasks as

$$\sum_{\forall \tau \in \vartheta.\overrightarrow{\tau}} \max((T(\tau, \{\vartheta\}), T(\tau, X_\vartheta)) - T(\tau, X_\vartheta)),$$

which is equivalent to $\sum_{\forall \tau} (T(X_\vartheta \cup \{\vartheta\}) - T(X_\vartheta))$. Since `SequentialMultiAgentDeployment` takes the trajectory that maximizes `Evaluate`, the result now follows from Theorem A.1. □

7 Empirical validation and evaluation of performance

In this section we perform simulations in order to validate our theoretical analysis and justify the use of `DDRP` over other model-based and model-free methods. All simulations were performed on a Linux-based workstation with 16 GB of RAM and a stock AMD Ryzen 1600 CPU. GPU was not utilized in our studies, but could be implemented to improve sampling speed of some testing algorithms. We first illustrate the optimality ratio obtained by the sub-environment search in single-agent and sequential multi-agent deployment scenarios. Next, we compare the performance of `DDRP`, `DDRP-OO`, MCTS, and ACKTR in a simple environment. Lastly, we study the effect of multi-agent interaction on the performance of the proposed algorithm.

7.1 Illustration of performance guarantees

Here we perform simulations that help validate the results of Sect. 6. We achieve this by determining X_x from `SubEnvSearch`, which is an implementation of greedy maximization of submodular set functions, and the optimal region set X_x^*, by brute force computation of all possible sets. In this simulation, we use 1 agent and 1 objective with 25 regions. We implement `DDRP-OO` by loading previously trained data and compare the value of the first sub-environment found to the value of the optimal sub-environment. 1000 trials are simulated by randomizing the initial state of the environments. We plot the probability distribution function of $f_x(X_x)/f_x(X_x^*)$ in Fig. 2. The empirical lower bound of $f_x(X_x)/f_x(X_x^*)$ is a little less than $1 - e^{-1}$, consistent with the result, cf. Theorem 6.2, that the submodularity ratio of `SubEnvSearch` may not be 1. We believe that

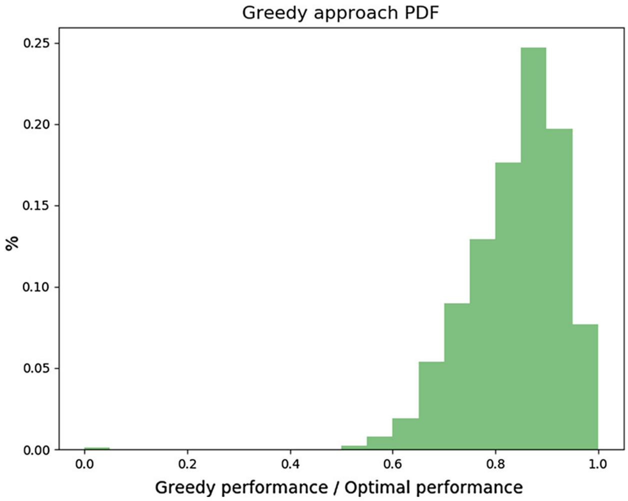

Fig. 2 Probability distribution function of $f_x(X_x)/f_x(X_x^*)$

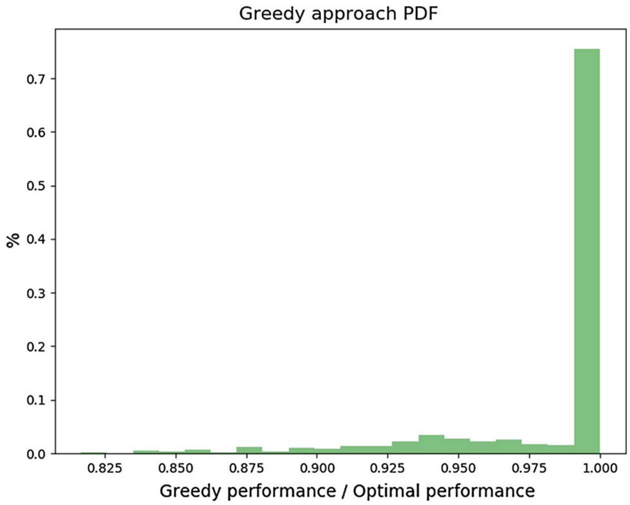

Fig. 3 Probability distribution function of $f_\vartheta(X_\vartheta)/f_\vartheta(X^*_\vartheta)$

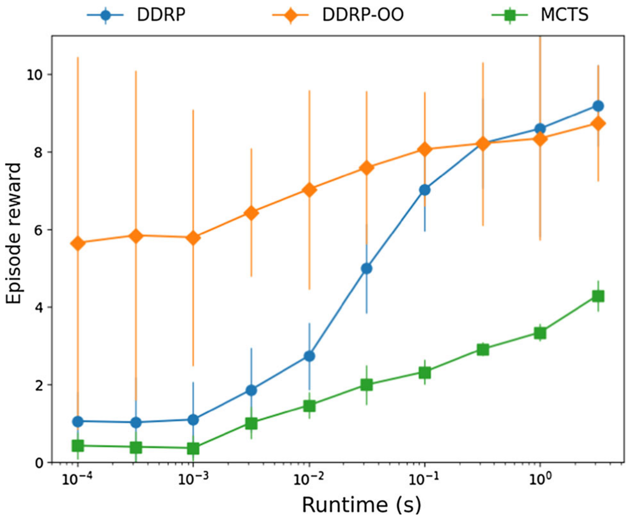

Fig. 4 Performance of DDRP, DDRP-OO, and MCTS in randomized 2D environments with one objective type

another factor for this empirical lower bound is that Assumption 6.2 is not fully satisfied in our experiments. In order to perform the simulation, we trained the agent on similar environments for 10 min. Because the number of possible states in this simulation is very large, some of the uncommon states may not have been visited enough for $\hat{Q}$ to mature.

Next we look for empirical validation for the lower bounds on $f_\vartheta(X_\vartheta)/f_\vartheta(X^*_\vartheta)$. This is a difficult task because determining the optimal set of trajectories X^*_ϑ is combinatorial with respect to the trajectory size, number regions, and number of agents. We simulate `SequentialMultiAgentDeployment` in a 36 region environment with one type of objective where 3 agents are required to start at the same region and share the same $\hat{Q}$, which is assumed to have converged to the optimal value. 1000 trials are simulated by randomizing the initial state of the environments. As shown in Fig. 3, the lower bound on the performance with respect to the optimal set of trajectories is greater than $1 - e^{-1}$, as guaranteed by Theorem 6.4. This empirical lower bound may change under more complex environments with an increased number of agents, more regions, and longer sub-environment lengths. Due to the combinatorial nature of determining the optimal set of trajectories, it is difficult to simulate environments of higher complexity.

7.2 Comparisons to alternative algorithms

In DDRP, DDRP-OO, and MCTS, the agent is given an allocated time to search for the best trajectory. In ACKTR, we look at the number of simulations needed to converge to a policy comparable to the ones found in DDRP, DDRP-OO, and MCTS. We simulate the same environment across these algorithms. The environment has $|\mathcal{O}| = 1$, where objectives have a random number of waypoints ($|\mathcal{O}^b| \leq 15$) placed uniformly randomly. The environment contains 100 × 100 points in $\mathbb{R}^2$, with 100 evenly distributed regions. The sub-environment length for DDRP and DDRP-OO are both 10 and the maximum number of steps that an agent is allowed to take is 100. Furthermore, the maximum reward per episode is capped at 10. We choose this environment because of the large state space and action space in order to illustrate the strength of Dynamic Domain Reduction for Multi-Agent Planning in breaking it down into components that have been previously seen. Figure 4 shows that MCTS performs poorly for the chosen environment because of the large state space and branching factor. DDRP initially performs poorly, but yields strong results given enough time to think. DDRP-OO performs well even when not given much time to think. Theorem 6.2 helps give intuition to the immediate performance of DDRP-OO. The ACKTR simulation, displayed in Fig. 5, performs well, but only after several million episodes of training, corresponding to approximately 2 h using 10 CPU cores. This illustrates the inherent advantage of model-based reinforcement learning approaches when the MDP model is available. Data is plotted to show the average and confidence intervals of the expected discounted reward of the agent(s) found in the allotted time. We perform 100 trials per data point in the case studies.

We perform another study to visually compare trajectories generated from DDRP, MCTS, and ACKTR as shown in Fig. 6. The environment contains three objectives with waypoints denoted by ‘x’, ‘square’, and ‘triangle’ markers. Visiting ‘x’ waypoints yield a reward of 3, while visiting ‘square’ or ‘triangle’ waypoints yield a reward of 1. We ran both MCTS and DDRP for 3.16 s and ACKTR for 10,000 trials, all with a discount factor of $\gamma = .99$. The best trajectories found by MCTS, DDRP, and ACKTR are shown in Fig. 6

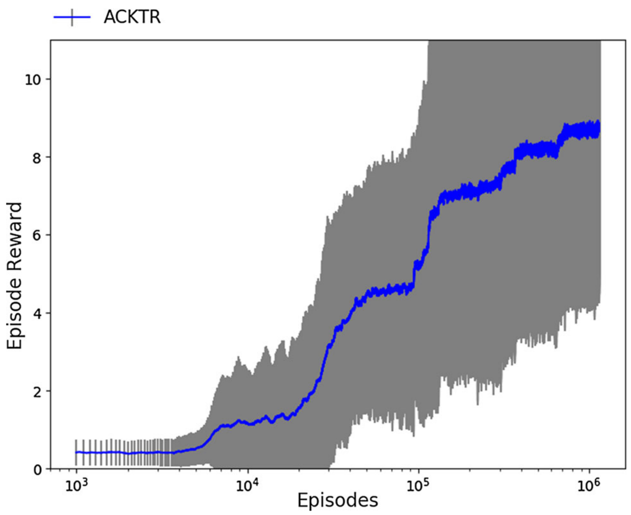

Fig. 5 Performance of offline algorithm: ACKTR in randomized 2D environments with one objective type

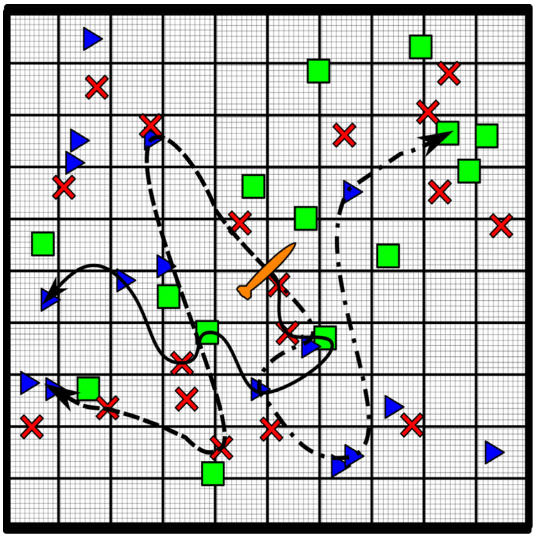

Fig. 6 The trajectories generated from MCTS, DDRP, and ACKTR are shown with dashed, solid and dash-dot lines respectively, in a 100 × 100 environment with 3 objectives. The squares, x's, and triangles represent waypoints of three objective types. The agent (represented by a torpedo) starts at the center of the environment

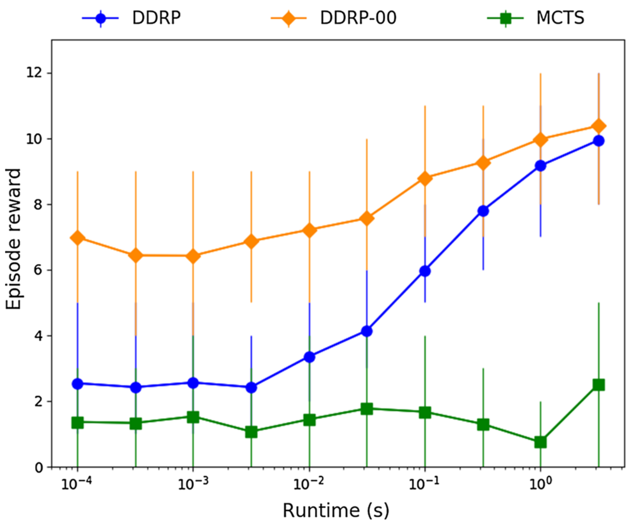

Fig. 7 Performance of DDRP, DDRP-OO, and MCTS in randomized 2D environments with three objective types

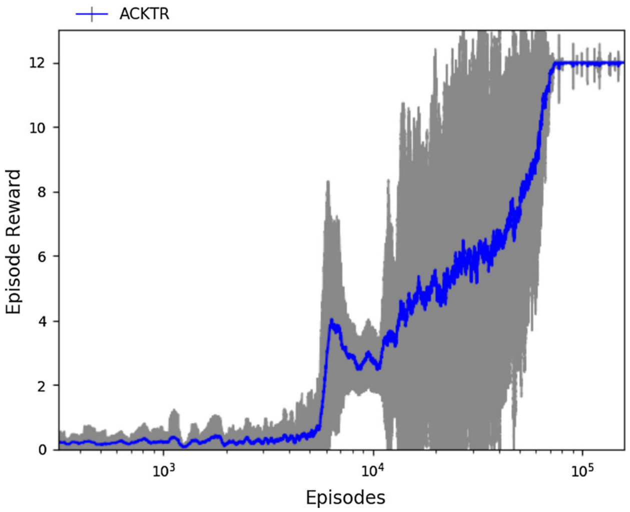

Fig. 8 Performance of ACKTR in a static 2D environment with three objective types

which yield discounted rewards of 1.266, 5.827 and 4.58, respectively. It is likely that the ACKTR policy converged to a local maximum because the trajectories generated near the ending of the 100,000 trials had little deviation. We use randomized instances of this environment to show a comparison of DDRP, DDRP-OO, and MCTS with respect to runtime in Fig. 7 and show the performance of ACKTR in a static instance of this environment in Fig. 8.

7.3 Effect of multi-agent interaction

Our next simulation evaluates the effect of multi-agent cooperation in the algorithm performance. We consider an environment similar to the one in Sect. 7.2, except with 10 agents and $|\mathcal{O}| = 3$, where objectives have a random number of waypoints ($|\mathcal{O}^b| \leq 5$) that are placed randomly. In this simulation we do not train the agents before trials, and *Dynamic Domain Reduction for Multi-Agent Planning* is used with varying $\mathcal{N}$ where agents asynchronously choose trajectories. In Fig. 9, we can see the benefit of allowing agents to interact with each other. When agents are able to take coupled actions, the expected potential discounted reward is greater, a feature that becomes more marked as agents are given more time T to think.

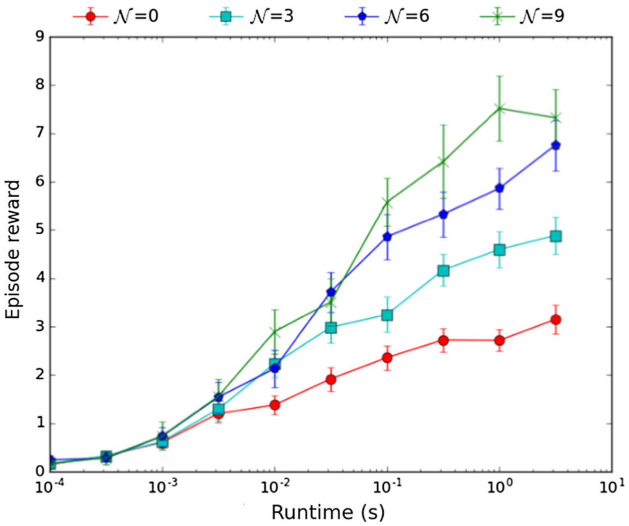

Fig. 9 Performance of multi-agent deployment using `DDRMAP` in 2D environment

8 Conclusions

We have presented a framework for high-level multi-agent planning leading to the *Dynamic Domain Reduction for Multi-Agent Planning* algorithm. Our design builds on a hierarchical approach that simultaneously searches for and creates sequences of actions and sub-environments with the greatest expected reward, helping alleviate the curse of dimensionality. Our algorithm allows for multi-agent interaction by including other agents' state in the sub-environment search. We have shown that the action value estimation procedure in `DDRP` converges to the optimal value of the sub-environment SMDP with probability 1. We also identified metrics to quantify performance of the sub-environment selection in `SubEnvSearch` and sequential multi-agent deployment in `SequentialMultiAgentDeployment`, and provided formal guarantees using scenario optimization and submodularity. We have illustrated our results and compared the algorithm performance against other approaches in simulation. The biggest limitation of our approach is related to the spatial distribution of objectives. The algorithm does not perform well if the environment is set up such that objectives cannot be split well into regions. Future work will explore the incorporation of constraints on battery life and connectivity maintenance of the team of agents, the consideration of partial agent observability and limited communication, and the refinement of multi-agent cooperation capabilities enabled by prediction elements that indicate whether other agents will aid in the completion of an objective. We also plan to explore the characterization, in specific families of scenarios, of positive lower bounds on the submodularity ratio of the set function that assigns the discounted reward of the selected sub-environment, and the use of parallel, distributed methods for submodular optimization capable of handling asynchronous communication and computation.

Acknowledgements This work was supported by ONR Award N00014-16-1-2836. The authors would like to thank the organizers of the International Symposium on Multi-Robot and Multi-Agent Systems (MRS 2017), which provided us with the opportunity to obtain valuable feedback on this research, and the reviewers.

A Submodularity

We review here concepts of submodularity and monotonicity of set functions following Clark et al. (2016). A power set function $f : 2^{\Omega} \to \mathbb{R}$ is *submodular* if it satisfies the property of diminishing returns,

$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y), \tag{A.1}$$

for all $X \subseteq Y \subseteq \Omega$ and $x \in \Omega \setminus Y$. The set function f is *monotone* if

$$f(X) \leq f(Y), \tag{A.2}$$

for all $X \subseteq Y \subseteq \Omega$. In general, monotonicity of a set function does not imply submodularity, and vice versa. These properties play a key role in determining near-optimal solutions to the *cardinality-constrained submodular maximization* problem defined by

$$\begin{aligned} &\max \quad f(X) \\ &\text{s.t. } |X| \leq k. \end{aligned} \tag{A.3}$$

In general, this problem is NP-hard. Greedy algorithms seek to find a suboptimal solution to (A.3) by building a set X one element at a time, starting with $|X| = 0$ to $|X| = k$. These algorithms proceed by choosing the best next element,

$$\max_{x \in \Omega \setminus X} f(X \cup \{x\}),$$

to include in X. The following result (Clark et al. 2016; Nemhauser et al. 1978) provides a lower bound on the performance of greedy algorithms.

Theorem A.1 *Let X^* denote the optimal solution of problem* (A.3)*. If f is monotone, submodular, and satisfies $f(\emptyset) = 0$, then the set X returned by the greedy algorithm satisfies*

$$f(X) \geq (1 - e^{-1}) f(X^*).$$

An important extension of this result characterizes the performance of a greedy algorithm where, at each step, one chooses an element x that satisfies

$$f(X \cup \{x\}) - f(X) \geq \alpha(f(X \cup \{x^*\}) - f(X)),$$

for some $\alpha \in [0, 1]$. That is, the algorithm chooses an element that is at least an α-fraction of the local optimal element choice, x^*. In this case, the following result (Goundan and Schulz 2007) characterizes the performance.

Theorem A.2 *Let X^* denote the optimal solution of problem (A.3). If f is monotone, submodular, and satisfies $f(\emptyset) = 0$, then the set X returned by a greedy algorithm that chooses elements of at least α-fraction of the local optimal element choice satisfies*

$$f(X) \geq (1 - e^{-\alpha}) f(X^*).$$

A generalization of the notion of submodular set function is given by the *submodularity ratio* (Das and Kempe 2011), which measures how far the function is from being submodular. This ratio is defined as largest scalar $\lambda \in [0, 1]$ such that

$$\lambda \leq \frac{\sum_{z \in Z} f(X \cup \{z\}) - f(X)}{f(X \cup Z) - f(X)}, \tag{A.4}$$

for all $X, Z \subset \Omega$. The function f is called weakly submodular if it has a submodularity ratio in $(0, 1]$. If a function f is submodular, then its submodularity ratio is 1. The following result (Das and Kempe 2011) generalizes Theorem A.1 to monotone set functions with submodularity ratio λ.

Theorem A.3 *Let X^* denote the optimal solution of problem (A.3). If f is monotone, weakly submodular with submodularity ratio $\lambda \in (0, 1]$, and satisfies $f(\emptyset) = 0$, then the set X returned by the greedy algorithm satisfies*

$$f(X) \geq (1 - e^{-\lambda}) f(X^*).$$

B Scenario optimization

Scenario optimization aims to determine robust solutions for practical problems with unknown parameters (Ben-Tal and Nemirovski 1998; Ghaoui et al. 1998) by hedging against uncertainty. Consider the following *robust convex optimization problem* defined by

$$\begin{aligned} \text{RCP:} \ & \min_{\gamma \in \mathbb{R}^d} c^T \gamma \\ & \text{subject to: } f_\delta(\gamma) \leq 0, \ \forall \delta \in \Delta, \end{aligned} \tag{B.1}$$

where f_δ is a convex function, d is the dimension of the optimization variable, δ is an uncertain parameter, and Δ is the set of all possible parameter values. In practice, solving the optimization (B.1) can be difficult depending on the cardinality of Δ. One approach to this problem is to solve (B.1) with sampled constraint parameters from Δ. This approach views the uncertainty of situations in the robust convex optimization problem through a probability distribution Pr^δ of Δ, which encodes either the likelihood or importance of situations occurring through the constraint parameters. To alleviate the computational load, one selects a finite number N_{SCP} of parameter values in Δ sampled according to Pr^δ and solves the *scenario convex program* (Campi et al. 2009) defined by

$$\begin{aligned} \text{SCP}_N : & \min_{\gamma \in \mathbb{R}^d} c^T \gamma \\ & \text{s.t. } f_{\delta^{(i)}}(\gamma) \leq 0, \ i = 1, \ldots, N_{\text{SCP}}. \end{aligned} \tag{B.2}$$

The following result states to what extent the solution of (B.2) solves the original robust optimization problem.

Theorem B.1 *Let γ^* be the optimal solution to the scenario convex program (B.2) when N_{SCP} is the number of convex constraints. Given a 'violation parameter', ε, and a 'confidence parameter', ϖ, if*

$$N_{SCP} \geq \frac{2}{\varepsilon} \left(ln \frac{1}{\varpi} + d \right)$$

then, with probability $1 - \varpi$, γ^* satisfies all but an ε-fraction of constraints in Δ.

C List of symbols

$(\mathbb{Z}_{\geq 1})\mathbb{Z}$	(Non-negative) integers
$(\mathbb{R}_{>0})\mathbb{R}$	(Positive) real numbers
$\lvert Y \rvert$	Cardinality of set Y
$s \in S$	State/state space
$a \in A$	Action/action space
Pr^s	Transition function
r, R	Reward, reward function
$(\pi^*)\pi$	(Optimal) policy
V^π	Value of a state given policy, π
γ	Discount factor
α, β	Agent indices
$\mathcal{A}$	Set of agents
$o \in \mathcal{O}^b \in \mathcal{O}$	Waypoint/objective/objective set
$\mathcal{E}$	Environment
$x \in \Omega_x$	Region/region set
$\mathcal{O}^b_i$	Set of waypoints of an objective in a region
s^b_i	Abstracted objective state
s_i	Regional state
τ	Task
Γ	Set of feasible tasks
$\vec{x}$	Ordered list of regions

$\xi_{\overrightarrow{x}}$	Repeated region list
$\phi_t(,)$	Time abstraction function
$(\epsilon_k)\epsilon$	(Partial) sub-environment
N_ϵ	Size of a sub-environment
$\overrightarrow{\tau}$	Ordered list of tasks
$\overrightarrow{\Pr}^t$	Ordered list of probability distributions
$(\vartheta_\beta^p)\vartheta$	(Partial) task trajectory
$\mathcal{I}_\alpha$	Interaction set of agent α
$\mathcal{N}$	Max size of interaction set
θ	Claimed regional objective
Θ_α	Claimed objective set of agent α
$\Theta^{\mathcal{A}}$	Global claimed objective set
Ξ	Interaction matrix
Q	Value for choosing τ given $\epsilon.s$
$\hat{Q}$	Estimated value for choosing τ given $\epsilon.s$
N	Number of simulations in $\epsilon.s$
$N_{\mathcal{O}^b}$	Number of simulations of an objective in $\epsilon.s$
t	Time
T	Multi-agent expected discounted reward per task
λ	Submodularity ratio
$\hat{\lambda}$	Approximate submodularity ratio
f_x	Sub-environment search value set function
$X_x \subset Y_x \subset \Omega_x$	Finite set of regions
f_ϑ	Sequential multi-agent deployment value set function
$X_\vartheta \subset Y_\vartheta \subset \Omega_\vartheta$	Finite set of trajectories
ϖ	Confidence parameter
ε	Violation parameter

References

Agha-mohammadi, A. A., Chakravorty, S., & Amato, N. M. (2011). FIRM: Feedback controller-based information-state roadmap-a framework for motion planning under uncertainty. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 4284–4291). San Francisco, CA.

Bai, A., Srivastava, S., & Russell, S. (2016). Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence* (pp. 3029–3039). New York, NY: IJCAI.

Bellman, R. (1966). Dynamic programming. *Science*, *153*(3731), 34–37.

Ben-Tal, A., & Nemirovski, A. (1998). Robust convex optimization. *Mathematics of Operations Research*, *23*, 769–805.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont: Athena Scientific.

Bian, A. A., Buhmann, J. M., Krause, A., & Tschiatschek, S. (2017). Guarantees for greedy maximization of non-submodular functions with applications. In *International conference on machine learning* Vol. 70 (pp. 498–507). Sydney.

Blum, A., Chawla, S., Karger, D. R., Lane, T., Meyerson, A., & Minkoff, M. (2007). Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, *37*(2), 653–670.

Broz, F., Nourbakhsh, I., & Simmons, R. (2008). Planning for human–robot interaction using time-state aggregated POMDPs. *AAAI*, *8*, 1339–1344.

Bullo, F., Cortés, J., & Martínez, S. (2009). *Distributed control of robotic networks. Applied mathematics series*. Princeton, NJ: Princeton University Press.

Campi, M. C., Garatti, S., & Prandini, M. (2009). The scenario approach for systems and control design. *Annual Reviews in Control*, *32*(2), 149–157.

Clark, A., Alomair, B., Bushnell, L., & Poovendran, R. (2016). *Submodularity in dynamics and control of networked systems*., Communications and control engineering New York: Springer.

Cortés, J., & Egerstedt, M. (2017). Coordinated control of multi-robot systems: A survey. *SICE Journal of Control, Measurement, and System Integration*, *10*(6), 495–503.

Das, A., & Kempe, D. (February 2011). Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In: *CoRR*.

Das, J., Py, F., Harvey, J. B. J., Ryan, J. P., Gellene, A., Graham, R., et al. (2015). Data-driven robotic sampling for marine ecosystem monitoring. *The International Journal of Robotics Research*, *34*(12), 1435–1452.

Dunbabin, M., & Marques, L. (2012). Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics and Automation Magazine*, *19*(1), 24–39.

Gerkey, B. P., & Mataric, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, *23*(9), 939–954.

Ghaoui, L. E., Oustry, F., & Lebret, H. (1998). Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, *9*(1), 33–52.

Goundan, P. R., & Schulz, A. S. (2007). Revisiting the greedy approach to submodular set function maximization. *Optimization Online* (pp. 1–25).

Hansen, E. A., & Feng, Z. (2000). Dynamic programming for POMDPs using a factored state representation. In *International conference on artificial intelligence planning systems* (pp. 130–139). Breckenridge, CO.

Howard, R. (1960). *Dynamic programming and Markov processes*. Cambridge: M.I.T. Press.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *ECML* Vol. 6 (pp. 282–293). Springer.

LaValle, S. M., & Kuffner, J. J. (2000). Rapidly-exploring random trees: Progress and prospects. In *Workshop on algorithmic foundations of robotics* (pp. 293–308). Dartmouth, NH.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, *28*(1), 47–65.

Ma, A., Ouimet, M., & Cortés, J. (2017). Dynamic domain reduction for multi-agent planning. In *International symposium on multi-robot and multi-agent systems* (pp. 142–149). Los Angeles, CA.

McCallum, A. K., & Ballard, D. (1996). *Reinforcement learning with selective perception and hidden state*. Ph.D. Dissertation, University of Rochester. Department of Computer Science.

Mesbahi, M., & Egerstedt, M. (2010). *Graph theoretic methods in multiagent networks*., Applied mathematics series Princeton: Princeton University Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Nemhauser, G., Wolsey, L., & Fisher, M. (1978). An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, *14*, 265–294.

Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized POMDPs.*, SpringerBriefs in intelligent systems New York: Springer.

Omidshafiei, S., Agha-mohammadi, A. A., Amato, C., & How, J. P. (May. 2015). Decentralized control of partially observable Markov decision processes using belief space macro-actions. In *IEEE international conference on robotics and automation* (pp. 5962–5969). Seattle, WA.

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

Parr, R., & Russell, S. (1998). *Hierarchical control and learning for Markov decision processes*. Berkeley, CA: University of California.

Prentice, S., & Roy, N. (2010). The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Robotics research* (pp. 293–305). Springer.

Puterman, M. (2014). *Markov decision processes: Discrete stochastic dynamic programming*. Hoboken: Wiley.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897). Lille: France.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1–2), 181–211.

Theocharous, G., & Kaelbling, L. P. (2004). Approximate planning in POMDPs with macro-actions. In *Advances in neural information processing systems* (pp. 775–782).

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., & Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems* Vol. 30 (pp. 5285–5294).

Aaron Ma is a Ph.D. student at the University of California, San Diego. He received his bachelor's degree at the University of California, Santa Barbara studying Mechanical Engineering in 2012. In 2016, Aaron received his master's degree at the University of California, San Diego where he continued under the supervision of Dr. Jorge Cortes in order to study dynamic systems and controls. Aaron's current research interests include high-level planning for heterogenous robotic swarms, model-based and model-free reinforcement learning, and development/implementation of distributed algorithms on robotic platforms. Aaron is funded by the Office of Naval Research.

Michael Ouimet is a researcher at NIWC Pacific interested in autonomy for multi-agent unmanned systems, especially in the intersection of control theory, artificial intelligence, and machine learning for robotics. He received his Masters and Ph.D. in Mechanical and Aerospace Engineering from University of California, San Diego from 2009 to 2014 under the direction of Prof. Jorge Cortes and was a postdoctoral researcher there until 2015.

Jorge Cortés is a Professor of Mechanical and Aerospace Engineering at the University of California, San Diego, CA, USA. He received the Licenciatura degree in mathematics from Universidad de Zaragoza, Zaragoza, Spain, in 1997, and the Ph.D. degree in engineering mathematics from Universidad Carlos III de Madrid, Madrid, Spain, in 2001. He held postdoctoral positions with the University of Twente, Twente, The Netherlands, and the University of Illinois at Urbana-Champaign, Urbana, IL, USA. He was an Assistant Professor with the Department of Applied Mathematics and Statistics, University of California, Santa Cruz, CA, USA, from 2004 to 2007. He is the author of Geometric, Control and Numerical Aspects of Nonholonomic Systems (Springer-Verlag, 2002) and co-author (together with F. Bullo and S. Martinez) of Distributed Control of Robotic Networks (Princeton University Press, 2009). He was an IEEE Control Systems Society Distinguished Lecturer (2010–2014) and is an IEEE Fellow. His current research interests include cooperative control, network science, game theory, multi-agent coordination in robotics, power systems, and neuroscience, geometric and distributed optimization, nonsmooth analysis, and geometric mechanics and control.