UNIVERSITY OF CALIFORNIA SAN DIEGO

Model-based reinforcement learning for cooperative multi-agent planning: exploiting hierarchies, bias, and temporal sampling

A Dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Engineering Sciences (Mechanical Engineering)

by

Aaron Ma

Committee in charge:

Jorge Cortés, Chair
Nikolay Atanasov
Mauricio De Oliveira
Kenneth Kreutz-Delgado
Sonia Martinez
Mike Ouimet

2020

The dissertation of Aaron Ma is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____

Chair

University of California San Diego

2020

DEDICATION

To my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Professor Jorge Cortés. Professor Cortés made a profound effect on my life and career by first accepting me as a Masters student in 2014, where I was introduced to the world of research and robotics. Nearing my graduation, Prof. Cortés motivated me to pursue a PhD. I cannot thank Prof. Cortés enough for the work that he has put into developing and finding my potential as a researcher by meeting weekly and going over the fundamentals and nuances of research with patience and understanding. Prof. Cortés takes special care to make sure his pupils are well rounded in terms of presentation and knowledge of the material. Furthermore, he provided the opportunity for me to work and mentor undergraduates in the Multi-agent Robotics lab, where I found my passion and appreciation for the development of robots in both hardware and software. Thank you for everything.

Dr. Mike Ouimet, deserves a huge thanks for being a great mentor, role-model, and friend. I first met Mike as a Masters student in the Multi-agent Robotics lab at UCSD. He taught me how to interface with robotics hardware and the importance of programming using ROS and Python. Mike was a mentor to program and acted as a liason in order to bridge the gap between UCSD and SPAWAR. Mike played a large part in my decision to pursue a PhD. He often inspired me with his creativity, imagination, and interest in the field of reinforcement learning and played a critical role in finding a research grant from the Office of Naval Research. I cannot thank both Prof. Cortés and Dr. Ouimet for all of the hardwork they have put into our collaboration.

A special thanks goes to my commitee: Prof. Sonia Martínez, Prof. Nikolay

Atansov, Prof. Mauricio De Oliveira, and Prof. Kenneth Kreutz-Delgado for taking time to attend my Senate and Final Defense presentations. The feedback that we received is invaluable.

I want to thank SPAWAR for granting me internship over the summers of my PhD. I enjoyed all the time spent there, especially due to Mike Tall, who was in charge of many of the projects that we worked on. Mike Tall worked tirelessly to enable us interns to gather valuable experience in robotics. Special thanks to everyone I met at SPAWAR as well, since they made the experience a lot of fun.

I want to express the deepest gratitude towards my family because they pushed me with love and support to pursue a life that I am passionate about, ultimately leading me to where I am today.

Lastly, a big thanks to the Office of Naval research for awarding me the grant which ultimately enabled me to pursue a PhD. Especially Maria Madeiros and Chris Duarte, who are program officers at ONR and sponsored my participation. Maria and Chris provided an amazing experience with my experiences during my PhD including the annual Naval Undersea Research Program, where participants present their research material and its possible future relevance to the Navy.

Chapter 1 is coauthored with Cortés, Jorge. The dissertation author was the primary author of this chapter.

Chapter 2, is coauthored with Cortés, Jorge and Ouimet, Mike. The dissertation author was the primary author of this chapter.

Chapter 3, is coauthored with Cortés, Jorge and Ouimet, Mike in full, is a reprint of the material as it appears in Autonomous Robotics 44 (3-4) 2020, 485-503, Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary

author of this chapter.

Chapter 4, in full, is a reprint of the material as it appears in Distributed Autonomouos Robotic Systems: The 14th International Symposium, Springer Proceedings in Advanced Robotics, vol. 9, pp.499-512, Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary author of this chapter.

Chapter 5, in full is currently being prepared for submission for publication of the material. Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in full, is a reprint of the material as it appears in IEEE Robotics and Autonation Letters 5 (2) 2020, 1819-1826. Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary investigator and author of this paper.

Chapter 7 is coauthored with Cortés, Jorge. The dissertation author was the primary author of this chapter.

VITA

2012	Bachelor of Science in Mechanical Engineering, University of California Santa Barbara
2016	Master of Science in Engineering Sciences (Mechanical Engineering), University of California San Diego
2020	Doctor of Philosophy in Engineering Sciences (Mechanical Engineering), University of California San Diego

PUBLICATIONS

Journal publications:

- A. Ma, M. Ouimet, and J. Cortés. Hierarchical reinforcement learning via dynamic subspace search for multi-agent planning. *Autonomous Robots*, 2020. To appear
- A. Ma, M. Ouimet, and J. Cortés. Exploiting bias for cooperative planning in multi-agent tree search. *IEEE Robotics and Automation Letters*, 2020. To appear
- A. Ma and J. Cortés. Distributed multi-agent deployment for full visibility of 1.5D and 2.5D polyhedral terrains. *Journal of Intelligent and Robotic Systems*, 2019. Submitted

Conference proceedings:

- A. Ma and J. Cortés. Visibility-based distributed deployment of robotic teams in polyhedral terrains. In *ASME Dynamic Systems and Control Conference*, Minneapolis, MN, October 2016. DSCC2016-9820
- A. Ma, M. Ouimet, and J. Cortés. Dynamic domain reduction for multi-agent planning. In *International Symposium on Multi-Robot and Multi-Agent Systems*, pages 142–149, Los Angeles, CA, 2017
- A. Ma, M. Ouimet, and J. Cortés. Cooperative dynamic domain reduction. In *Distributed Autonomous Robotic Systems: The 14th International Symposium*, volume 9 of *Springer Proceedings in Advanced Robotics*, pages 499–512. Springer, New York, 2019

ABSTRACT OF THE DISSERTATION

Model-based reinforcement learning for cooperative multi-agent planning: exploiting hierarchies, bias, and temporal sampling

by

Aaron Ma

Doctor of Philosophy in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2020

Jorge Cortés, Chair

Autonomous unmanned vehicles (UxVs) can be useful in many scenarios including disaster relief, production and manufacturing, as well as carrying out Naval missions such as surveillance, mapping of unknown regions and pursuit of other hostile vehicles. When considering these scenarios, one of the most difficult challenges is determining which actions or tasks the vehicles should take in order to most efficiently satisfy the objectives. This challenge becomes more difficult with the inclusion of multiple vehicles, because the action and state space scale exponentially with the number of agents. Many planning algorithms suffer from the *curse of dimensionality*;

as more agents are included, sampling for suitable actions in the joint action space becomes infeasible within a reasonable amount of time. To enable autonomy, methods that can be applied to a variety of scenarios are invaluable because they reduce human involvement and time.

Recently, advances in technology enable algorithms that require more computational power to be effective but work in broader frameworks. We offer three main approaches to multi-agent planning which are all inspired by model-based reinforcement learning. First, we address the curse of dimensionality and investigate how to spatially reduce the state space of massive environments where agents are deployed. We do this in a hierarchical fashion by searching subspaces of the environment, called sub-environments, and creating plans to optimally take actions in those sub-environments. Next, we utilize game-theoretic techniques paired with simulated annealing as an approach for agent cooperation when planning in a finite time horizon. One problem with this approach is that agents are capable of breaking promises with other agents right before execution. To address this, we propose several variations that discourage agents from changing plans in the near future and encourages joint planning in the long term. Lastly, we propose a tree-search algorithm that is aided by a convolutional neural network. The convolutional neural network takes advantage of spatial features that are natural in UxV deployment and offers recommendations for action selection during tree search. In addition, we propose some design features for the tree search that target multi-agent deployment applications.

Chapter 1

Introduction

Unmanned vehicles are increasingly deployed in a wide range of challenging scenarios such as use in disaster response, tools for production in factories, and to carry out missions in military applications. A couple examples of tasks that benefit from autonomy include surveillance of an area, search and rescue, pick-up and delivery, and mapping of an unknown and hard to get to area. The use of autonomous vehicles can keep human lives away from dangerous areas, aid in tedius and mundane tasks, and improve performance in areas that are difficult for humans to do as an operator. With the inclusion of multiple agents, more scenarios become relevant that require cooperation and competition amongst agents to perform well. The ability for multiple agents to plan and cooperate is critical across these robotic applications. 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. An example of heterogeneous multi-agent deployment with UxVs is depicted in

Figure 1.1. Generally, in this scenario a large team of human operators are required to send commands to underwater vehicles via communication network of surface and aerial vehicles. This is a domain in which communication is costly and unreliable, so UxVs benefit greatly with increased autonomous capability.

One example of an autonomous deployment scenario where the environment is complex with many uncertain properties is the DARPA Subterranean Challenge. In the challenge, vehicles are tasked with exploring, mapping, and navigating underground tunnels, passages, and facilities. In these harsh, GPS-denied environments there are often multiple levels and vertical passages. Furthermore, unknown changes to the environment are expected to occur, such as closing doors and segments of caves collapsing. One of the goals of the challenge is to inspire innovation in autonomous algorithms for increasing situational awareness, which requires an autonomy framework that is flexible when it comes to environmental assumptions.

In the past, algorithms for multi-agent deployment typically have a narrow scope regarding the task at hand. Algorithms are typically designed to handle a particular mission and are built with many assumptions and dynamics that are relevant to the targeted task. The need for algorithms that are able to handle more generalized tasks are becoming important as many of the tasks that we are interested in have unknown or changing environmental parameters. Logistically, it is time consuming to design, test, and debug algorithms for UxVs, especially in harsh environments where testing could lead to loss or destruction of the vehicle. Because of this, algorithms that are capable of being applied to a variety of scenarios are of great interest to those who use UxVs because it is time consuming to handcraft algorithms for each deployment scenario.

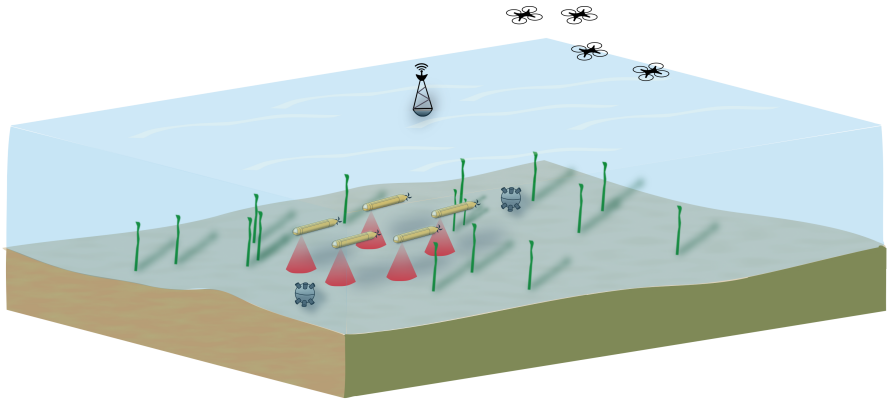


Figure 1.1: A depiction of a multi-agent heterogeneous UxV deployment. In this mission there are multiple objectives. The underwater vehicles are tasked with mapping the seafloor, maintaining communication connectivity amongst themselves, and seeking objects of interest such as mines. Aerial and surface vehicles are tasked with maintaining connectivity to the underwater vehicles to provide localization and commands from a human operator.

Reinforcement learning has become a compelling strategy for multi-agent deployment because it has the capability of generating policies when just given the environment and time. In terms of development time, models of the environment that the agents are to be deployed in can be done on a case-by-case basis. Once that is complete, one of many reinforcement learning algorithms can be trained offline, with little human intervention, in order to generate a policy for agents to follow during online deployment. However, there are many challenges associated with reinforcement learning. For example, given an environment that is generalized into a Markov decision process (MDP), it can be difficult to find optimal policies, which often requires a massive amount of sampling. This problem becomes much worse when multiple agents need to coordinate to develop an optimal joint plan because their action spaces become joint as well. For this reason, the topic of multi-agent planning is particularly interesting and difficult.

Recently, rise in computational power and the miniturization of microcon-

trollers and single board computers are enabling the use of algorithms for multi-agent deployment. Online sampling methods for finding optimal routes can be run on microcontrollers/single board computers on the vehicle itself. Specialized hardware that specifically target popular neural network structures are being manufactured in anticipation of use on smart cars and unmanned vehicles so that inference for machine learning can be done on the vehicle itself. This progress in technology makes the field of multi-agent robotics promising as the computational needs that were once prohibitive, may now be feasible to compute. One example of this is where reinforcement learning has recently been applied to a multitude of competitive games on a professional level such as Chess and Go, and multi-agent games such as such as Dota 2 and Starcraft 3. Inspired by recent technology, we propose several algorithms which utilize offline learning algorithms to aid online sampling algorithms which can now be ran onboard vehicles.

1.1 Literature review

We introduce state-of-the-art strategies that are used for single and multi-agent task planning. We begin with a short overview of general multi-agent deployment algorithms and then move to strategies that treat the environment as a Markov process.

1.1.1 Multi-agent deployment

Multi-agent deployment is a broad term that spans any algorithm that can be used to control a two or more agents. Many of these algorithms target a particular problem such as finding the optimal path in path planning or patrolling an area.

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., [GM04, BCM09, ME10, DM12, DPH$^+$15, CE17] and references therein. To structure the multi-agent planning algorithm, we draw motivation from coordination strategies in swarm robotics where the goal is to satisfy some global objective [GM04, BCM09, ME10, DM12, DPH$^+$15, CE17]. These algorithms utilize interaction and communication between neighboring agents to act cooperatively.

The mentioned algorithms are generally powerful when applied to the scenario that they are designed for. These algorithms are designed for specific tasks and take advantage of particular geometric or structural assumptions that can be made about the environment. Exploiting these assumptions often yield strong theoretical results, however, this makes some algorithms difficult to be applied to a multitude of scenarios or to be applied to uncertain and unstructured environments where algorithms that can generalize often perform well. That being said, one of our motivations is to improve the accessibility of robots being applied to a variety of situations. Currently, if one wants to deploy agents for multiple scenarios, they also have to implement an algorithm that targets each particular scenario, which takes a lot of time. Because of this, we look to algorithms that can applied to many different scenarios which are able to develop policies that are specific to each environment. Such algorithms are generally computationally costly, however as computation power steadily increases these algorithms become more relevant.

1.1.2 Potential games and online stochastic optimization

Game theory provides some powerful tools which can used to as a tool for multi-agent planning. In particular, we are interested in the field of cooperative, where the goal for agents are to maximize utility by employing strategic interactions. One can design the deployment and interactions of multiple agents as a game. Doing so can be powerful because analysis of the agents strategies under particular games and conditions can lead to interesting properties such as agents reaching a Nash equilibrium. Of these frameworks we are particularly interested in potential games [MS96], where the agents' incentive to change strategies are expressed by a global potential function. With a proper potential function, agents reach Nash equilibrium in finite time by incrementally choosing actions that improve their utility, albiet this Nash equilibrium may not be a global optimizer. Potential games have been utilized for cooperative control [MAS09] as well as for cooperative task planning [CMKJ09]. One strategy that agents can utilize in potential games is spatial adaptive play [MAS09, B$^+$93, You98], where an agent at random changes its strategy with respect to a stochastic policy that balances exploration vs exploitation.

Methods that cast multi-agent task planning problems as games benefit by gaining access to properties such as Nash equilibrium convergence. Some of these algorithms however have no guarantee that given enough time, the plans that the agents decide on will be optimal.

1.1.3 Reinforcement learning

Reinforcement learning is relevant in multi-agent task planning because it enables planning in generalized scenarios. 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. [SPS99, BNS08, Put14, How60]. Given a (finite or infinite) time horizon, the MDP framework is conducive to constructing a policy for optimally executing actions in an environment [PT87, Lov91, Bel13]. 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 [BSR16] 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. The dec-POMDP framework [OA16] incorporates joint decision making and collaboration of multiple agents under uncertain and high-dimensional environments. *Masked Monte Carlo Search* is a dec-POMDP algorithm [OAmAH15] 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 [LK00], the Belief Roadmap [PR10] allows an agent to find minimum cost paths efficiently by finding a trajectory through belief spaces. Similarly, the algorithm in [AmCA11] 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.

Another 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 [MKS+15]. Policy optimization reinforcement algorithms focus on directly optimizing a policy of an agent in an environment. Trust Region Policy Optimization (TRPO) [SLA+15] 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) [WMG+17] 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 last 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) [Ber95] and modifications such as the upper confidence bound tree search [KS06]. Many works under this category attempt to address the curse of dimensionality by lowering the state space through either abstracting the state space [HF00], the history in POMDPs [MB96], or the action space [TK04]. Simulated annealing (SA) [KGV83, LA87, MGPO89, Haj88, SK06] fits in this category being a sampling approach for finding a state or action with optimal value in a Markov chain that borrows the idea of annealing from nature and is capable of handling high-dimensional, nonconvex problems.

1.1.4 Imitation learning in tree search

The process of training a policy to mimic the actions of another policy is called imitation learning [HGEJ17]. The policy can also be trained to mimic the value of taking actions at a given state [DLM09]. These techniques have been applied to tree searches in various ways. UCT has been used to generate data for a final policy constructed through imitation learning and deployed on Atari games [GSL$^+$14]. Tree search actions are sometimes used to construct a *default policy* which helps simulate rollouts in UCT [GS07a]. A variation on this is to train a policy to reflect values of certain actions at a given state in UCT, which are used instead of rollouts [SHM$^+$16]. Recently policies have been trained to reflect the distribution of actions selected during a state in the tree search. These policies are then used to influence the selection

of actions in future tree searches in both AlphaZero [SSS+17], MuZero [SAH+19], and Expert Iteration (ExIt) [ATB17]. This improves planning in large action spaces because agents focus on more promising choices based on previous experiences. In both AlphaZero and ExIt, agents play against a version of themselves to improve without human intervention.

Deep learning can be utilized to create trained policies which can predict the actions of other agents and to aid an individual agent's own search [RCN18, HLKT19]. Multi-agent Markov decision processes (MMDPs) [Bou96] are used to describe Markov decision processes using joint actions from multiple agents. Another approach to multi-agent tree search is DEC-MCTS [BCP+19], where agents communicate compressed forms of their tree searches. In contrast, we are interested in the scenario where agents do not communicate their intentions during runtime. The work [PRHK17] combines components of linear temporal logic and hierarchical planning using MCTS with options learned from reinforcement learning, and demonstrates them on simulated autonomous driving. The work [KGKG15] uses tree search to create artificial cyclic policies which improve convergence in the multi-agent patrolling problem.

1.2 Contributions

Of the previously mentioned algorithms, we are most interested in methods which function in a variety environments and are able to take advantage of online computation. Monte-Carlo tree search fits that category well as it has proven results in games of varying structure such as Chess and Go. As we are interested in multi-agent deployment in large, multi-dimensional environments, we first present a variation on

Monte-Carlo tree search that addresses the 'curse of dimensionality' by optimizing actions in subspaces of the environment.

1.2.1 Multi-agent planning in massive environments

In this segment, we aim to address problems that occur in massive environments where the possible outcomes in actions and states make it difficult to effectively sample online. 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 chapter 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. 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.

1.2.2 Trajectory search via simulated annealing

In Chapter 4, we strive to enable a swarm of UxVs to cooperate and complete a large variety of objectives in massive environments. Inspired by simulated annealing, we provide sampling matrix, which determines the probability that an agent will sample an action schedule given that its current solution is another actions schedule. The structure of the sampling matrix that we provide is novel in that the placement of action schedule elements are determined with respect to a finite time horizon. Taking this sampling matrix structure we provide a hand-designed matrix and provide proofs regarding stationary distributions and convergence to Nash equilibrium, when used in the multi-agent case. Inspired by recent work in tree searches guided by neural networks, we utilize convolutional neural networks to generate efficient sampling matrices given the state of the environment. Performance and theoretical results are validated with metrics tested in several environments.

1.2.3 Biased tree search for multi-agent deployment

In Chapter 5, we provide two novel contributions that work together as a refinement of existing state-of-the-art model-based reinforcement learning techniques. The first contribution, *Multi-agent informed policy construction* (`MIPC`), is a process

where we use deep imitation learning to build a heuristic offline, called *informed policy*, that guides the agents' tree search. Our strategy is to develop an informed policy for a small number of agents and use that informed policy to accelerate the tree search for environments with more agents. The second contribution, *Cascading agent tree search* (`CATS`), is a variation on tree search with an action selection that is biased by the informed policy and is catered for multiple agents. We prove convergence to optimal values of the Markov decision process under the Bellman operation. To evaluate the performance of this algorithm, we train a deep neural network as an informed policy, deploy the algorithm distributively across agents, and evaluate the performance across several metrics in the environments of Figure 6.1. In a comparison with similar tree search and model-free reinforcement learning approaches, CATS excels when the number of agents increase and when search time is limited to realistic time constraints for online deployment.

1.3 Organization

In Chapter 2, we introduce the notation and preliminary information that we use in the dissertation. Chapter 3 first introduces Dynamic domain reduction planning which is an algorithm that can be applied for multi-agent planning in massive state environments. Next, in Chapter 4 we present Cooperative dynamic domain reduction planning. In this chapter we discuss problems that arise in Dynamic domain reduction planning and propose a solution in the framework of a potential game. In Chapter 5 we create a variant of Monte-Carlo tree search which uses imitation learning to produce a bias for future action sampling. In contrast to previous state-of-the-art algorithms that do this, we modify the action selection process for the multi-agent case. Finally,

Chapter 6 summarizes our contributions and proposes future work which may address some of the shortcomings that were discovered during the development of this work.

Chapter 1 is coauthored with Cortés, Jorge. The dissertation author was the primary author of this chapter.

Chapter 2

Preliminaries

2.1 Notation

We introduce here essential concepts and tools for the rest of the dissertation, beginning with some notation. We use $\mathbb{Z}$ and $\mathbb{R}$ to denote integers and real numbers, respectively. An objective-oriented approach with the use of tuples is present throughout the dissertation: for an arbitrary tuple $a = \langle b, c \rangle$, the notation $a.b$ means that b belongs to tuple a. Last, $|\mathcal{Y}|$ indicates the cardinality of a set $\mathcal{Y}$. In what follows, we will use t to denote a discrete time step in which an event occurs. We use the notation π to denote significant probability distributions.

Because most of the dissertation is concerned with multi-agent deployment, we universally refer to agents as $\alpha \in \mathcal{A}$. Agents will be able to select actions $\mathrm{a}_\alpha \in \mathrm{A}_\alpha$. The combination of agents actions is the *joint action* $\mathrm{a} \in \mathrm{A}$. In some cases we will use the notation $-\alpha$, which means all other agents, e.g. $\mathrm{a}_{-\alpha}$ means the set of all other agents actions. We use the notation P to denote the probability of an occurrence.

2.2 Markov chains

A Markov chain describes a sequence of states such that the probability of transitioning from one state to another only depends on the current state. We define a Markov chain as a tuple $\langle \mathcal{S}, P^s \rangle$ of states $s \in \mathcal{S}$ and probability $P^s(s'|s)$ that s transitions to s'. Every transition of states in a Markov chain satisfies the Markov property

$$P(s_{t+1}|s_t, s_{t-1}, \ldots, s_0) = P(s_{t+1}|s_t),$$

for all s, which implies that the probability of transitioning to s_{t+1} from s_t is independent of the sequence of states leading up to t. If we assign states of the Markov chain to nodes, and positive state transition probabilities $P^s(s_{t+1}|s_t) > 0$ to edges, we can create a graph of the Markov chain. The *state transition matrix* P^s is a row-stochastic matrix of dimension $|\mathrm{S}| \times |\mathrm{S}|$ where an element $P^s_{s_{t+1},s_t} = P^s(s_{t+1}|s_t)$ such that s_{t+1} corresponds to the i-th row and s_t corresponds to the j-th column. As Markov chains evolve in time with respect to P^s the probability of a particular state being active converges. The row eigenvector that is associated with the eigenvalue 1 is called the *stationary distribution*

$$\pi^s P^s = \pi^s,$$

which describes the long term distribution of states in the Markov chain. A value can be associated to each state s in a Markov chain, $V(s) \in \mathbb{R}$, allowing us

to order nodes in the graph vertically with respect to their value, creating an ordered graph as shown in Figure 4.3. Define $H \in \mathbb{R}$ such that $H \leq V(s)$ for any s. The following properties of a Markov chain are important and can be analyzed by visualization of an ordered graph:

(i) *Strong irreducibility:* there exists a path from s to s' for all s'.

(ii) *Weak reversibility:* if s can be reached by s' at height H, then there exists a path from s' to s at height H as well.

2.3 Simulated annealing

Let the tuple $\langle \mathcal{S}, P, V \rangle$ define a Markov chain with value associated to its state. Simulated annealing seeks to determine $\text{argmax}_s V(s)$, which is generally a combinatorial problem. Algorithm 1 outlines the process of simulated annealing, where $K \in \mathbb{R}$.

Algorithm 1: Simulated annealing

```
1 Initialize s, T
2 for k = 1 to k = K
3     Sample new state s' from P^g
4     if V(s') > V(s) or Random(0,1) < e^((V(s')-V(s))/T):
5         s = s'
6     Update T
```

As the temperature of the system $\mathcal{T}$ is incrementally decreased, the probability that a new solution with lower value is accepted decreases. Simulated annealing yields strong results [KGV83, LA87, Haj88, SK06] and converges to the global optimal if

the temperature is decreased sufficiently slowly. The cooling rate

$$\mathcal{T}_k = \frac{c}{\log(k)}, \tag{2.1}$$

is shown [Haj88] to be a necessary and sufficient condition for the algorithm to converge in probability to a set of states that globally optimize V when the underlying Markov chain has both strong irreducible and weak reversible properties.

2.4 Potential games

In a game theoretic framework agents select strategies, which we will denote as a_α for agent α, in order to maximize their utility $u_\alpha : \mathrm{a}_\alpha \times \mathrm{a}_{-\alpha} \to \mathbb{R}$ for selecting a particular strategy, a_α while the set of actions that other agents have selected is $\mathrm{a}_{-\alpha}$. The set of strategies that agents select is a *Nash equilibrium* when no agent can select a strategy that unilaterally improves their utility given the strategies of other agents, more formally

$$u(\mathrm{a}_\alpha, \mathrm{a}_{-\alpha}) \geq u(\mathrm{a}'_\alpha, \mathrm{a}_{-\alpha}) \quad \forall\ \alpha \in \mathcal{A},\ \mathrm{a}'_\alpha \in \mathrm{A}.$$

In a potential game, the incentive for any agent to change their strategy is represented by a single function called the *potential function* denoted Φ. In this dissertation we are most interested in *exact potential games* which have a potential function of the form

$$\Phi(\mathrm{a}_\alpha, \mathrm{a}_{-\alpha}) - \Phi(\mathrm{a}'_\alpha, \mathrm{a}_{-\alpha}) = u(\mathrm{a}_\alpha, \mathrm{a}_{-\alpha}) - u(\mathrm{a}'_\alpha, \mathrm{a}_{-\alpha}),$$

such that $\Phi : \mathrm{a}_\alpha \times \mathrm{a}_{-\alpha} \to \mathbb{R}$. We will take advantage of the *wonderful life utility*

$$u_\alpha(\mathrm{a}_\alpha, \mathrm{a}_{-\alpha}) = \Phi(\mathrm{a}_\alpha, \mathrm{a}_{-\alpha}) - \Phi(\emptyset, \mathrm{a}_{-\alpha}),$$

where $\emptyset$ is the null strategy of agent α in which α does not contribute to the completion of any objective in the game. The wonderful life utility is a measure of the marginal gain that an agent contributes when selecting a given action. It is straightforward to see realize that the wonderful life utility is one possible type of utility that yields a potential game. Generally, potential games yield at least 1 pure Nash equilibrium. An *improvement path* is any sequence of strategies ${\mathrm{a}^t}_{t=0,1,\ldots}$ such that $u(\mathrm{a}^{t+1}) > u(\mathrm{a}^t)$ wherever $action^{t+1}$ is defined. A *finite improvement path* is an improvement path that terminates at a Nash equilibrium. Exact and weighted potential games have finite improvement paths.

2.5 Markov decision processes

A Markov decision process (MDP) is a tuple $\langle \mathrm{A}, \mathrm{S}, \mathcal{R}, P^s, \gamma \rangle$, where $\mathrm{s} \in \mathrm{S}$ and $\mathrm{a} \in \mathrm{A}$ are state and action spaces respectively; $P^s(\mathrm{s}'|\mathrm{s}, \mathrm{a})$ is the transition function which encodes the probability of the next state being s' given current state s and

action a. After every transition, a reward is obtained according to the reward function $\mathcal{R} : \mathrm{s}', \mathrm{a}, \mathrm{s} \to r \in \mathbb{R}$. A policy π specifies the actions given a state according to $\pi : \mathrm{s}, \mathrm{a} \to (0, 1)$ such that $\sum_{\mathrm{a} \in \mathrm{A}} \pi_{\mathrm{s,a}} = 1$.

Given a policy π, the *state value* is

$$V_{\mathrm{s}}^{\pi} = \sum_{\mathrm{a} \in \mathrm{A}} \pi_{\mathrm{s,a}} \sum_{\mathrm{s}' \in \mathrm{S}} P^{s}(\mathrm{s}'|\mathrm{s}, \mathrm{a}) \Big(\mathcal{R}_{\mathrm{s}',\mathrm{a},\mathrm{s}} + \gamma V_{\mathrm{s}'}^{\pi} \Big),$$

where $\gamma \in [0, 1]$ is a discount factor. The *state-action value* is the value of taking an action at state is

$$Q_{\mathrm{s,a}}^{\pi} = \sum_{\mathrm{s}' \in \mathrm{S}} P^{s}(\mathrm{s}'|\mathrm{s}, \mathrm{a}) \Big(\mathcal{R}_{\mathrm{s}',\mathrm{a},\mathrm{s}} + \gamma V_{\mathrm{s}'}^{\pi} \Big).$$

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 $\phi_{\mathrm{s}} : \mathrm{s} \to \mathrm{s}_{\phi}$. Similarly, actions can be abstracted with an action abstraction function $\phi_{\mathrm{a}} : \mathrm{a} \to \mathrm{a}_{\phi}$. 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, $P^{t}(t|\mathrm{a}_{\phi}, \mathrm{s})$. When considering the problem using abstracted actions $\mathrm{a}_{\phi} \in \mathrm{A}_{\phi}$ in $\langle \mathrm{S}, \mathrm{A}_{\phi}, P^{s}, \mathcal{R}, P^{t} \rangle$, 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.

Amongst several extensions of MDPs are multi-agent Markov decision processes (MMDP). A MMDP is a tuple $\langle \mathrm{S}, \mathcal{A}, \{\mathrm{A}_\alpha\}_{\alpha \in \mathcal{A}}, P^s, R \rangle$, similar to a MDP with the addition of agents $\alpha \in \mathcal{A}$, where each agent has an action space $\{\mathrm{A}_\alpha\}$. With this modification, the state transition function P^s maps the probability of state transition with respect to the joint actions of agents in $\mathcal{A}$. One strategy for multi-agent cooperation is to reason over the joint action space as in MMDPs. This solution does not scale with the number of agents, as the joint action space increases and the problem dimension grows exponentially.

2.6 Model-based reinforcement learning

This section introduces basic preliminaries on Markov decision processes and Monte-Carlo tree searches. We also discuss imitation learning and its use to bias tree searching in multi-agent environments to improve performance.

When the transition function of the MDP is known, a popular method for approximating the optimal policy is Monte-Carlo tree search [BPW$^+$12]. There are four major steps in MCTS: selection, expansion, simulation, and backpropagation. During the selection process, actions are chosen from A to transition the MDP until a state has been reached for the first time, where it then expands the tree by one node. The next step is to use a predefined rollout policy to simulate future moves until a specified depth or state. If no information is known regarding the environment, it is common for the rollout policy to return a random move. Finally, rewards obtained

during the tree traversal are backpropagated to update the estimated values for taking actions at the visited states.

Upper confidence bound tree search (UCT) [KS06] executes the first step with the following action selection policy

$$\underset{\mathrm{a}\in\mathrm{A}}{\operatorname{argmax}}\left(\hat{Q}_{\mathrm{s,a}}+c_1\sqrt{\frac{\ln N_{\mathrm{s}}}{N_{\mathrm{s,a}}}}\right). \tag{2.2}$$

Here the first term, $\hat{Q}_{\mathrm{s,a}}$ is the empirical value estimation for choosing action a at state s, which is exploitive and influences the action selection towards actions that yielded higher rewards in previous iterations of the tree search. In the second term, N_{s} and $N_{\mathrm{s,a}}$ are the number of times that the state has been visited and the number of times that a particular action has been chosen at that state, respectively. The second term is explorative and biases the search towards actions that have been selected least often at a state. The probability that the optimal action is selected by UCT goes to 1 as shown in rigorous finite-time analysis [KSW06, ACBF02].

Tree search bias via imitation learning

Deep learning has also been used [GSL$^+$14, GS07a, SSS$^+$17, SHM$^+$16, ATB17] to predict which actions to take in a never before visited state. We call the resulting network an *informed policy*, $\hat{\pi}$, which specifies that training used data created from previous iterations of tree search on the environment. In both ExIt [ATB17] and AlphaZero [SSS$^+$17], an informed policy is trained to resemble the *action selection distribution* $N_{\mathrm{s,a}}/N_{\mathrm{s}}$. The informed policy is then used to improve future tree searches

by biasing the action selection of the tree search as follows

$$\underset{\mathrm{a}\in\mathrm{A}}{\operatorname{argmax}}\left(\hat{Q}_{\mathrm{s,a}}+c_1\sqrt{\frac{\ln N_{\mathrm{s}}}{N_{\mathrm{s,a}}}}+c_2\frac{\hat{\pi}_{\mathrm{s,a}}}{N_{\mathrm{s,a}}}\right), \tag{2.3}$$

where c_2 weights the neural networks influence on action selection. Our algorithm builds on this idea to modify the action selection by tempering the explorative term using a sequential process in order to balance exploitation versus exploration. In the multi-agent domain, the informed policy can be used to enable distributed tree searches. To do this, one agent will search using its own action space (as opposed to the joint action space amongst all agents) while other agents essentially become part of the environment and are modeled by taking the max-likelihood action according to $\hat{\pi}$.

2.7 Submodularity

In the analysis of our algorithms, we rely on the notion of submodular set functions and the characterization of the performance of greedy algorithms, see e.g., [CABP16, BBKT17]. Even though some processes of our algorithms are not completely submodular, we are able to invoke these results by resorting to the concept of submodularity ratio [DK11], that quantifies how far a set function is from being submodular, using tools from scenario optimization [CGP09].

We review here concepts of submodularity and monotonicity of set functions following [CABP16]. A power set function $f : 2^{\Omega} \to \mathbb{R}$ is *submodular* if it satisfies the

property of diminishing returns,

$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y), \tag{2.4}$$

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

$$f(X) \leq f(Y), \tag{2.5}$$

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

$$\begin{aligned} &\max \ f(X) \\ &\text{s.t. } |X| \leq k. \end{aligned} \tag{2.6}$$

In general, this problem is NP-hard. Greedy algorithms seek to find a suboptimal solution to (2.6) 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 [CABP16, NWF78] provides a lower bound on the performance of greedy algorithms.

Theorem 2.7.1. *Let X^* denote the optimal solution of problem (2.6). 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 [GS07b] characterizes the performance.

Theorem 2.7.2. *Let X^* denote the optimal solution of problem (2.6). 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* [DK11], which measures how far the function is from being submodular. This ratio is defined as largest scalar $\lambda \in [0, 1]$ such that

$$\lambda \leq \frac{\sum_{z \in Z} f(X \cup \{z\}) - f(X)}{f(X \cup Z) - f(X)}, \tag{2.7}$$

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 [DK11] generalizes Theorem 2.7.1 to monotone set functions with submodularity ratio λ.

Theorem 2.7.3. *Let X^* denote the optimal solution of problem (2.6). 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*

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

2.8 Scenario optimization

Scenario optimization aims to determine robust solutions for practical problems with unknown parameters [BTN98, GOL98] by hedging against uncertainty. Consider the following *robust convex optimization problem* defined by

$$\begin{aligned} \text{RCP: } & \min_{\gamma \in \mathbb{R}^d} c^T \gamma \\ & \text{subject to: } f_\delta(\gamma) \leq 0, \ \forall \delta \in \Delta, \end{aligned} \tag{2.8}$$

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 (2.8) can be difficult depending on the cardinality of Δ. One approach to this problem is to solve (2.8) 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*[CGP09] defined by

$$\begin{aligned} \text{SCP}_N : &\min_{\gamma \in \mathbb{R}^d} c^T \gamma \\ &\text{s.t. } f_{\delta^{(i)}}(\gamma) \leq 0, \ i = 1, \ldots, N_{\text{SCP}}. \end{aligned} \tag{2.9}$$

The following result states to what extent the solution of (2.9) solves the original robust optimization problem.

Theorem 2.8.1. *Let γ^* be the optimal solution to the scenario convex program (2.9) when N_{SCP} is the number of convex constraints. Given a 'violation parameter', ε, and a 'confidence parameter', ϖ, if*

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

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

Chapter 2, is coauthored with Cortés, Jorge and Ouimet, Mike. The dissertation author was the primary author of this chapter.

Chapter 3

Dynamic domain reduction planning

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 chapter 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.

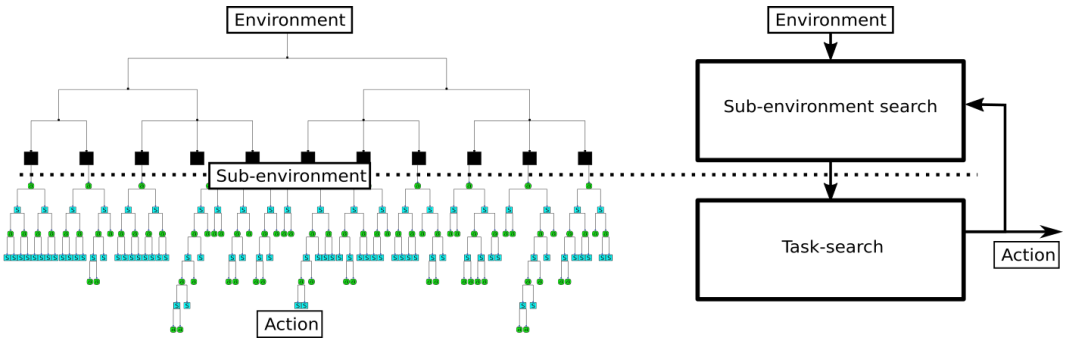


Figure 3.1: Workflow of the proposed hierarchical algorithm. A *sub-environment* is dynamically constructed as series of spatial-based state abstractions in the environment in a process called `SubEnvSearch`. Given this sub-environment, an agent performs *tasks*, which are abstracted actions constrained 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`.

3.1 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.

3.2 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.

3.2.1 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.

Region abstraction

We define a *region* to be a convex set $\omega \subseteq \mathbb{R}^d$. Let Ω_ω be a set of disjoint regions where the union of all regions in Ω_ω 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 ω_i, i.e., such that $\mathcal{O}_i^b \subseteq \omega_i$. We use an abstraction function, $\phi : \mathcal{O}_i^b \to \mathrm{s}_i^b$, 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*, $s_i = (\mathrm{s}_i^1, \ldots, \mathrm{s}_i^{|\mathcal{O}|})$, to be the Cartesian product of s_i^b for all objectives, $\mathcal{O}^b \in \mathcal{O}$, with respect to ω_i.

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 $\tau = \langle \mathrm{s}_i^b, \mathrm{s}_i'^b, \omega_i, \omega_j, b \rangle$, where s_i^b and $\mathrm{s}_i'^b$ are abstracted objective states associated to ω_i, ω_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, $\tau.\mathrm{s}_i^b \rightarrow \tau.\mathrm{s}_i'^b$.
- Regional transition: low-level controller executing τ drives the agent's location to ω_j after the objective transition is complete.

Candidates for low-level controllers include policies determined using the approaches in [BSR16, KS06] after setting up the region as a MDP, modified traveling salesperson [BCK+07], 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 $P^t(t|\tau)$, 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 $P^t(\infty|\tau) = 1$ and the agent never completes it.

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*, $\overrightarrow{\omega}$, be a sequence of regions of length up to $N_\epsilon \in \mathbb{Z}_{\geq 1}$. The k^{th} region in $\overrightarrow{\omega}$ is denoted with $\overrightarrow{\omega}_k$. The regional state of $\overrightarrow{\omega}_k$ is given by $s_{\overrightarrow{\omega}_k}$. For example, $\overrightarrow{\omega} = [\omega_2, \omega_1, \omega_3]$ is a valid sub-environment region sequence with $N_\epsilon \geq 3$, the first region in $\overrightarrow{\omega}$ is $\overrightarrow{\omega}_1 = \omega_2$, and the regional state of $\overrightarrow{\omega}_1$ is $s_{\overrightarrow{\omega}_1} = s_{\omega_2}$. In order

to simulate the sub-environment, the agent must know if there are repeated regions in $\overrightarrow{\omega}$. Let $\xi(k, \overrightarrow{\omega})$, return the first index h of $\overrightarrow{\omega}$ such that $\overrightarrow{\omega}_h = \overrightarrow{\omega}_k$. Define the repeated region list to be $\xi_{\overrightarrow{\omega}} = [\xi(1, \overrightarrow{\omega}), \ldots, \xi(N_\epsilon, \overrightarrow{\omega})]$. Let $\phi_t(\omega_i, \omega_j) : \omega_i, \omega_j \to \mathbb{Z}$ be an abstracted amount of time it takes for an agent to move from ω_i to ω_j, or ∞ if no path exists. Let $s = [s_{\overrightarrow{\omega}_1}, \ldots, s_{\overrightarrow{\omega}_{N_\epsilon}}] \times \xi_{\overrightarrow{\omega}} \times [\phi_t(\overrightarrow{\omega}_1, \overrightarrow{\omega}_2), \ldots, \phi_t(\overrightarrow{\omega}_{N_\epsilon-1}, \overrightarrow{\omega}_{N_\epsilon})]$ be the *sub-environment state* for a given $\overrightarrow{\omega}$, and let S^ϵ be the set of all possible sub-environment states. We define a *sub-environment* to be $\epsilon = \langle \overrightarrow{\omega}, s \rangle$.

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.

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 $\overrightarrow{\tau}_k.\omega_i = \epsilon.\overrightarrow{\omega}_k$, and $\overrightarrow{\tau}_k.\omega_j = \epsilon.\overrightarrow{\omega}_{k+1}$ for all $k \in [N_\epsilon - 1]$, where ω_i and ω_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 $\overrightarrow{P^t}_k$. We define $\overrightarrow{P^t} = [\overrightarrow{P^t}_1, \ldots, \overrightarrow{P^t}_{N_\epsilon-1}]$ to be the ordered list of probability distributions. We construct the *task trajectory* to be $\vartheta = \langle \overrightarrow{\tau}, \overrightarrow{P^t} \rangle$, which is used to determine the finite time reward for a sub-environment.

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 P^r that is determined heuristically. Let r^ϵ be the *abstracted reward function*, determined by

$$r^\epsilon(\tau) = \sum_{r \in \mathbb{R}} P^r(r|\tau) r, \tag{3.1}$$

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 $\epsilon.\overrightarrow{\omega}_1$ and end up in region $\epsilon.\overrightarrow{\omega}_2$ 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 $\epsilon.\overrightarrow{\omega}$ 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

$$\epsilon' = \langle \overrightarrow{\omega} = [\overrightarrow{\omega}_2, \ldots, \overrightarrow{\omega}_{N_\epsilon}], s = P^s(\epsilon'.s|\epsilon.s, \tau) \rangle. \tag{3.2}$$

Finally, we can represent the process of executing tasks in sub-environments as the *sub-environment SMDP* $\mathcal{M} = \langle \mathrm{S}^\epsilon, \Gamma, P^{s^\epsilon}, r^\epsilon, P^t \rangle$. The goal of the agent is to

determine a policy $\pi : \epsilon.s \to \tau$ that yields the greatest rewards in $\mathcal{M}$. The state value under policy π is given by

$$V^{\pi}_{\epsilon.s} = r^{\epsilon} + \sum_{t^{\epsilon} \in \mathbb{R}} P^{t^{\epsilon}} \gamma^{t^{\epsilon}} \sum_{\epsilon.s \in S^{\epsilon}} P^{s} V^{\pi}_{\epsilon.s'} \tag{3.3}$$

We strive to generate a policy that yields optimal state value

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

with associated optimal value $V^{\pi *}_{\epsilon.s} = \max_{\pi} V^{\pi}_{\epsilon.s}$.

Remark 3.2.1 (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,* $\overrightarrow{P^{t}}_{k}(\infty) = 1$*, 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.* •

3.2.2 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 $\beta \in \mathcal{A}$.

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 $\theta = \langle \mathcal{O}^b, \omega_i \rangle$. 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 $\Theta'_\alpha = \Theta^{\mathcal{A}} \setminus \left\{ \bigcup_{\beta \in \mathcal{I}_\alpha} \Theta_\beta \right\}$ 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.

$$s^b_{\Theta'_\alpha,\overrightarrow{\omega}_k} = \begin{cases} 0 & \text{if } \langle \mathcal{O}^b, \epsilon_\alpha.\overrightarrow{\omega}_k \rangle \in \Theta'_\alpha, \\ s^b_{\overrightarrow{\omega}_k} & \text{otherwise.} \end{cases} \tag{3.4}$$

Let $\overrightarrow{\mathcal{S}}_{\Theta'_\alpha,\overrightarrow{\omega}} = \overrightarrow{\mathcal{S}}_{\Theta'_\alpha,\overrightarrow{\omega}_1} \times \ldots \times \overrightarrow{\mathcal{S}}_{\Theta'_\alpha,\overrightarrow{\omega}_{N_\epsilon}}$, where $\overrightarrow{\mathcal{S}}_{\Theta'_\alpha,\overrightarrow{\omega}_k} = (s^{\mathcal{O}^1}_{\Theta'_\alpha,\overrightarrow{\omega}_k}, \ldots, s^{\mathcal{O}^{|\mathcal{O}|}}_{\Theta'_\alpha,\overrightarrow{\omega}_k})$. 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 $\vartheta^p_{\beta,k} = \langle \vartheta_\beta.\overrightarrow{\tau}_k.\mathrm{s}^b_i,\ \vartheta_\beta.\overrightarrow{\tau}_k.b \rangle$. The *partial trajectory*, $\vartheta^p_\beta = [\vartheta^p_{\beta,1}, \ldots, \vartheta^p_{\beta,|\vartheta_\beta|}]$ describes β's trajectory with respect to $\epsilon_\alpha.\overrightarrow{\omega}$. Let $\xi(k, \epsilon_\alpha, \epsilon_\beta)$ return the first index of $\epsilon_\alpha.\overrightarrow{\omega}$, h, such that $\epsilon_\beta.\overrightarrow{\omega}_h = \epsilon_\alpha.\overrightarrow{\omega}_k$, 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*, $s = \overrightarrow{\mathcal{S}}_{\Theta'_\alpha,\overrightarrow{\omega}} \times \left\{ \langle \vartheta^p_1, \Xi_1 \rangle, \ldots, \langle \vartheta^p_{|\mathcal{I}_\alpha|}, \Xi_{|\mathcal{I}_\alpha|} \rangle \right\} \times [\phi_t(\overrightarrow{\omega}_1, \overrightarrow{\omega}_2), \ldots, \phi_t(\overrightarrow{\omega}_{N_\epsilon - 1}, \overrightarrow{\omega}_{N_\epsilon})]$. With these modifications to the sub-environment state, we define the (multi-agent) *sub-environment* as $\epsilon_\alpha = \langle \overrightarrow{\omega}, s, \mathcal{I}_\alpha \rangle$.

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 $\mathrm{s}^b_i \rightarrow \mathrm{s}^{\prime b}_i$. 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 $P^t_{\mathcal{I}_\alpha}(t|\tau, T^{\mathcal{A}}_{\mathcal{I}_\alpha})$ the probability that τ is completed at exactly time t if other agents work on transitioning $\mathrm{s}_i^b \rightarrow \mathrm{s}_i'^b$. We redefine here the definition of $\overrightarrow{P^t}$ in Section 3.2.1, as $\overrightarrow{P^t} = [\overrightarrow{P^t}_{1,\mathcal{I}_\alpha}, \ldots, \overrightarrow{P^t}_{N_\epsilon-1,\mathcal{I}_\alpha}]$, 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 $\overrightarrow{P^{t\prime}} = [\overrightarrow{P^{t\prime}}_{1,\mathcal{I}_\alpha}, \ldots, \overrightarrow{P^{t\prime}}_{N_\epsilon-1,\mathcal{I}_\alpha}]$. With these modifications, we redefine the *task trajectory* to be $\vartheta = \left\langle \overrightarrow{\tau}, \overrightarrow{P^t} \right\rangle$. Finally, we designate X_ϑ to be the set that contains all trajectories of the agents.

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

$$r^\phi(\tau, X_\vartheta) = \begin{cases} \mathfrak{R}(\tau, X_\vartheta) & \text{if } \tau \in \vartheta^p_\beta \text{ for any } \beta, \\ r^\epsilon(\tau) & \text{otherwise.} \end{cases} \tag{3.5}$$

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 $\mathcal{M} = \langle S^{\epsilon}, \Gamma, P^{s}, r^{\phi}, P^{t}_{\mathcal{I}_\alpha} \rangle$. The state value from (4.2) is updated using (3.5)

$$V^{\pi}_{\epsilon.s} = r^{\phi} + \sum_{t^{\epsilon} \in \mathbb{R}} P^{t^{\epsilon}} \gamma^{t^{\epsilon}} \sum_{\epsilon.s \in S'^{\epsilon}} P^{s} V^{\pi}_{\epsilon.s'}. \tag{3.6}$$

We strive to generate a policy that yields optimal state value

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

with associated optimal value $V^{\pi *}_{\epsilon.s} = \max_{\pi} V^{\pi}_{\epsilon.s}$. Our next section introduces an algorithm for approximating this optimal state value.

3.3 Dynamic Domain Reduction for Multi-Agent Planning

This section describes our algorithmic solution to approximate the optimal policy π^{*}. The *Dynamic Domain Reduction for Multi-Agent Planning* algorithm consists of three main functions: `DDRMAP`, `TaskSearch`, and `SubEnvSearch`[1]. Algorithm 2 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.

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

Algorithm 2: : Dynamic Domain Reduction for Multi-Agent Planning

$\Omega_\omega = \bigcup \{x\}\ \forall x$
$\mathcal{E} \leftarrow$ current environment
$\Theta^{\mathcal{A}} \leftarrow \bigcup \Theta_\beta,\ \forall \beta \in \mathcal{A}$
$\hat{Q}, V_\omega, N_{\epsilon.s}, N_b \leftarrow$ loaded from previous trials
DDRMAP $(\Omega_\omega, N_\epsilon, \mathcal{E}, \Theta^{\mathcal{A}}, \hat{Q}, V_\omega, N_{\epsilon.s}, N_b, \mathcal{M})$:
    $Y_\vartheta = \emptyset$
    while run time < step time:
        $\epsilon \leftarrow$SubEnvSearch $(\Omega_\omega, \mathcal{E}, \Theta^{\mathcal{A}}, V_\omega)$
        TaskSearch $(\hat{Q}, N_{\epsilon.s}, N_b, \epsilon)$
        $Y_\vartheta = Y_\vartheta \cup \{$MaxTrajectory$(\epsilon)\}$
    return Y_ϑ

TaskSearch $(\hat{Q}, N_{\epsilon.s}, N_b, \epsilon)$
    if $\epsilon.\overrightarrow{\omega}$ is empty
        return 0
    $\tau \leftarrow \max_{\tau \in \mathcal{M}.\Gamma} \left\{ \hat{Q}[\epsilon.s][\tau.b] + 2C_p \sqrt{\frac{lnN_{\epsilon.s}[\epsilon.s]}{N_b[\epsilon.s][\tau.b]}} \right\}$
    $t \leftarrow$ Sample† $\mathcal{M}.P^t(t|\tau, \epsilon)$
    $\epsilon' = \langle [\epsilon.\overrightarrow{\omega}_2 \ldots, \epsilon\overrightarrow{\omega}_{N_\epsilon}]$,Sample† $\mathcal{M}.Pr^s(\epsilon.s, \tau)\rangle$
    $r = \mathcal{M}.r^\epsilon(\tau, \epsilon) + \gamma^t$TaskSearch $(\hat{Q}, N_{\epsilon.s}, N_b, \epsilon')$
    TaskValueUpdate $(\hat{Q}, N_{\epsilon.s}, N_b, \epsilon.s, \tau.b, r)$
    return r

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

SubEnvSearch $(\Omega_\epsilon, \mathcal{E}, \Theta^{\mathcal{A}}, V_\omega)$
    $X_\omega = \emptyset$
    while $|X_\omega| < N_\epsilon$:
        for $x \in \Omega_\epsilon$:
            if $V_\omega[X_\omega \cup \{x\}]$ is empty:
                $V_\omega[X_\omega \cup \{x\}] = \left\{ \max_{\tau \in \mathcal{M}.\Gamma} \hat{Q}[$InitSubEnv$(X_\omega, \mathcal{E}, \Theta^{\mathcal{A}}).S][\tau] \right\}$
        $x = \underset{x \in \Omega_\epsilon}{\text{argmax}}\ V_\omega[X_\omega \cup \{x\}]$
        $X_\omega = X_\omega \cup \{x\}$
    return InitSubEnv$(X_\omega, \mathcal{E}, \Theta^{\mathcal{A}})$

InitSubEnv $(X_\omega, \mathcal{E}, \Theta^{\mathcal{A}})$
    $\overrightarrow{\omega} = \underset{\overrightarrow{\omega} \in \binom{X_\omega}{|X_\omega|} \setminus \mathbb{V}_\epsilon}{\text{argmax}} \left\{ \max_{\tau \in \mathcal{M}.\Gamma} \hat{Q}[$GetSubEnvState$(\overrightarrow{\omega}, \Theta^{\mathcal{A}})][\tau] \right.$
    $\epsilon = \langle \overrightarrow{\omega}$, GetSubEnvState$(\overrightarrow{\omega}, \Theta^{\mathcal{A}})\rangle$
    return ϵ

The following variables are common across the multi-agent system: the set of regions Ω_ω, 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_ω. 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 P^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 (3.2). On line 18, we get the discounted reward of the sub-environment by summing the reward for the current task using (3.1,3.5) 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 $\epsilon.\overrightarrow{\omega}$. `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 $X_\omega \subseteq \Omega_\omega$ (we use subindex 'ω' 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 $\overrightarrow{\omega}$ given a X_ω that maximizes $\max_{\tau\in\Gamma} \hat{Q}[\epsilon.s][\tau]$ on line 35. We keep track of the expected value of choosing X_ω and the sub-environment that is returned by `InitSubEnv` with V_ω. `SubEnvSearch` begins by initializing X_ω to empty. The region that increases the value V_ω the most when appended to X_ω is then appended to X_ω. This process is repeated until the length of X_ω is N_ϵ. Finally, the best sub-environment given X_ω is returned with `InitSubEnv`. The time complexity of `InitSubEnv` and `SubEnvSearch` is $O(N!)$ and $O(|\Omega_\omega|N_\epsilon \log(N_\epsilon!))$, respectively. `InitSubEnv` requires iteration over all possible permutations of X_ω, however in practice we use heuristics to reduce the time of computation.

3.4 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.

3.4.1 `TaskSearch` estimated value convergence

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

Assumption 3.4.1. *There always exists a task that an agent can complete in finite time. Furthermore, no task can be completed in zero time steps.*

Assumption 3.4.1 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 3.4.2. *Let $\hat{Q}$ be the estimated value of performing a task determined over time using `TaskSearch`. Under Assumption 3.4.1, $\hat{Q}$ 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 [PR98], rewritten here to match our notation:

(i) State and action spaces are finite;

(ii) $\text{Var}\{r^\epsilon\}$ is finite;

(iii) $\sum_{p=1}^{\infty} \kappa_p(\epsilon.s, \tau) = \infty$ and $\sum_{p=1}^{\infty} \kappa_p^2(\epsilon.s, \tau) < \infty$ uniformly over $\epsilon.s, \tau$;

(iv) $0 < \mathcal{B}^{\max} = \max_{\epsilon.s \in S^\epsilon, \tau \in \Gamma} \sum P^t(t|\epsilon.s\tau)\gamma^t < 1$.

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 $\text{Var}\{r^\epsilon\} = 0$ 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 3.4.1 (there always exists some τ such that $P^t(\infty|\epsilon.s, \tau) < 1$) and the fact that $\gamma \in (0, 1)$ to ensure that $\mathcal{B}^{\max}$ will always be greater than 0. We use Assumption 3.4.1 (for all $\epsilon.s, \tau$ $P^t(t > 0|\epsilon.s\tau) = 1$) 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 3.4.2 as justification for the following assumption.

Assumption 3.4.3. *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.

3.4.2 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 3.4.4. *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 2.7, 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_ω be a finite set of regions. `InitSubEnv` takes a set of regions $X_\omega \subseteq \Omega_\omega$ and returns the sub-environment made up of regions in X_ω in optimal order. We define the power set function $f_\omega : 2^{\Omega_\omega} \to \mathbb{R}$, mapping each set of regions to the discounted reward that an agent expects to receive for choosing the corresponding sub-environment

$$f_\omega(X_\omega) = \max_{\tau \in \Gamma} \hat{Q}[\texttt{InitSubEnv}(X_\omega).s][\tau]. \tag{3.7}$$

For convenience, let $X_\omega^* = \text{argmax}_{X_\omega \in \Omega_\omega} \ f_\omega(X_\omega)$ 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_ω is, in general, not submodular.

Lemma 3.4.5. *Let f_ω be the discounted reward that an agent expects to receive for choosing the corresponding sub-environment given a set of regions, as defined in (3.7). Under Assumptions 3.4.3-3.4.4, f_ω is not submodular.*

Proof. We provide a counterexample to show that f_ω is not submodular in general. Consider a 1-dimensional environment. For simplicity, let regions be singletons, where $\Omega_\omega = \{\omega_0 = \{0\}, \omega_1 = \{-1\}, \omega_2 = \{1\}, \omega_3 = \{2\}\}$. 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, $t = |\tau.\omega_i - \tau.\omega_j|$. Let $X_\omega = \{x_1\} \subset Y_\omega = (x_1, x_3) \subset \Omega_\omega$. f_ω is submodular only if the marginal gain including $\{x_2\}$ is greater for X_ω than Y_ω. Assuming that the agent begins in ω_0, one can verify that the region sequences of the sub-environments returned by `InitSubEnv` are as follows:

$$\texttt{InitSubEnv}(X_\omega) \to \overrightarrow{\omega} = [x_1]$$

$$\texttt{InitSubEnv}(X_\omega \cup \{x_2\}) \to \overrightarrow{\omega} = [x_1, x_2]$$

$$\texttt{InitSubEnv}(Y_\omega) \to \overrightarrow{\omega} = [x_1, x_3]$$

$$\texttt{InitSubEnv}(Y_\omega \cup \{x_2\}) \to \overrightarrow{\omega} = [x_2, x_3, x_1]$$

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

$$f_\omega(X_\omega \cup \{x_2\}) - f_\omega(X_\omega) =$$
$$(\gamma^{t_1} r + \gamma^{t_1}\gamma^{t_2} r) - (\gamma^{t_1} r) \approx .73r,$$

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_ω is

$$f_\omega(Y_\omega \cup \{x_2\}) - f_\omega(Y_\omega) =$$
$$(\gamma^{t_3} r + \gamma^{t_3}\gamma^{t_4} r + \gamma^{t_3}\gamma^{t_4}\gamma^{t_5} r) - (\gamma^{t_1} r + \gamma^{t_1}\gamma^{t_2} r) \approx .74r,$$

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_ω than X_ω. Hence, f_ω is not submodular. • □

Even though f_ω 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 (2.7), the submodularity ratio of a function f_ω is the largest scalar $\lambda \in [0, 1]$ such that

$$\lambda \leq \frac{\sum_{z \in Z_\omega} f_\omega(X_\omega \cup \{z\}) - f_\omega(X_\omega)}{f_\omega(X_\omega \cup Z_\omega) - f_\omega(X_\omega)} \tag{3.8}$$

for all $X_\omega, Z_\omega \subseteq \Omega_\omega$. 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 3.4.6. *Let X_ω be region set returned by the sub-environment search algorithm in* `DDRP-OO`*. Under Assumptions 3.4.3-3.4.4, it holds that $f_\omega(X_\omega) \geq (1 - e^{-\lambda}) f_\omega(X^*_\omega)$.*

Proof. According to Theorem 2.7.3, we need to show that f_ω is a monotone set function, that $f_\omega(\emptyset) = 0$, and that the sub-environment search algorithm has greedy characteristics. Because of Assumption 3.4.4, adding regions to X_ω monotonically increases the expected reward, hence equation (2.5) is satisfied. Next, we note that if $X_\omega = \emptyset$, then `InitSubEnv` returns an empty sub-environment, which implies from equation (3.7) that $f_\omega(\emptyset) = 0$. 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 2.7.3.

• □

Algorithm 3: : Submodularity ratio estimation

$\Delta \leftarrow$ set of all possible pairs of X_ω, Z_ω.
$Pr^\delta \leftarrow$ probability distribution of sampling δ from Δ.

`submodulariyRatioEstimation` $(\varpi, \varepsilon, \Omega_\omega, \Delta, Pr^\delta)$
    $\lambda^+ = 1, d = 1, h = \emptyset$
    $N_{\text{SCP}} = \frac{2}{\varepsilon}(ln\frac{1}{\varpi} + d)$
    for $n = 0; n$++$; n < N_{\text{SCP}}$
        $X_\omega, Z_\omega =$`Sample`† (Pr^δ)
        $\lambda^\delta = \frac{\sum_{z \in Z_\omega} f_\omega(X_\omega \cup \{z\}) - f_\omega(X_\omega)}{f_\omega(X_\omega \cup Z_\omega) - f_\omega(X_\omega)}$
        if $\lambda^\delta < \lambda^+$
            $\lambda^+ = \lambda^\delta$
            h.append(λ^δ, n)
    $a, b = \underset{a,b \in \mathbb{R}}{\text{argmin}} \sum_{\lambda^\delta, n \in h} (\lambda^+ - a - bn)^2$
    $\hat{\lambda} = a + b|\Delta|$
    return $\hat{\lambda}$

Given the generality of our proposed framework, the submodularity ratio of f_ω is in general difficult to determine. To deal with this, we resort to tools from scenario optimization, cf. Appendix 2.8, 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 3, 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 3.4.7. *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 (2.8).*

Proof. First, we show `submodulariyRatioEstimation` can be formulated as a scenario convex program and that it satisfies the convex constraint condition in Theorem 2.8.1. Lines 6-11 provide a solution, λ^+, to the following scenario convex program.

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

where line 8 is a convex function that comes from equation (3.8). Since f_{δ^i} is a convex function, we can apply Theorem 2.8.1; 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 $n = N_{\text{SCP}}$ which implies that $\hat{\lambda} \leq \lambda^+$ 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_ω 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 3.4.8. *Under Assumptions 3.4.1-3.4.4, 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 (3.8) 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 3.4.4, 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 3.4.4, concluding the proof. • □

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

Remark 3.4.9. *Beyond the result in Theorem 3.4.8, 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 Section 3.5.1 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.* •

3.4.3 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 3.4.10. *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

$$\hat{W}^{\omega}_{\tau,X_\vartheta} = \max_{\vartheta \in X_\vartheta} \vartheta.\overrightarrow{P^t}_k \gamma^t r^\phi = \max_{\vartheta \in X_\vartheta} \hat{W}^{\omega}_{\tau,\{\vartheta\}}, \tag{3.9}$$

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

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

Algorithm 4: : Sequential multi-agent deployment

$\Omega_\omega = \bigcup \{x\}\ \forall x$
$\mathcal{E} \leftarrow$ current environment
$\hat{Q}, V_\omega, N_{\epsilon.s}, N_b \leftarrow$ loaded from previous trials

`Evaluate` (ϑ, X_ϑ):
    $val = 0$
    for $\overrightarrow{\tau}_k$ in $\vartheta.\overrightarrow{\tau}$:
        if $\hat{W}^{\omega}_{\overrightarrow{\tau}_k,X_\vartheta} < \vartheta.\overrightarrow{Pr^t}_k(t)\gamma^t r^\phi$:
            $val = val + \vartheta.\overrightarrow{Pr^t}_k(t)\gamma^t r^\phi - \hat{W}^{\omega}_{\overrightarrow{\tau}_k,X_\vartheta}$
    return val

`SequentialMultiAgentDeployment` $(\mathcal{E}, \mathcal{A}, \Omega_\omega)$
    $\Theta_\mathcal{A} = \emptyset$
    $X_\vartheta = \emptyset$
    for $\beta \in \mathcal{A}$
        $\Theta_\mathcal{A} = \Theta_\mathcal{A} \cup \{\Theta^\beta\}$
        $Z_\vartheta =$ `DDRMAP` $(\Omega_\omega, N_\epsilon, \mathcal{E}, \Theta^\mathcal{A}, \hat{Q}, V_\omega, N_{\epsilon.s}, N_b, \mathcal{M})$
        $\vartheta_\alpha = \underset{\vartheta \in Z_\vartheta}{\text{argmax}}$ `Evaluate`(ϑ, X_ϑ) $X_\vartheta = X_\vartheta \cup \{\vartheta_\alpha\}$
    return X_ϑ

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 $f_\vartheta : 2^{\Omega_\vartheta} \rightarrow \mathbb{R}$ to be a metric for measuring the performance of

`SequentialMultiAgentDeployment` as follows:

$$f_\vartheta(X_\vartheta) = \sum_{\forall \tau} \hat{W}^{\omega}_{\tau, X_\vartheta}.$$

This function is interpreted as the sum of discounted rewards given all a set of trajectories X_ϑ. The definition of T from Assumption 3.4.10 allows us to state the following result.

Lemma 3.4.11. *Under Assumptions 3.4.4-3.4.10, f_ϑ is a submodular, monotone set function.*

Proof. With (3.9) and the fact that rewards are non-negative (Assumption 3.4.4), 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

$$\mathcal{G}(X_\vartheta, \vartheta_\alpha) = \hat{W}^{\omega}_{\tau, X_\vartheta \cup \{\vartheta_\alpha\}} - \hat{W}^{\omega}_{\tau, X_\vartheta} = \max_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau, \{\vartheta\}} - \max_{\vartheta \in X_\vartheta} \hat{W}^{\omega}_{\tau, \{\vartheta\}}$$

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: $\vartheta_\alpha = \text{argmax}_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau, \{\vartheta\}} = \text{argmax}_{\vartheta \in Y_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau, \{\vartheta\}}$. This implies that $\hat{W}^{\omega}_{\tau, X_\vartheta \cup \{\vartheta_\alpha\}} = \hat{W}^{\omega}_{\tau, Y_\vartheta \cup \{\vartheta_\alpha\}}$.

Case 2: $\vartheta = \text{argmax}_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau, \{\vartheta\}} = \text{argmax}_{\vartheta \in Y_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau, \{\vartheta\}}$ such that

$\vartheta \in X_\vartheta$. This implies that $\hat{W}^{\omega}_{\tau, X_\vartheta \cup \{\vartheta_\alpha\}} = \hat{W}^{\omega}_{\tau, Y_\vartheta \cup \{\vartheta_\alpha\}}$.

Case 3: $\vartheta_\alpha = \text{argmax}_{\vartheta \in X_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau,\{\vartheta\}}$, and $\vartheta = \text{argmax}_{\vartheta \in Y_\vartheta \cup \{\vartheta_\alpha\}} \hat{W}^{\omega}_{\tau,\{\vartheta\}}$ 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 $\hat{W}^{\omega}_{\tau, Y_\vartheta} \geq \hat{W}^{\omega}_{\tau, X_\vartheta}$. 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 2.7.1 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 3.4.12. *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 3.4.13. *$\mathfrak{R}$ 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 3.4.2. We are now ready to characterize the lower bound performance of `SequentialMultiAgentDeployment` with respect to the optimal set of task trajectories. For convenience, let $X^*_{\vartheta} = \underset{X_{\vartheta} \in \Omega_{\vartheta}}{\operatorname{argmax}} f_{\vartheta}(X_{\vartheta})$ denote the optimal set of task trajectories.

Theorem 3.4.14. *Let X_{ϑ} be the trajectory set returned by `SequentialMultiAgentDeployment`. Under Assumptions 3.4.3-3.4.13, 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 2.7.1 for `SequentialMultiAgentDeployment`. From Lemma 3.4.11, 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 3.4.3 and 3.4.12. 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

$$f_{\vartheta}(X_{\vartheta} \cup \{\vartheta_{\alpha}\}) - f_{\vartheta}(X_{\vartheta}) =$$

$$\sum_{\forall \tau} \hat{W}^{\omega}_{\tau, X_\vartheta \cup \{\vartheta_\alpha\}} - \sum_{\forall \tau} \hat{W}^{\omega}_{\tau, X_\vartheta} =$$
$$\sum_{\forall \tau} (\hat{W}^{\omega}_{\tau, X_\vartheta \cup \{\vartheta_\alpha\}} - \hat{W}^{\omega}_{\tau, X_\vartheta}).$$

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

$$\sum_{\forall \tau \in \vartheta . \overrightarrow{\tau}} \max((\hat{W}^{\omega}_{\tau, \{\vartheta\}}, \hat{W}^{\omega}_{\tau, X_\vartheta}) - \hat{W}^{\omega}_{\tau, X_\vartheta}),$$

which is equivalent to $\sum_{\forall \tau}(\hat{W}^{\omega}_{X_\vartheta \cup \{\vartheta\}} - \hat{W}^{\omega}_{X_\vartheta})$. Since `SequentialMultiAgentDeployment` takes the trajectory that maximizes `Evaluate`, the result now follows from Theorem 2.7.1. • □

3.5 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.

3.5.1 Illustration of performance guarantees

Here we perform simulations that help validate the results of Section 3.4. We achieve this by determining X_ω from `SubEnvSearch`, which is an implementation of greedy maximization of submodular set functions, and the optimal region set X_ω^*, 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 $f_\omega(X_\omega)/f_\omega(X_\omega^*)$ in Figure 3.2. The empirical lower bound of $f_\omega(X_\omega)/f_\omega(X_\omega^*)$ is a little less than $1-e^{-1}$, consistent with the result, cf. Theorem 3.4.6, that the submodularity ratio of `SubEnvSearch` may not be 1. We believe that another factor for this empirical lower bound is that Assumption 3.4.3 is not fully satisfied in our experiments. In order to perform the simulation, we trained the agent on similar environments for 10 minutes. 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 sim-

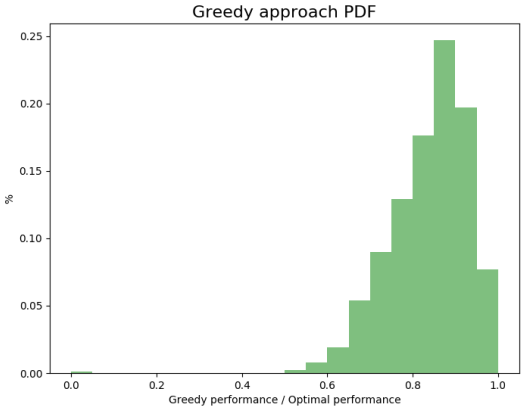


Figure 3.2: Probability distribution function of $f_\omega(X_\omega)/f_\omega(X^*_\omega)$.

ulated by randomizing the initial state of the environments. As shown in Figure 3.3, the lower bound on the performance with respect to the optimal set of trajectories is greater than $1 - e^{-1}$, as guaranteed by Theorem 3.4.14. 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.

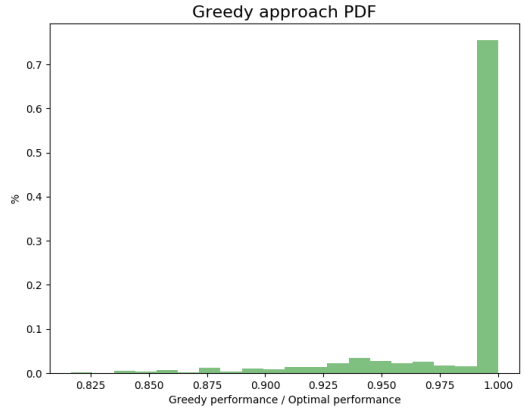


Figure 3.3: Probability distribution function of $f_\vartheta(X_\vartheta)/f_\vartheta(X^*_\vartheta)$.

3.5.2 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 3.4 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 3.4.6 helps give intuition to the immediate performance of DDRP-OO. The ACKTR simulation, displayed in Figure 3.5, performs well, but only after several million episodes of training, corresponding to approximately 2 hours 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

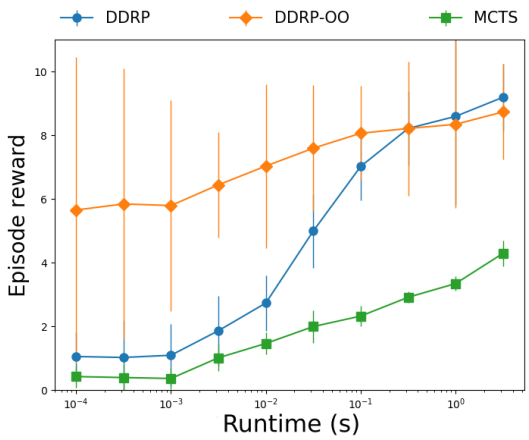


Figure 3.4: Performance of DDRP, DDRP-OO, and MCTS in randomized 2D environments with one objective type.

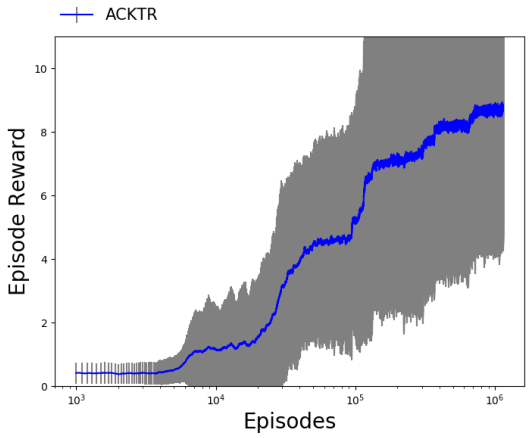


Figure 3.5: Performance of offline algorithm: ACKTR in randomized 2D environments with one objective type.

DDRP, MCTS, and ACKTR as shown in Figure 3.6. 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 seconds and ACKTR for 10000 trials, all with a discount factor of $\gamma = .99$. The best trajectories found by MCTS, DDRP, and ACKTR are shown in Figure 3.6 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 100000 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 Figure 3.7 and show the performance of ACKTR in a static instance of this environment in Figure 3.8.

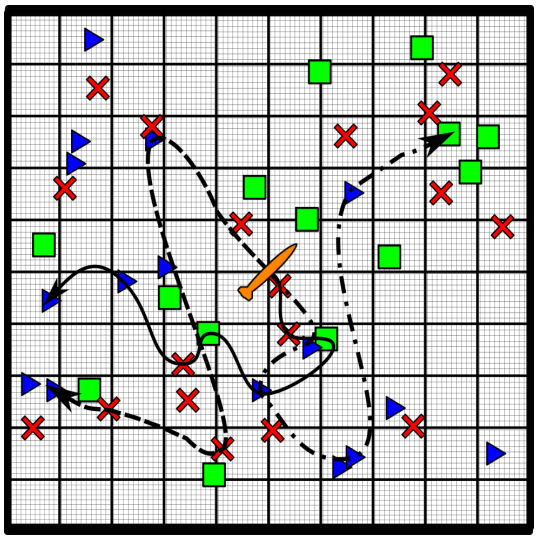


Figure 3.6: The trajectories generated from MCTS, DDRP, and ACKTR are shown with dashed, solid, and dash-dot lines respectively, in a 100×100 environment with 3 objectives. The squares, x's, and triangles represent waypoints of three objective types. The agent (represented by a torpedo) starts at the center of the environment.

3.5.3 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 Section 3.5.2, 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 Figure 3.9,

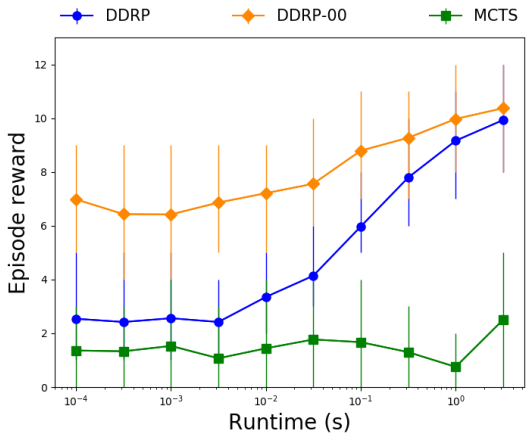


Figure 3.7: Performance of `DDRP`, `DDRP-OO`, and MCTS in randomized 2D environments with three objective types.

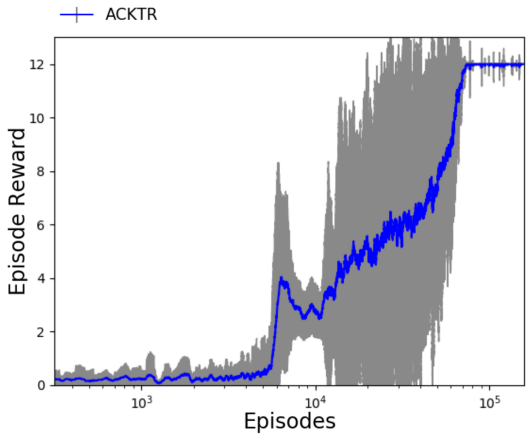


Figure 3.8: Performance of ACKTR in a static 2D environment with three objective types.

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.

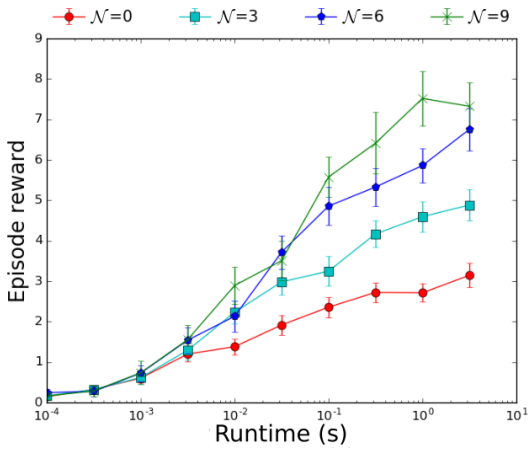


Figure 3.9: Performance of multi-agent deployment using `DDRMAP` in 2D environment.

3.6 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.

Acknowledgments

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.

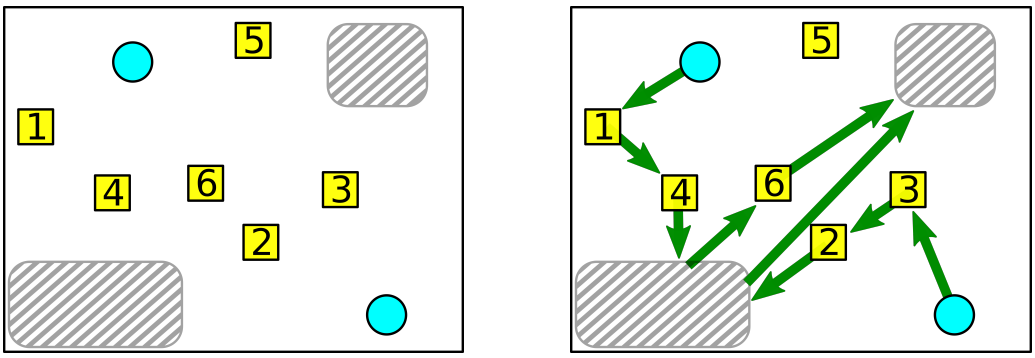
Chapter 3, is coauthored with Cortés, Jorge and Ouimet, Mike in full, is a reprint of the material as it appears in Autonomous Robotics 44 (3-4) 2020, 485-503, Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary author of this chapter.

Chapter 4

Cooperative dynamic domain reduction planning

UxVs are an outlet for the implementation of state-of-the-art algorithms that pertain to many fields of dynamic systems and machine learning. Recently, particular interest in the autonomous capability of these vehicles is growing. Characterizing multiple UxVs that interact with each other is difficult because of the joint number of possibilities that exist due to the joint state and action spaces. To approach this challenge, we propose `DDRP`, a hierarchical algorithm that takes slices of the environment and models them as a semi-Markov decision process. `DDRP` lacks a structure for agents requesting and assisting executing objectives that require more than one agent to complete. The motivation of this chapter is to extend the `DDRP` framework to allow agents to share their requests and to assist others if they deem it beneficial to the entire swarm.

Figure 4.1 provides an example application scenario of interest. In this example, agents are tasked with building structures. Agents are able to gather resources



(a) Initial conditions (b) Final trajectories

Figure 4.1: Example use of CDDRP. Agents are tasked with gathering resources and building structures at build sites. The symbols, ●, 1, ▨, represent agents, resources, and build sites respectively. In this specific case, the build site on the bottom left requires resources 1, 2, and 4, while the build site on the top right requires resources 3 and 6 to begin construction. Figure 4.1(a) shows the initial conditions and Figure 4.1(b) illustrates the trajectories resulting from the algorithm proposed in this chapter.

that are randomly scattered in the environment, but they are only able to carry two resources at any given time. The agents bring resources to building sites which require certain combination of resources for construction, a process that is expedited when agents cooperate.

4.1 Problem statement

Consider a set of agents $\mathcal{A}$ indexed by α. The agents seek to service a number of different objectives, whose *objective type* is indexed by b. A *sub-objective* $q = \langle w, b \rangle$ contains a waypoint $w \in \mathbb{R}^d$ and an objective type. An *objective* $o = \langle \mathcal{Q}, r \rangle$ consists of a set of sub-objectives $\mathcal{Q}$ and a reward $r \in \mathbb{R}$. We let $\mathcal{O}$ denote the set of all objectives. One possibility is to have objectives that require only one agent to be satisfied, as in [MOC17]. Instead, here we consider objectives with a sub-objective

set of cardinality $|\mathcal{Q}| > 0$. In this case, agents need to simultaneously be at specified waypoints and take actions in $q \in o.\mathcal{Q}$ in order to complete objective o. Agents use DDRP to generate a set of potential trajectories, termed $\mathcal{V}$, that they may take to service objectives in the environment. We strive to extend the capabilities of DDRP to include handling objectives that need two or more agents to complete. Given N agents, we determine a structure that allows the agents to share their trajectories and to distributively determine the joint trajectory that globally maximizes the sum of future discounted rewards.

4.2 Cooperative dynamic domain reduction planning

In this section we provide an overview DDRP and extend the framework to deal with objectives that require more than one agent. We organize this section as follows. First we review basic definitions from DDRP introduced in our previous work [MOC17]. As we discuss these definitions, we provide modifications to enable the agents to communicate desire for cooperation from other team members. Next, we present a high-level overview of algorithms used in DDRP. This sets the basis to introduce the multi-agent system where agents communicate and search for the joint optimal actions for deployment on large scale environments with cooperative objectives. We call the resulting framework *Cooperative dynamic domain reduction planning* (CDDRP).

4.2.1 Abstractions and definitions

We begin with some core definitions in `DDRP`. First we introduce abstracted regions and actions, the construction of sub-environments, and task trajectories. Then we give a high level overview of main algorithms in `DDRP`, `SubEnvSearch` and `TaskSearch`, and then finish with a new trajectory selection algorithm on the multi-agent level with some analysis.

Abstracted regions: A *region* is a convex set $x \subseteq \mathbb{R}^d$ such that the union of all regions are disjoint. The state of a region ω is an abstraction of the objectives that reside in ω. Let $\Phi_x : w \to x$ define the abstraction function that returns the region that q belongs to. We use this function to map where sub-objectives exist, i.e., $\Phi_x(q.w) = x$. Given a region x, let $\mathcal{Q}_x^b = \{q : \Phi_x(q.w) = x, q.b = b\}$ be the set of sub-objectives of objective type b that exist in it. We define the function $\Phi_o : \mathcal{Q}_x^b \to s_x^b$ to describe the abstracted state of the corresponding type of objective in the region. Define the regional state to be $s_\omega = (s_x^1, s_x^2, \ldots)$.

Abstracted tasks: In [MOC17], a task is a tuple $\tau = \langle s_{\omega_i}^{b'}, s_{\omega_i}^b, \omega_i, \omega_j, b \rangle$, where ω_i is the region that the agent is executing sub-objective of objective type b in, ω_j is the next region that the agent plans to travel, $s_{\omega_i}^b$ is the prior state of ω_i, and $s_{\omega_i}^{b'}$ is the post state of ω_i. Here, we augment the notion of task to include the concept of time abstraction. Mapping the time to an interval allows the agents to communicate approximate times to complete coordinated tasks by. If the length of the time intervals is too small, then the number of possible joint actions increases and the problem may become intractable. On the other hand, if the length of the time intervals is too big then the execution time of coordinated tasks become less precise. Let the convex set $\varsigma \subseteq \mathbb{R}$ specify a *time interval*. We specify a *sub-task*, $\mu = \langle \omega_i, b, \varsigma \rangle$,

to be a tuple that contains a region ω_i that the agent acts in, a objective type b, and a time interval ς. The modified definition of a task is now $\tau = \langle \mu, s^b_{\omega_i}, s^{b'}_{\omega_i}, \omega_j \rangle$. This modification allows agents to communicate the bare minimum information that is necessary for others to know when they are attempting a coordinated objective. We denote by $\mathcal{T}$ the set of all tasks.

Sub-environments: The DDRP framework takes the environment and generates *sub-environments* ϵ composed of a sequence of abstracted regions and a state encoding proximity and regional states of those regions. We extend the definition of sub-environment to include requirements that the agent agrees to satisfy. We do this by incorporating the requirements into the state of the sub-environment. Let $\overrightarrow{\omega}$ be a finite sequence of regions in the environment (e.g., $[\omega_2, \omega_1, \omega_3]$). We determine a state of the sub-environment given $\overrightarrow{\omega}$. To do this, we need to determine if regions are repeated in $\overrightarrow{\omega}$. Let $\xi(k, \overrightarrow{\omega})$, return the first index h of $\overrightarrow{\omega}$ such that $\overrightarrow{\omega}_h = \overrightarrow{\omega}_k$. Another necessary component is the time that it takes for the agent to travel between regions. We use $d(\omega_i \omega_j) : \omega_i, \omega_j \to \mathbb{Z}$ to designate an abstracted amount of time it takes for an agent to move from ω_i to ω_j, or ∞ if no path exists. We create sub-environments with the constraint that agents may need to satisfy some cooperative tasks. Let the set $\mathcal{U} = \{\mu_1, \mu_2, \ldots\}$ be a set of subtasks that the agent agreed to partake in the sub-environment. We define the *sub-environment state* with the addition of requirements as

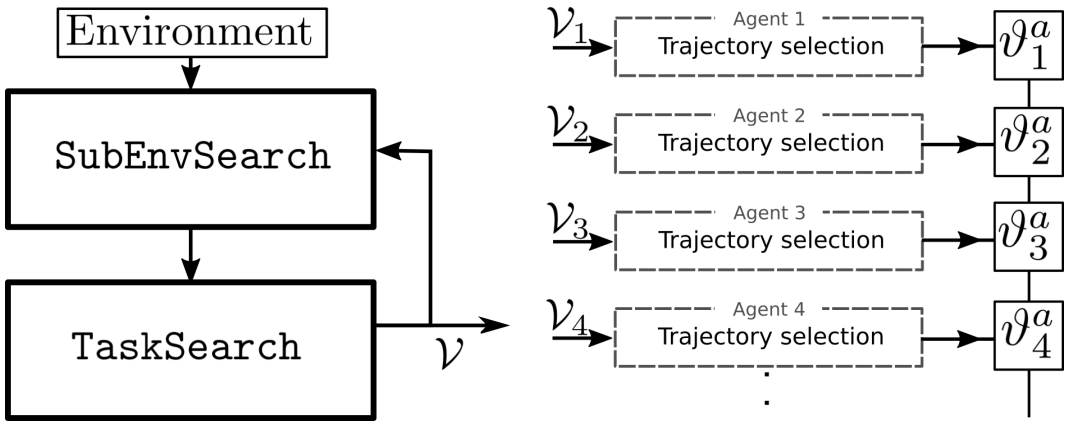
$$s = [s_{\overrightarrow{\omega}_1}, s_{\overrightarrow{\omega}_2}, \ldots] \times [\xi(1, \overrightarrow{\omega}), \xi(2, \overrightarrow{\omega}), \ldots] \times [d(\overrightarrow{\omega}_1, \overrightarrow{\omega}_2), d(\overrightarrow{\omega}_2, \overrightarrow{\omega}_3), \ldots] \times \{\mu_1, \mu_2, \ldots\}. \tag{4.1}$$

We denote by S^ϵ the set of all possible sub-environment states. A sub-environment is $\epsilon = \langle \overrightarrow{\omega}, s, \mathcal{U} \rangle$.

When an agent performs a task in the sub-environment, it expects the action to take multiple time steps to complete, the environment to change states, and to receive some reward. Let Pr^s and Pr^t be the probability distributions for state transition and time of completion for executing a task in a sub-environment, respectively. Also, let r^ϵ designate the expected reward that an agent receives for choosing a task given the state of a sub-environment. Finally, we can represent the process of executing tasks in sub-environments as the *sub-environment SMDP*, $\mathcal{M} = \langle S^\epsilon, \mathcal{T}, Pr^s, r^\epsilon, Pr^t \rangle$. The goal of the agent is to determine a policy $\pi : \epsilon.s \rightarrow \tau$ that yields the greatest rewards in $\mathcal{M}$. The state value under policy π is given by

$$V^\pi_{\epsilon.s} = r^\epsilon + \sum_{t^\epsilon \in \mathbb{R}} P^{t^\epsilon} \gamma^{t^\epsilon} \sum_{\epsilon.s \in S^\epsilon} P^s V^\pi_{\epsilon.s'} \text{ s.t. } 0 \leq \gamma < 1. \tag{4.2}$$

Task trajectories: Agents cooperate with others by sharing their current and prospective plans for the receding time horizon. In DDRP, we call this information a *task trajectory*. This is defined as $\vartheta = \langle [\tau_1, \tau_2, \ldots], [Pr^1, Pr^2, \ldots] \rangle$, where tasks in $[\tau_1, \tau_2, \ldots]$ are executed in order and the completion time of τ_1 is according to the probability density function $Pr^1(t)$, etc. Task trajectories are created with respect to some sub-environment ϵ and are constrained so that a task in $\vartheta.\tau_p.\omega_j$ must be $\vartheta.\tau_{p+1}.\omega_i$, making it so that the region that agents travel to next is always the active region of the next task to complete. Here, we redefine the concept of task trajectory to carry information about what cooperative tasks the agent has. Given the cooperative tasks in a trajectory, the agent may have some cooperative tasks that it plans on executing with others, and some cooperative tasks that it might not have found



(a) Algorithms of DDRP (b) CooperativeTrajectorySearch

Figure 4.2: Workflow of DDRP and CooperativeTrajectorySearch. DDRP in Figure ?? is a hierarchical algorithm that dynamically creates sub-environments with sub-environment search. Sub-environments are modeled as a semi-Markov decision process where the agent uses TaskSearch to find tasks which yield the greatest expected discounted reward. CooperativeTrajectorySearch is a process that is run in parallel with DDRP. With respect to the simulated annealing process, agents asynchronously choose active trajectories ϑ_α^a from a set of trajectories $\mathcal{V}_\alpha$ found in DDRP as shown in Figure ??. The result is a list of active trajectories, $\rho = [\vartheta_1^a, \vartheta_2^a, \ldots]$.

partners for execution. The agent puts all subtasks in the set $\mathcal{U}$. A task trajectory is then defined as $\vartheta = \langle[\tau_1, \tau_2, \ldots], [Pr^1, Pr^2, \ldots], \mathcal{U}\rangle$.

4.2.2 DDRP algorithms and task generation for communication

DDRP is a hierarchical framework which has an algorithm called SubEnvSearch that generates subsets of the entire environment. The sub-environment that is created is modeled as an SMDP, and TaskSearch is used to optimally find a policy for it. In this section, we give a brief description of the algorithmic components SubEnvSearch and TaskSearch, and introduce some necessary modifications for CDDRP. DDRP organizes SubEnvSearch and TaskSearch into a hierarchical structure as shown in Figure ??.

Sub-environment search: `SubEnvSearch` uses the environment as an input and creates a sub-environment. Using the set of all regions in the environment, we add one region at a time to a 'sub-environment set' until there are N_ϵ regions. As the sub-environment set is being generated, its value (determined by `TaskSearch`) is evaluated. The agent uses this value to add regions that locally maximize the value of the sub-environment.

We extend the sub-environment search process to now include cooperative tasks. To do this, first we run `SubEnvSearch` to get a sub-environment from the environment. If a cooperative sub-objective q exist in the sub-environment, then with some probability we include q to the sub-environment sub-task set $\epsilon.\mathcal{U}$ as a requirement for the agent to complete. These cooperative sub-objectives must be compatible with each other. For example, an agent is not able to create a sub-environment with two cooperative sub-objectives at the same time interval. After this, the agent updates the sub-environment state accordingly and sends the created SMDP to `TaskSearch`.

Task search: Given a sub-environment ϵ generated from `SubEnvSearch`, the purpose of `TaskSearch` is to determine the value of taking a task τ. This is done using upper confidence bound tree search (UCT) [KS06], which is a Monte-Carlo tree search (MCTS) that uses the upper confidence bound of action values to explore the SMDP. The agent takes the sub-environment SMDP and runs `TaskSearch` for many iterations, where the values of taking a task given the state of the sub-environment converges to a real number. When choosing which task to take, the agent chooses the task that maximizes the action value given the state of the sub-environment.

We extend the task search process under the constraint that it must satisfy the

cooperative subtasks in the sub-environment. To do this, given the sub-environment from `SubEnvSearch`, the agent must obey the constraints of cooperative subtasks in the sub-environment. At each task selection, the agent chooses from a set of tasks such that it is still feasible to visit regions in $\epsilon.\mathcal{U}$ at the required times. The agent determines the action value for choosing tasks given the sub-environment state and satisfying the constraints given to it and returns a task trajectory. This task trajectory also contains all the cooperative tasks that it was constrained to satisfy as unpaired cooperative tasks.

4.2.3 Joint `DDRP` via simulated annealing

In this section we introduce the main contribution of this chapter, aside from the modifications to `DDRP` described above. We specify a value for the joint set of trajectories by determining which objectives can be completed given the planned sub-tasks of all the agents. Then we introduce the algorithm in which agents use simulated annealing to search for the best set of joint trajectories.

Agents execute `DDRP` yielding a set of trajectories $\mathcal{V}$. The trajectory that an agent currently plans to execute, called the *active trajectory*, is denoted by ϑ^a. Agents share their active trajectories so that they know which cooperative objectives they can collaborate on. The active trajectories are ordered in a list with respect to some arbitrary agent ordering to form the *active trajectory list* $\rho = [\vartheta_1^a, \vartheta_2^a, \ldots]$. In doing so, agents have access to the subtasks of others. The set of all current subtasks is the *collective subtask set*, denoted $\mathcal{U}^{\mathcal{A}} = \{\mu : \mu \in \vartheta.\mathcal{U}, \ \forall \ \vartheta \in \rho\}$. We also create a set of subtasks that are planned to be executed in a given time interval, $\mathcal{U}_\varsigma^{\mathcal{A}} = \{\mu : \mu \in \mathcal{U}^{\mathcal{A}}, \ \mu.\varsigma = \varsigma\}$. For the remainder of the chapter, we assume that agents

have access to $\mathcal{U}^{\mathcal{A}}$ and $\mathcal{U}_{\varsigma}^{\mathcal{A}}$ for all time intervals ς. We say that o is satisfied if all sub-objectives in o exist in a subtask set at ς such that $o.\mathcal{U} \subseteq \mathcal{U}_{\varsigma}^{\mathcal{A}}$. With knowledge about $\mathcal{U}_{\varsigma}^{\mathcal{A}}$ for all time intervals ς, agents are able to determine the minimal time interval in which each objective is expected to be completed. Let the *expected time of completion* for an objective be defined as

$$\varsigma_o(\mathcal{U}^{\mathcal{A}}, \rho) = \underset{\varsigma}{\operatorname{argmin}} \int_{\varsigma} \gamma^t dt \quad \text{s.t. } o.\mathcal{U} \subseteq \mathcal{U}_{\varsigma}^{\mathcal{A}}. \tag{4.3}$$

or $\varsigma_o = \infty$ if there are no time intervals for which $o.\mathcal{U} \subseteq \mathcal{U}_{\varsigma}^{\mathcal{A}}$. Next, we introduce the *joint task trajectory value* as

$$V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho) = \sum_{o \in \mathcal{O}} \int_{\varsigma_o(\mathcal{U}^{\mathcal{A}}, \rho)} \gamma^t \, o.r \, dt, \tag{4.4}$$

which corresponds to the cumulative discounted reward of all the agents in the swarm given ρ.

We are now ready to introduce `CooperativeTrajectorySearch` (cf. Algorithm 5). This strategy takes as input a set of agents, their respective task trajectory sets, and the set of all objectives. The agents initially choose active trajectories that form the active trajectory list and the temperature is initialized to ∞. We use the notation $\rho|\vartheta_\alpha$ to indicate the resulting active trajectory list that occurs when agent α chooses a new active trajectory $\vartheta_\alpha = \vartheta_\alpha^a$. This operation simply changes the α-index element of ρ to be ϑ_α. An agent at random then chooses to select a trajectory from its trajectory set with probability distribution Pr^{ϑ}. This distribution follows one of

two schemes. The first is called a *flat* scheme, where agent α chooses a trajectory ϑ with respect to the number of trajectories in the set as follows:

$$Pr^{\vartheta}(\vartheta)^{\dagger} = \frac{1}{|\mathcal{V}|}. \tag{4.5}$$

The next method of selecting a trajectory that we explore is called a *weighted* scheme. In this method, agent α chooses a trajectory with probability with respect to the local joint task trajectory values as follows:

$$Pr^{\vartheta}(\vartheta_{\alpha})^{\ddagger} = \frac{V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho|\vartheta_{\alpha})}{\sum\limits_{\vartheta_{\alpha}^{i} \in \mathcal{V}_{\alpha}} V^{\mathcal{A}}(, \mathcal{U}^{\mathcal{A}}, \rho|\vartheta_{\alpha}^{i})} \tag{4.6}$$

Once the agent has chosen ϑ_{α} from the distribution, it evaluates the marginal gain of the trajectory given by $V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho|\vartheta_{\alpha}) - V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho)$. If the marginal gain is positive, or if the simulated annealing acceptance given by line 8 is satisfied, then the agent accepts the new trajectory and notifies all other agents of the change. The temperature decreases by the cooling schedule in (2.1), and the above process is repeated. `CooperativeTrajectorySearch` is illustrated in Figure **??**. `DDRP` and `CooperativeTrajectorySearch` are run in parallel, where the trajectory set $\mathcal{V}^{\mathcal{A}}$ for agents is constantly being updated as trajectories are discovered in `DDRP`. We call the resulting framework `CDDRP`.

This process can be characterized by a Markov chain. The Markov chain is $\mathbb{C} = \langle \mathcal{P}, Pr^{\rho}, V^{\mathcal{A}} \rangle$ which contains the set of possible active trajectory lists $\mathcal{P}$, the probability distribution Pr^{ρ} that encodes the chance of hopping from one active trajectory length to another, and $V^{\mathcal{A}}$. Figure 4.3 gives an example of the Markov chain that characterizes our system. For the following analysis we define $\mathcal{P}^{*} = \{\rho :$

Algorithm 5: `CooperativeTrajectorySearch`

```
1  CooperativeTrajectorySearch (V, A):
2      ρ = [ϑ_1^0, ϑ_2^0, ...]
3      T = ∞
4      for k ∈ [2, 3, ...]:
5          α ← RandomSelection(A)
6          ϑ ← sample from Pr^ϑ
7          if V^A(ρ|ϑ, Q) > V^A(ρ, Q)
8          or Random(0,1) < e^((V^A(ρ|ϑ,Q) − V^A(ρ,Q))/T)
9              ρ = ρ|ϑ
10         T = c / log(k)
```

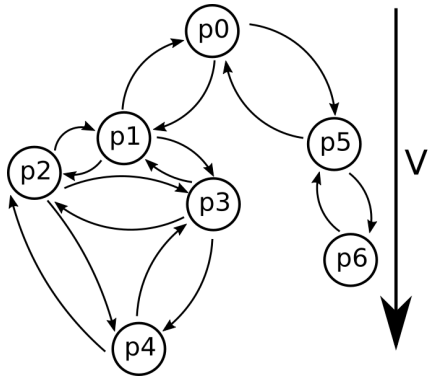


Figure 4.3: (Left): Pseudocode description of `CooperativeTrajectorySearch`. (Right): A graph generated from the Markov chain $\langle \rho, Pr, V \rangle$. Nodes represent configurations of task trajectory lists and edges represent positive state transition probabilities $Pr(\rho, \rho')$ between states ρ and ρ'. The graph is ordered such that the y-component of a node increases as node value decreases.

$\rho \in \mathcal{P}$ s.t. $\text{argmax}_{\rho \in \mathcal{P}}\ V^{\mathcal{A}}(\mathcal{Q}, \rho)\}$ as the set of active trajectory lists that maximize $V^{\mathcal{A}}$ (more than one may exist).

4.3 Performance of selection schemes: 'flat' vs. 'weighted'

In this section we show experimental results for two trajectory selection schemes and show relative performance differences in the same environments. We perform 3 simulations with the parameters shown in Table 6.1.

Table 4.1: Parameters used in the simulations.

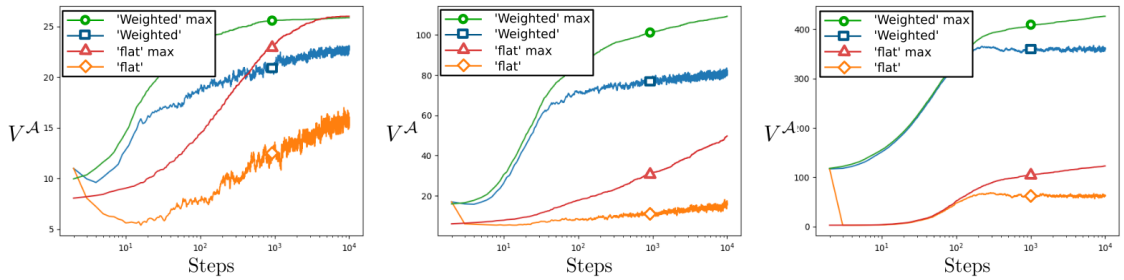
simulation	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V}\rvert$	N_ϵ	c	Δt	total time
1	5	5	5	20	1×10^{-4}	~1s
2	10	10	5	100	2×10^{-4}	~2s
3	100	100	3	800	2×10^{-3}	~20s

In these simulations, we assume that the agents have run `DDRP` for some time to generate $\mathcal{V}$ and then use `CooperativeTrajectorySearch` to find ρ. We create the

simulation set up with 100 objectives, 50 of which require $2-5$ agents to complete. Rewards for satisfying an objective are randomly picked from a range that scales with the number of agents required to satisfy the objective (which incentivizes collaboration). Varying parameters in the three simulations are number of agents, number of trajectories in their respective trajectory sets, length of the sub-environments they create, and approximated c. The average time per step, denoted by Δt in Table 6.1, varies because the agent needs to calculate Pr^{ϑ}. Each simulation is run 100 times for both the 'weighted' and 'flat' schemes. We illustrate the results for each simulation in Figure 4.4. The average time for completion of experiments for cases 1,2 and 3 were 1, 2, and 20 seconds, respectively. In all cases, the maximal value found by the 'weighted' scheme approaches the optimal joint trajectory value sooner than the 'flat' approach. We find that the 'flat' scheme struggles to find joint trajectories with values comparable to the 'weighted' scheme when used for large action spaces. In cases with low action space, such as simulation 1, the maximal value determined from the 'flat' scheme was able to approach the optimal trajectory value. The average value of both schemes is lower than the maximal value found due to the algorithms propensity to escape local maximums.

4.4 Conclusions

We have extended the dynamic domain reduction framework for multi-agent planning over long time horizons and a variety of objectives to scenarios where some objectives require two or more agents to complete. In order to do this, we have made modified `DDRP` to allow for necessary information to be shared amongst agents. These include the inclusion of time abstractions and cooperative objectives, and modifica-



(a) Simulation 1 (b) Simulation 2 (c) Simulation 3

Figure 4.4: Results for simulations. Here the joint trajectory discounted reward is shown on the y-axis and the number of steps in log-scale base 10 is shown on the x-axis. We plot the average discounted reward of the current states in each time step of the simulations which are labeled 'weighted' and 'flat'. The lines that correspond to 'weighted' max and 'flat' max indicate the max value that was found in each trial by time step k, averaged over all trials.

tions to both trajectories and sub-environments. Building on this framework, we have designed an algorithm based on simulated annealing that allows agents to expedite the exploration of solutions by increasing the chance that they choose tasks that help one another. This is important in the distributed setting in order to reduce the communication needed to find a good solution. Our analysis of the algorithm has shown that, given enough time, the active trajectory list converges in probability to an optimal active trajectory. We do this by showing that the Markov chain that characterizes our multi-agent process satisfies weak reversibility and strong irreducibility properties, and by using a logarithmic cooling schedule. Simulations compare our algorithm with a weighted approach versus our algorithm with a flat approach when it comes to agent select trajectories. In the future, we plan to develop efficient methods for the agents to come up with their trajectories during `DDRP`, examine the trade-offs in designing how the simulated annealing process can influence the search of trajectories in `DDRP`, and introduce asynchronous implementations to broaden the utility

for real-world scenarios.

Chapter 4, in full, is a reprint of the material as it appears in Distributed Autonomouos Robotic Systems: The 14th International Symposium, Springer Proceedings in Advanced Robotics, vol. 9, pp.499-512, Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary author of this chapter.

Chapter 5

Temporal sampling schemes for receding horizon cooperative task planning

Interest in the autonomous capability of UxVs and their applications to real-world scenarios is increasing. Problems of interest include mapping of unknown regions, surveillance, and pursuit of other vehicles. The use of UxVs in these scenarios is motivated by keeping people away from danger and for improved efficiency/performance of the task. The ability to complete tasks to satisfy these objectives are largely determined by the vehicle's capabilities for autonomy and planning. When considering multiple vehicles, task planning can become infeasible to do centralized because the size of the joint action space of the agents grows exponentially as the number of agents increase. In this paper, we consider distributed task planning by modeling the problem as a potential game, where agents utilize stochastic policies for selecting plans iteratively over a moving time horizon. One common problem

that occurs when agents choose their plans in a stochastic manner is that they may change their action right before execution, leading to the loss of some other agent's utility. To address this, we propose a sampling scheme which discourages an agent from switching actions in the near future and encourages agents to plan for the long term.

5.1 Problem statement

Consider a scenario with a team of N agents distributively determine a schedule of future actions. Let the agents plan over a finite time horizon where they must select actions for time steps $t = [1, \ldots, T]$. Agents, indexed by α, have states represented by s_α. They must select actions that alter the *environment state* $\mathrm{s} = (\mathrm{s}_1, \ldots, \mathrm{s}_N, s^e)$, where states s_1 through s_N represents states of agents 1 through N and s_e represents the state of the environment. Agent α is able to select actions according to their individual action space $a_\alpha^t \in A_\alpha^t$ available to it at time t. Each agent develops an *action schedule* that plans to execute during the finite time horizon denoted $\bar{a}_\alpha = [a_\alpha^1, \ldots, a_\alpha^T] \in \bar{A}_\alpha$. The *joint action plan*, $\bar{a} = \{\bar{a}_1, \ldots, \bar{a}_n\}$ is the set of all agents action schedules. For the sake of notation, let $\bar{a}_{-\alpha} = \bar{a} \setminus \bar{a}_\alpha$. We also use $\bar{a}^t = \{a_1^t, \ldots, a_N^t\}$ to represent the set of actions that agents plan to select at time t. Agents receive a reward for taking a joint action at particular states, $R^t : s^t \times \bar{a}^t \to \mathbb{R}$, while $R^T : s^T \to \mathbb{R}$ represents a terminal reward the agents get at the final time step in the finite time horizon. The sum of future rewards gathered by the agents is

represented by

$$\Phi(\bar{a}) = \sum_{t=1}^{T-1} R_{s^t,\bar{a}^t} + R_{s^T}. \tag{5.1}$$

The goal of the agents is to maximize their 'wonderful life utility' as defined in (**??**).

5.2 Task scheduling with recycled solutions

We study three components to the evolution of events that occur during deployment in our algorithms. First, agents execute an action. Then, actions are shifted to represent the step, i.e., $\bar{\mathrm{a}}_\alpha^t$ becomes $\bar{\mathrm{a}}_\alpha^{t-1}$ for all $t \in (2, \ldots, T)$ and $\alpha \in \mathcal{A}$. The shifted action schedules will become the initial solution for agents in the next step of the finite time horizon. However, these action schedules will be missing $\bar{\mathrm{a}}_\alpha^T$, so the third component we consider is the generation of that action. We refer to this process as *recycling solutions*.

Here, we propose a deployment strategy, outlined in Algorithm 6, called *horizon shift planning* (`HSP`). In `HSP`, between lines 3-18 agents sample action schedules with respect to the sampling matrix P^g (specific schemes to determin this matrix are introduced later in Section 5.3, giving rise to different instantiations of `HSP`). With probability respective to P^a, the agent accepts the candidate action schedule. The agent then broadcasts the change in action schedule with some probability. It is important to note that after agents take actions, and the finite time horizon shifts, agents recycle their chosen action schedules $\bar{a}$ to be used as an initial action schedule in lines 3-18 for the next time step.

Theorem 5.2.1. *(Reaching pure Nash equilibria through `HSA`): Consider the horizon*

Algorithm 6: Horizon shift planning

```
HSP(s, 𝒜)
    while deploying is true:
        for k = 0 → K; Agents α ∈ 𝒜 do in parallel:
            ā_α = HSA(ā_α, k)
        executeActions(ā = [ā_1, ..., ā_|𝒜|])
        for t = 0 → (T − 1):
            s^t = s^{t+1}
            ā^t = ā^{t+1}

Let i denote the current action schedule ā_α
Let j denote the candidate action schedule ā'_α
HSA(i := ā_α, k)
    Select j with probability P^g_ij
    Δ = u_j − u_i
    if Δ > 0 or rand()> e^{Δ/T_k}:
        i ← j
        if rand()> τ:
            updateAgents(i)
    return i := ā_α
```

shift annealing, ***HSA****, outlined in Algorithm 6 using the linear cooling schedule* (**??**)*. Agents that use 'wonderful life utility' with potential function given by* (5.1) *will reach a pure Nash equilibrium with probability* 1 *as* k *goes to infinity.*

Proof. We use the fact that exact potential games exhibit the *finite improvement property*, meaning that every improvement path, $\bar{a}^1, \bar{a}^2, \ldots$ such that $\Phi(\bar{a}^k) > \Phi(\bar{a})^{k-1}$, is finite in length. Because of this property, a Nash equilibrium exists. From (**??**), for $k > c$ the temperature becomes 0 and agents will only choose action schedules with positive utility u. Agents then select actions that are along some finite improvement path until they reach a pure Nash equilibrium. □

So long as $P^g_{\bar{a},\bar{a}} > 0$ for all $\bar{a}$, $\bar{a}$, the temperature can be set to 0 initially and the locally optimal solution can be found given large enough t. The inclusion of a

temperature schedule is motivated by the inherent temporal structure of the solutions. Action schedules likely have similar utilities when their actions are similar. Using a cooling schedule means that extra pathways in the Markov chain to the optimal action schedule will exist. The motivation is that by increasing the number of pathways to the optimal solution, we will increase the probability to find that solution and it will be found more quickly with a cooling schedule than without.

5.3 Sampling schemes for task scheduling

In this section we design sampling schemes for task scheduling. In general, the probability P^g is not viewed as a design choice in annealing approaches, and is often a result of existing state transition probabilities in a Markov chain. In our case, since agents can choose to take an arbitrary action schedule, we design P^g to make sampling more efficient with respect to the finite time horizon and to confront the issue of agents breaking promises in the near future, which may cause other agents to lose utility.

5.3.1 Sampling matrix structure

We choose to structure and position the indices of P^g with the following conventions. P^g is a block matrix with $|A| \times |A|$ blocks. A block $P^g_{a^1_\alpha, a^{1'}_\alpha} \in P^g$ is a matrix of probabilities of transition for action schedules with first action a^1 to action schedules with first action $a^{1'}$. $P^g_{a^1, a^{1'}}$ is also a block matrix with $|A| \times |A|$ blocks. A block $P^g_{a^2, a^{2'}|a^1, a^{1'}}$ is a matrix of probabilities of transition for action schedules first and second action, a^1 and a^2, to action schedules with first and second action, a^2 and $a^{2'}$.

This organization of P^g has the following properties

- Diagonal elements are transitions to the same action schedule. Diagonal blocks are transitions to the same action taken at that time step.
- P^g is row-stochastic
- The probability that an agent chooses an element in a block matrix is the summation of probabilities of elements in that block.

5.3.2 Geometric sampling

Here we introduce the *geometric sampling* scheme. The idea is inspired by simulated annealing, however, we modify the probability that actions are sampled based on their position in the finite time horizon. In particular, the generation probability for a sampled action schedule geometrically reduced as follows

$$\mathrm{P}_{ij}^{g\dagger} = \frac{\gamma^{T-t_{ij}}}{|\bar{A}_\alpha| \sum_{p \neq j} \frac{\gamma^{T-t_{i,p}}}{|\bar{A}_\alpha|}}, \tag{5.2}$$

where $\gamma \in (0, 1)$ and t_{ij} is the minimum time where actions deviate in the two action schedules, i.e., $t_{ij} = \min\{t \in \mathbb{R} \mid i^t \neq j^t\}$.

Figure 5.1 illustrates P^g given (5.2). In this scheme we force the agents to sample action schedules that will change actions in the near future less often.

Remark 5.3.1. *(Keeping promises with geometrically reduced sampling):* Here we address the issue of agents 'breaking promises' with other agents by influencing the agents to sample action schedules with actions that deviate in the near future, less

often. It should be noted that the agents are not decentivized from working together as searching for cooperative plans is prioritized for future actions. •

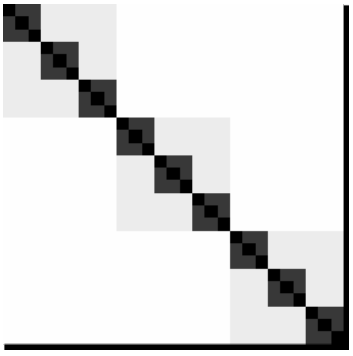


Figure 5.1: A matrix that illustrates the probability of action schedule generation. In this case, the agent has an action space $\bar{A}^t$ of 3 and is planning up to $T = 3$ time steps ahead. In the image, the intensity of the pixel i, j corresponds to the probability of the agent sampling $j := \bar{a}_{\alpha,j}$ when its last solution is $i := \bar{a}_{\alpha,i}$. Light and dark pixels correspond to high and low probabilities of generation, respectively.

Agents recycle their solution from iterations in the time horizon. Agents do this by the following *horizon shift* operation for all elements in P^g. We define the 'horizon shift' operations as $\sigma(P) \to P'$, for any square row-stochastic matrix P such that

$$\begin{aligned} P'_{ij} &= \gamma P_{ij} && \forall j \neq i \\ P'_{ii} &= P_{ii} + (1-\gamma)\sum_{j \neq i} P_{ij} && \forall i \end{aligned} \tag{5.3}$$

where $\gamma \in (0, 1)$. When this operation is applied to $\mathrm{P}^{g\dagger}$ the probability of all transitions between differing action schedules in P^{tr} is reduced by γ. We show how the eigenvalues of P^{tr} evolve when (5.3) is applied to P^g.

Theorem 5.3.2. *Let* $\mathrm{P}^{g\prime}$ *be the output of the 'horizon shift' operation on* P^g *as described in* (5.3)*, and* P^a *be any acceptance matrix. Let the initial transition matrix*

P^{tr} *be determined as shown in* (**??**) *with* P^g *and the modified transition matrix* $\mathrm{P}^{tr\prime}$ *be determined with* $\mathrm{P}^{g\prime}$. *Also, let* $\lambda_1, \lambda_2, \ldots$ *and* $\lambda'_1, \lambda'_2, \ldots$ *be the eigenvalues of* P^{tr} *and* $\mathrm{P}^{tr\prime}$, *respectively, such that*

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{|\bar{A}_\alpha|}$$
$$\lambda'_1 \geq \lambda'_2 \geq \ldots \geq \lambda'_{|\bar{A}_\alpha|},$$

Then,

$$\lambda'_k = 1 + \gamma\lambda_k - \gamma.$$

Furthermore, the row-eigenvectors are invariant under the operation.

Proof. We first show that the application of the 'horizon shift' operation can be applied to P^g or P^{tr} to get the same resultant transition matrix $\mathrm{P}^{tr\prime}$

$$\mathrm{P}^{tr\prime} = \sigma(P^g)\mathrm{P}^a = \sigma(\mathrm{P}^{tr}). \tag{5.4}$$

For all $\mathrm{P}^{tr\prime}_{ij}$ such that $j \neq i$

$$\begin{aligned}\mathrm{P}^{tr\prime}_{ij} &= (\gamma P^g_{ij})\mathrm{P}^a_{ij} \\ &= \gamma \mathrm{P}^{tr}_{ij}.\end{aligned}$$

Thus, all off-diagonal elements are equivalent. Then for all diagonal elements $\mathrm{P}^{tr\prime}_{ii}$

from (**??**)

$$
\begin{aligned}
\mathrm{P}^{tr\prime}_{ii} &= \mathrm{P}^{g\prime}_{ii} + \sum_{j\neq i} \mathrm{P}^{g\prime}_{ij}(1-\mathrm{P}^{a}_{ij}) \\
&= (P^{g}_{ii} + (1-\gamma)\sum_{j\neq i} P^{g}_{ij}) + \gamma \sum_{j\neq i} P^{g}_{ij}(1-\mathrm{P}^{a}_{ij}) \\
&= P^{g}_{ii} + \sum_{j\neq i} P^{g}_{ij} - \gamma \sum_{j\neq i} P^{g}_{ij}\mathrm{P}^{a}_{ij} \\
&= P^{g}_{ii} + \sum_{j\neq i} P^{g}_{ij} - \gamma \sum_{j\neq i} P^{g}_{ij}\mathrm{P}^{a}_{ij} + \sum_{j\neq i} P^{g}_{ij}\mathrm{P}^{a}_{ij} - \sum_{j\neq i} P^{g}_{ij}\mathrm{P}^{a}_{ij} \\
&= P^{g}_{ii} + \sum_{j\neq i} P^{g}_{ij}(1-\mathrm{P}^{a}_{ij}) + (1-\gamma)\sum_{j\neq i} P^{g}_{ij}\mathrm{P}^{a}_{ij} \\
&= \mathrm{P}^{tr}_{ii} + (1-\gamma)\sum_{ij} \mathrm{P}^{tr}_{ij}
\end{aligned}
$$

This means that we can apply (5.3) to either P^g or directly to P^{tr} to get $\mathrm{P}^{tr\prime}$. Now we prove that the stationary distribution is invariant when the 'horizon shift' operation is applied directly to P^{tr}, i.e.

$$
v\mathrm{P}^{tr\prime} = v\mathrm{P}^{tr} = vP = v = [v_1, \ldots, v_N]
$$

Consider the element-wise calculation for v_i.

$$
(v\mathrm{P}^{tr\prime})_i = \sum_{\forall j} v_j \mathrm{P}^{tr\prime}_{ji} = v_i \mathrm{P}^{tr\prime}_{ii} + \sum_{i\neq j} v_j \mathrm{P}^{tr\prime}_{ji}
$$

We apply (5.3) to the right-hand side to obtain

$$
(v_i \mathrm{P}^{tr}_{ii} + (1-\gamma)\sum_{j\neq i} v_i \mathrm{P}^{tr}_{ij}) + \gamma \sum_{j\neq i} v_j \mathrm{P}^{tr}_{ji}
$$

$$= (1-\gamma)v_i \mathrm{P}^{tr}_{ii} + \gamma v_i \mathrm{P}^{tr}_{ii} + (1-\gamma)\sum_{j\neq i} v_i \mathrm{P}^{tr}_{ij} + \gamma \sum_{j\neq i} v_j \mathrm{P}^{tr}_{ji}$$

$$= \gamma(v_i \mathrm{P}^{tr}_{ii} + \sum_{j\neq i} v_j \mathrm{P}^{tr}_{ji}) + (1-\gamma)(v_i \mathrm{P}^{tr}_{ii} + \sum_{j\neq i} v_i \mathrm{P}^{tr}_{ij})$$

Then we use the fact that $v_i \mathrm{P}^{tr}_{ii} + \sum_{j\neq i} v_j \mathrm{P}^{tr}_{ji} = \lambda v_i$ and group the terms by γ

$$\gamma\lambda v_i + (1-\gamma)(v_i \mathrm{P}^{tr}_{ii} + \sum_{i\neq j} v_i \mathrm{P}^{tr}_{ij})$$

Next, since P^{tr} is row-stochastic, we conclude

$$\lambda'_k v_i = \gamma\lambda_k v_i + (1-\gamma)v_i$$

$$\lambda'_k = 1 + \gamma\lambda_k - \gamma,$$

for all $k \in \{1, \ldots, |\bar{A}_\alpha|\}$. □

The 'horizon shift' operation helps to analyze properties of the stationary distribution as agents execute actions in the environment.

Corollary 5.3.3. *(Stationary distribution under 'horizon shift'): Let* $\mathrm{P}^{g\prime}$ *be the output of the 'horizon shift' operation on* P^g *as described in* (5.3)*, and* P^a *be any acceptance matrix. Let the stationary distributions for the transition matrix* P^{tr} *and* $\mathrm{P}^{tr\prime}$ *be denoted* v *and* v'*, respectively. Then*

$$v\mathrm{P}^{tr} = v\mathrm{P}^{tr\prime} = v.$$

The primary motivation behind recycling solutions is to reduce the number of iterations required for the simulated annealing step of `HSA` in order to reach the

stationary distribution. The distribution of recycled solutions remain close to the stationary distribution after executing a step in the environment and after the 'horizon shift' operation on P^g. Furthermore, by focusing the sampling probability on action schedules that deviate distant in the future, that sampling aids in the mixing of the Markov chain for future solutions more efficiently. Another interesting property of the 'horizon shift' operation is its effect on the relaxation time t_{rel}.

Corollary 5.3.4. *(Relaxation time under 'horizon shift'): Let* $\mathrm{P}^{g\prime}$ *be the output of the 'horizon shift' operation on* P^g *as described in* (5.3)*, and* P^a *be any acceptance matrix. Let the relaxation time for the transition matrix* P^{tr} *and* $\mathrm{P}^{tr\prime}$ *be denoted* t_{rel} *and* t'_{rel}*, respectively. Then*

$$t'_{rel} = \frac{t_{rel}}{\gamma}.$$

Proof. From Theorem 5.3.2, we have that the maximum eigenvalue that is not 1 is

$$\lambda'_2 = 1 + \gamma\lambda_2 - \gamma$$

The relaxation time follows from (**??**)

$$t'_{\text{rel}} = \frac{1}{\gamma'} = \frac{1}{\rho\gamma} = \frac{1}{\rho\gamma}$$
$$t'_{\text{rel}} = \frac{t_{\text{rel}}}{\rho}$$

□

This is a useful property to be aware of because the relaxation time yields bounds found in (**??**), for the mixing times of the Markov chain induced by the

transition matrix. The trade-off for using a smaller γ is that the mixing time increases, however it does allow for more mixing to take place in future events, which is beneficial because we recycle solutions for future time steps. We now focus on the multi-agent aspect of the algorithm by creating recommendations with machine learning.

5.3.3 Inference-based sampling

In this section, we aim to create a sampling matrix that is more efficient in terms of number of samples necessary in order to reach a Nash equilibria. To do this we design a process that generates a dataset $\mathcal{D}$ that contains inputs which correspond to images of the environment, and outputs which correspond to real-number values for selecting action schedules.

Creating a dataset

We train a model on the dataset so that the learned policy can provide recommendations for sampling during deployment. We take advantage of the fact that most robotic deployment scenarios are spatial by nature by training our policy to map a local image, $x_{\alpha,\mathrm{s}}$, of the environment to a vector of values that correspond to action schedules that the agent can select $\mathbb{y} \in \mathbb{R}^{|\bar{A}_\alpha|}$, i.e., $\pi : x_{\alpha,\mathrm{s}} \to \mathbb{y}$. The local image $x_{\alpha,\mathrm{s}}$ is translated and rotated with respect to the pose of agent α. We choose to assign values in $\mathbb{y}$ to action schedules according to

$$\mathbb{y}_{\bar{a}_\alpha} = \max_{\bar{a}_{-\alpha} \in \bar{A}_{-\alpha}} \; u(\bar{a}_\alpha, \bar{a}_{-\alpha}), \tag{5.5}$$

for all $\bar{a}_\alpha \in \bar{A}_\alpha$ as an incentive to select a joint action schedule that yield high rewards and cooperates with other agents. Particular high rewarding joint actions that require

two or more agents to cooperate may have difficulty being selected because they do not exist in any available 'finite improvement path' from the current solution. In this case, an agent may have to choose an action that yields less reward in order to escape local maximums. In order to get a set of inputs and outputs, $\mathbb{x}$ and $\mathbb{y}$, we randomize many states and solve for (5.5) as outlined in Algorithm 7.

Algorithm 7: Creating a dataset

```
CreateDataSet()
    D = ∅
     for n = 0 → N:
        x(s)
        y ∈ R^{|Ā_α|}
        for ā_α ∈ Ā_α:
            y_{ā_α} = 0
            for ā_{−α} ∈ Ā_{−α}:
                if Φ(ā_α, ā_{−α}) − Φ(∅, ā_{−α}) > y_{ā_α}
                    y_{ā_α} = Φ(ā_α, ā_{−α}) − Φ(∅, ā_{−α})
        D.append(input: x, label: y)
    return D
```

Generating sampling matrix $\mathrm{P}^{g,\pi}$

After collecting the data we train our learned policy π. We use the softmax function, indicated by $\Psi(\pi(x,\mathrm{s}),i) : \pi(x,\mathrm{s}) \times i \to \mathbb{R}$ as follows

$$\Psi(\pi(x,\mathrm{s}),i) = \frac{e^{\pi(x,\mathrm{s})_i}}{\sum_{\forall j} e^{\pi(x,\mathrm{s})_j}}$$

in order to convert the output values into a probability distribution to be used in the sampling matrix $\mathrm{P}^{g,\pi}$

$$\mathrm{P}^{g,\pi}_{\alpha,\mathrm{s}} = \begin{bmatrix} \Psi_{\pi(x_{\alpha,\mathrm{s}}),1} & . & . & \Psi_{\pi(x_{\alpha,\mathrm{s}}),|\bar{A}_\alpha|} \\ . & . & . & . \\ . & . & . & . \\ \Psi_{\pi(x_{\alpha,\mathrm{s}}),1} & . & . & \Psi_{\pi(x_{\alpha,\mathrm{s}}),|\bar{A}_\alpha|} \end{bmatrix} \tag{5.6}$$

With this sampling matrix, the probability of choosing an action schedule is the same from any other initial action schedule.

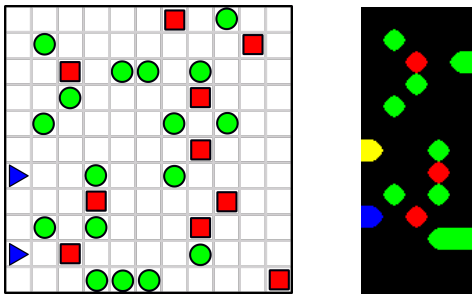
Remark 5.3.5. *(Keeping promises with inference-based sampling):* With the inference-based sampling scheme, the agents are incentivized to sample action schedules that achieve potentially high 'wonderful life utility'. Thus the recommended distribution of actions is weighted more heavily on action schedules that cooperate with other agents. We propose that this distribution inhibits the probability of breaking promises in the near term, as those action schedules will be sampled less often.

•

5.4 Cooperative orienteering

We design an algorithm to test and compare the different generation matrices. In the *cooperative orienteering* 2-D environment, agents are tasked with collecting resources, which may require multiple agents. Resources that require 1 agent yield a reward of 1 and and resources that require 2 agents yield a reward of 3. The environment scrolls to the left, but agents always occupy the left-most column where they

can choose actions in $(\text{up}, \text{stay}, \text{down})$. Figure 5.2 shows 'cooperative orienteering'.



(a) Cooperative orienteering (b) $x_{\alpha,\text{s}}$

Figure 5.2: (a) shows the 'cooperative orienteering' environment. In this environment agents are labeled with ▶ and are shown on left column in the environment. Agents move up and down in order to collect resource ●, which requires at least 1 agent on the coordinate that the resource occupies and yields 1 reward. Agents can also collect ■ which requires 2 agents on the coordinate and yields a reward of 3. (b) shows the relative view $x_{\alpha,\text{s}}$ of the agent which is used as an input to our learned policy π. In $x_{\alpha,\text{s}}$ green pixel represents resources which require 1 agent to collect and red pixels represent resources which require 2. The yellow pixel represents the agent α and the blue pixel represents an agent other than α.

We test the following modifications of `HSA` using different generation sampling schemes.

`HSA`$^{-}$: Horizon shift annealing with a 'flat' distribution P^{g-} defined as

$$\text{P}^{g-}_{ij} = \frac{1}{|\bar{A}_\alpha|},$$

for all i and j.

`HSA`†: Horizon shift planning with a 'geometrically reduced' distribution $\text{P}^{g\dagger}$ as defined in (5.2). For testing in the 'cooperative orienteering' we use $\gamma = .25$.

`HSA`$^{\pi}$: Horizon shift planning with the inferred sampling $\text{P}^{g,\pi}$ as defined in (5.6). For testing in the 'cooperative orienteering' environment we use a random forest

classifier with a decision tree depth of 12 with 12 estimators. We convert the environment to local images for each agent as depicted in Figure 5.2(b). We are able to achieve 82.3% validation accuracy with a validation loss (KL-divergence) of 0.1849 when trained on $10,000$ datasamples generated from Algorithm 7.

5.4.1 Single shift stationary distribution

In this section we test several of the presented algorithms by examining the number of steps that are required in order to reach the stationary distributions defined by the transition probabilities after a shift in the finite time horizon. In this particular test we use one agent. To be more precise, we take a scenario from the cooperative orienteering environment and allow the agent to search for a parameterized number of steps. The agent then executes their active selected action schedule. Once the agent acts, the agent begins searching again. We are particularly interested in this test because it validates our intuition that utilization of recycled action schedules decreases the number of steps required to reach the stationary distribution. Formally, the testing process is outlined in Algorithm 8.

The stationary distribution v is determined by calculating the transition probabilities determined with generation and acceptance probabilities. We calculate the KL-divergence between v and the empirically found distribution $\hat{v}$ for every iteration k as follows

$$L^k_{\hat{v}^k,v} = -\sum_{i=0}^{M} v_i^k \log \hat{v}_i + \sum_{i=0}^{M} v_i^k \log v_i. \tag{5.7}$$

The state of the environment is initially the same for each sample. Agents

Algorithm 8: Single shift sampling

$N^k_{\bar{a}_\alpha} = 0$ for all $k \in (0, \ldots, K_2)$ and $\bar{a}_\alpha \in \bar{A}_\alpha$

for $j = 0 \to J$:
    `environment.initialize`(seed=j)
    $\bar{a}_\alpha =$ `random.choice`$(\bar{A}_\alpha)$
    for $k = 0 : K_1$:
        $\bar{a}_\alpha =$ `HSA`$(\bar{a}_\alpha, k)$
    `environment.step`$(\bar{a}_\alpha)$
    for $k = 0 \to K_2$:
        $\bar{a}_\alpha =$ `HSA`$(\bar{a}_\alpha, k)$
        $N^k_{\bar{a}_\alpha} = N^k_{\bar{a}_\alpha} + 1$

$\hat{\pi}^k_{\bar{a}_\alpha} = \frac{N^k_{\bar{a}_\alpha}}{J}$ for all $k \in (0, \ldots, K_2)$ and $\bar{a}_\alpha \in \bar{A}_\alpha$
return $\hat{\pi}$

iterate through `HSA` for $k \in K_1$ steps and then select an action. We take samples such that the agent chooses to move straight and discard the rest. After executing the step in the environment, the stationary distribution of the agent's next choice is determined. Figure 5.3 illustrates this stationary distribution after the shift. The agent then plans for $k \in K_2$ iterations and the empirical distribution is determined with respect to k. The resulting KL-divergence between π and $\hat{\pi}$ is shown in Figure 5.4. We also examine the average expected reward of the action schedule selected during the second step as shown in Figure 5.5.

Let $\texttt{HSA}^-{}_{k_1}$ and $\texttt{HSA}^\dagger{}_{k_1}$ denote that the agent ran $\texttt{HSA}^-$and $\texttt{HSA}^\dagger$, respectively, for k_1 iterations during the first step. Note that the stationary distributions are slightly different for each of the sampling schemes and that (5.7) is determined with each of the schemes' stationary distributions, respectively. As expected, the empirical distributions converge to the stationary distributions as the number of iterations in the second step increase. Also, as the first step iterations k_1 increase, the second step iterations k_2 required for the KL-divergence to converge is smaller for both

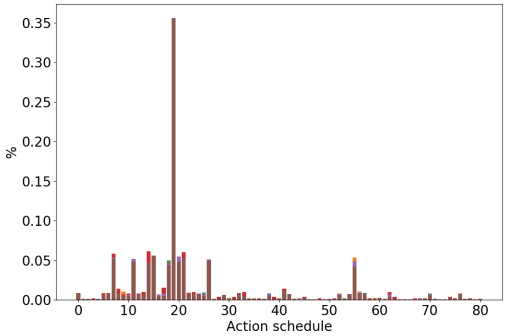


Figure 5.3: The average reward per step is shown between the stationary distribution and empirical distribution in the single shift experiment. The y-axis indicates probabilities in the stationary distribution for the corresponding action schedules that are indexed 0 through 80 on the x-axis.

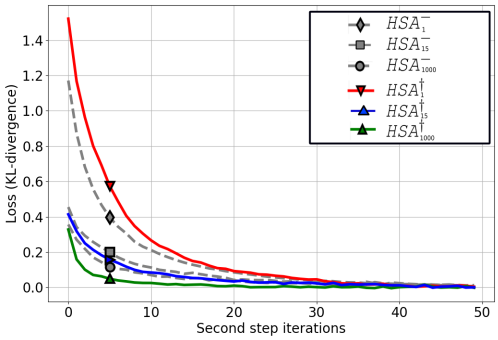


Figure 5.4: The KL-divergence is shown for the single shift experiment. $\texttt{HSA}^{-}$and $\texttt{HSA}^{\dagger}$are both ran initialy for 1, 15, and 1000 first step iterations. The x-axis indicates number of iterations during the second step and the y-axis indicates the KL-divergence between the stationary distribution and the empirical distribution during the second step.

sampling schemes. The faster convergence implies that Markov chain induced by the transition matrix at the second step is being mixed partially by first step, which compels our choice in recycling solutions. When comparing $\texttt{HSA}^{-}$and $\texttt{HSA}^{\dagger}$it is notable that $\texttt{HSA}^{\dagger}$performs preferably as k_1 increases. Intuitively, this is because the agent will dedicate more first step iterations for sampling future than sooner ones. This is undesirable when k_1 is low because the second step will not be mixed well and the

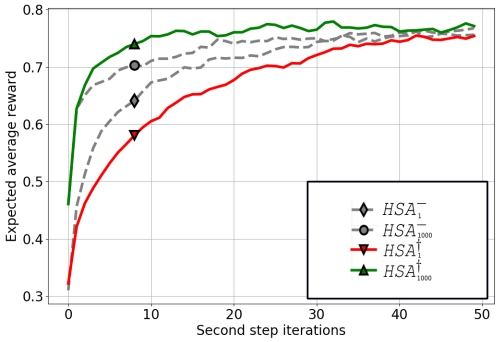


Figure 5.5: The expected reward is shown for the single shift experiment. The x-axis indicates number of iterations during the second step and the y-axis indicates the expected reward for action schedules chosen during the second step.

mixing time required for $\texttt{HSA}^\dagger$ is greater since the 'relaxation time' for $\texttt{HSA}^\dagger$is greater.

5.4.2 Probability in optimal Nash equilibrium

For this experiment we are interested in determining how often 2 agents arrive at a joint action schedule $\bar{a}$ that is in the set of optimal Nash equilibriums when considering a single step (no horizon shift) in the environment. Because we are not considering horizon shifts, we omit $\texttt{HSA}^\dagger$, which converges slower than $\texttt{HSA}^-$if there is no 'pre-shift' mixing.

As shown in Figure 5.6, the probability that the current joint action schedule is in the set of optimal Nash equilibriums increases with the number of iterations. We see that $\texttt{HSA}^\pi$yields a higher probability. This is because the learned policy is trained to output action schedules with high utility more often as determined in (5.5) and Algorithm 7.

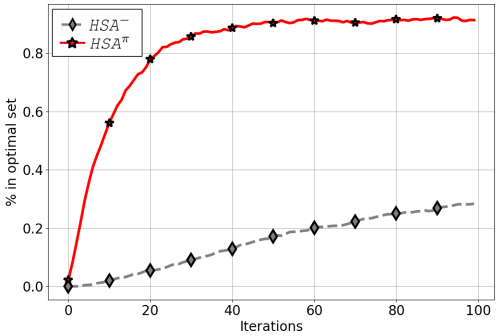


Figure 5.6: The percentage of joint action schedules that are in the set of optimal Nash equilibriums are shown. The y-axis indicates the percentage of trials where the joint action schedule selected at iteration k, on the x-axis, was in the set of joint Nash equilibriums.

5.4.3 Full trial cooperative reward

We determine the average reward per step that 2 agents receive when agents use the algorithms on 'cooperative orienteering' for N steps. In this experiment, we vary the number of iterations per step that agents take during the simulation and plot the results in Figure 5.7.

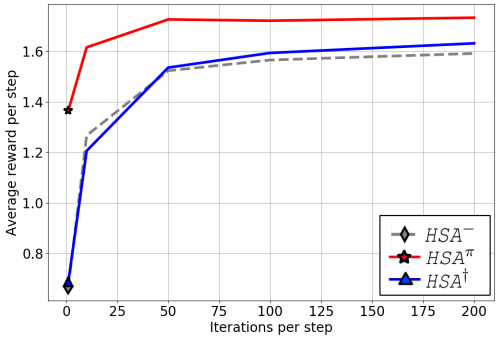


Figure 5.7: The average reward per step is shown for full trials vs. allowed iterations per time step. The y-axis indicates the average expected reward per step and the x-axis indicates the number of iterations that were allowed per time step.

Because $\texttt{HSA}^{\pi}$ requires inference from π at every time step, we also plot the results with respect to the amount of time that agents use for each step in Figure 5.8.

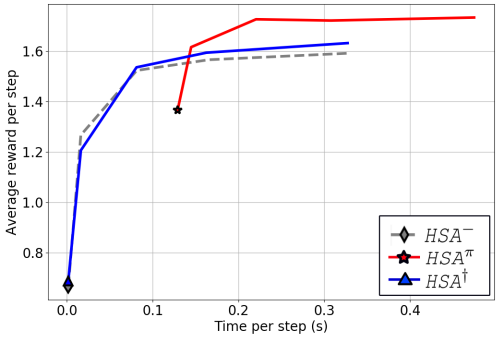


Figure 5.8: The average reward per step is shown for full trials vs. allowed time per time step. The y-axis indicates the average expected reward per step and the x-axis indicates the time in seconds that were allowed per time step.

As shown in Figure5.7 and Figure 5.8 the average reward gained per step increases with the number of iterations. As suspected, HSA†performs worse than HSA^{-}for low number of iterations per step. As the number of iterations increase, HSA†outperforms HSA^{-}, which may be a consequence its synergy with recycling solutions and its ability to mix future actions more efficiently. HSA$^{\pi}$performs the best which is likely because it sampled better action schedules more often due to the high categorical accuracy of the model. HSA$^{\pi}$does take some time for inference however, and initially performs worse than the other algorithms when considering real time per step.

5.4.4 Keeping promises

Lastly, we design a metric for the notion of keeping and breaking 'promises'. Given a deployment with two or more agents, we say that there was a 'promise' broken during a time step if all are true

- During iterations of simulated annealing, an agent expects to successfully collect

a resource that requires more than one agent.

- Another agent who was required for that resource's collection chooses a different action, resulting in the first agents inability to collect the resource.

In this experiment, we run the algorithms on the 'cooperative orienteering' environment with 2 agents for N time steps and determine the probability of steps where a promise was broken between the two agents.

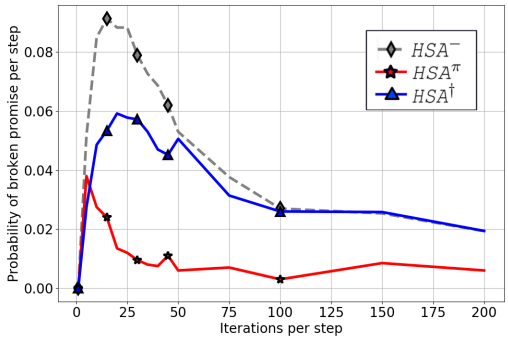


Figure 5.9: 1000 trials of 'cooperative orienteering' with $N = 1000$ time steps where the probability of steps where 'promises' are broken is depicted by the y-axis and the number of iterations per step is depicted by the x-axis.

As shown in Figure 5.9, the probability of breaking promises decreases as the number of iterations increase for all algorithms. As expected, promises are kept more often under both the geometrically reduced and inference-based sampling schemes when compared to the flat sampling scheme.

5.5 Conclusions

In this chapter, we have considered simulated annealing as a method for determining strategies amongst the agents. In particular, we consider the agents planning in a finite time horizon and recycle our solutions from one step to the next. We de-

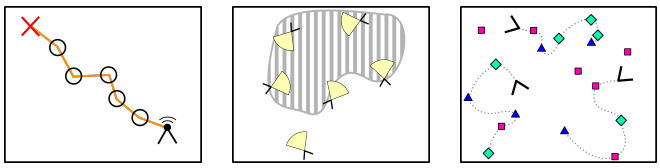
sign sampling algorithms to take advantage of the recycled solutions as well as use a learned model to output sampling probabilities that help us reach optimal Nash equilibriums during runtime. We provide analysis for the algorithms to show that given enough time, at least a local Nash equilibrium is found. Furthermore, we analyze the properties of our geometrically reduced sampling matrix $\mathrm{P}^{g\dagger}$ with respect to its stationary distribution as the finite time horizon shifts. In the future, we would like to extend this work by determining optimal γ to be used in the geometrically reduced sampling scheme that is dependent on number of iterations that agents have during each step in the finite time horizon. Futhermore, it may be possible to improve the learned policy and its ability to influence agents to reach an optimal Nash equilibrium by training the output conditioned on the current solution of the agents which would generate different values for $\mathrm{P}_{ij}^{g,\pi}$ and $\mathrm{P}_{kj}^{g,\pi}$.

Chapter 5, in full is currently being prepared for submission for publication of the material. Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Cascading agent tree search

Multi-agent planning is critical across robotic applications in disaster relief scenarios, exploration, navigation, surveillance, and production. Handling these scenarios is difficult due to the large number of possible states and actions that agents can take. This complexity grows when meeting objectives requires coordination among the agents. To tackle this, our approach leverages model-based reinforcement learning to develop efficient algorithms which scale with the number of agents and incorporate the need to plan cooperatively. We rely on training deep neural networks to predict promising actions for the purpose of improving the tree search in the future. During runtime, agents use tree search in a distributed fashion, guided by the previously-trained policy to complete an objective under time constraints. Figure 6.1 shows illustrative examples of environments motivating our algorithm design.



(a) $\mathcal{E}_{\text{comm}}$ (b) $\mathcal{E}_{\text{cov}}$ (c) $\mathcal{E}_{\text{res}}$

Figure 6.1: 2D multi-agent environments. In (a), agents must form a chain of communication from an operator to a point of interest. Agents are able to communicate if they are within a certain range of each other and agents are rewarded if a communication link between the target and operator exists. In (b), agents are able to detect in a cone-shaped region in front of them. The agents are tasked to jointly maximize their detection in an area of interest which is specified by an operator who is tasked to seek the target. In (c), static resources are scattered in the environment. Agents receive a reward for collecting a resource based on resource type, and some resources require two or more agents for collection.

6.1 Problem statement

Consider a scenario where cooperative homogeneous agents seek to maximize future discounted rewards in an environment modeled as a Markov decision process. Let $\mathcal{A}$ denote the set of agents and $\mathrm{s}_i \in \mathbb{R}^d$ be the state of agent i. The state of the environment is then given by $\mathrm{s} = [\mathrm{s}_1, \ldots, \mathrm{s}_{|\mathcal{A}|}, \mathrm{s}^\alpha] \in \mathrm{S}$, where $\mathrm{s}^\alpha = \mathbb{R}^p$ corresponds to the non-agent states of the environment. At every time step, each agent i chooses from a set of discrete actions given by $\mathrm{a}_i \in \mathrm{A}$. Agents act simultaneously according to a *joint action* defined as $\mathrm{a} \in \mathrm{A} = \mathrm{A}_1 \times \ldots \times \mathrm{A}_{|\mathcal{A}|}$, which is determined by the distributed selection of actions amongst agents. Given a joint state s and a joint action a, the probability that a state transitions to s' is $P^s_{s'|s,a}$. Agents receive the same reward at every step in the environment with $\mathcal{R}$ which encodes the success of cooperative tasks. The goal is to determine a policy that maximizes the state values for the Markov decision process $\langle \mathrm{A}, \mathrm{S}, \mathcal{R}, \Pr, \gamma \rangle$. Achieving this goal is challenging for multi-agent environments because the joint state and action space grow exponentially with the number of agents. We are interested in reasoning over environments where

the alignment of the joint actions of the agents may have a significant impact on performance.

6.2 Multi-agent tree search and bias exploitation

In this section we introduce MIPC and CATS, cf. Figure 6.2. First we discuss MIPC, an algorithm for constructing an informed policy offline. This construction requires the simulation of environments and data collection, which is performed by CATS.

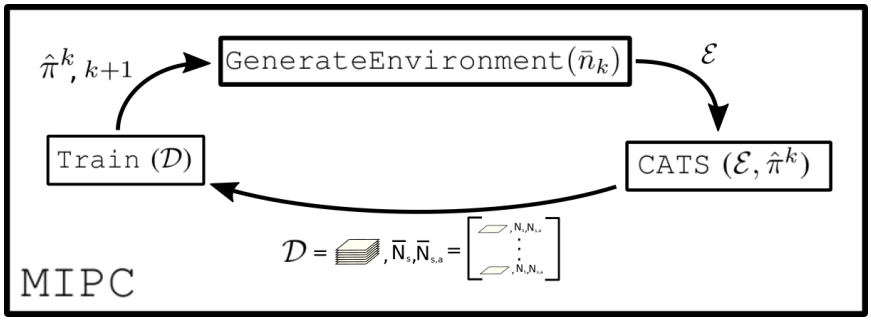


Figure 6.2: Flowchart of the iterative process MIPC. First, many environments are generated parameterized by a specified number of agents. In parallel processes, the tree search CATS generates expected state-action values and action selection distributions for each of the generated environments. A convolutional neural network is trained to map an image that represents a local perspective of a single agent to the output of CATS, yielding an informed policy.

6.2.1 Multi-agent informed policy construction

MIPC is an algorithm that runs offline to iteratively build an informed policy. There are three main components to it, cf. Figure 6.2: GenerateEnvironment, CATS, and Train. Let $\bar{n} = [\bar{n}_1, \bar{n}_2, \ldots]$ be a list with entries, $\bar{n}_k \in \mathbb{R}$, that correspond to the number of agents that will be spawned in an environment during the k-th iteration of MIPC. Hand tuning $\bar{n}$ allows for the customization of difficulty for each iteration.

In practice, we start with a small amount of agents which is slowly incremented with iterations of `MIPC`. The function `generate_environment`($\bar{n}_k$) creates a randomized environment with $\bar{n}_k$ agents. Iterations of `MIPC` begin by generating environments with n_1 agents. The environments are simulated in `CATS`, which gathers data using tree search. The data collected is used to create an informed policy $\hat{\pi}^1$ with `Train`. On the next iteration, environments are generated with n_2 agents and $\hat{\pi}^1$ is used to catalyze the data collection in `CATS`. The new data is used to create a new informed policy, $\hat{\pi}^2$. This process iterates up to $|\bar{n}|$ times.

The informed policy is a convolutional neural network that maps an image and an action to a number, $(x_\alpha, \mathrm{a}_\alpha) \mapsto \hat{\pi}_{x_\alpha, \mathrm{a}_\alpha}(x_\alpha, a_\alpha) \in (0, 1)$, where $x_\alpha$is an image that represents the local perspective of an agent α during the initial state of the simulation that is labeled by the action selection distribution determined in `CATS`. This mapping from state to an image is translated and rotated for agent α and is low resolution in practice to allow for fast inference. We map the state to an image for two reasons. First, the number of dimensions of the full state is dependent on the number of agents. Mapping the full state to an image results in a state space dimension that is not a function of the number of agents. Second, many multi-agent deployment problems are spatial by nature. Relevant spatial information can be captured by convolutional neural networks.

6.2.2 Cascading agent tree search

`CATS` is a variation of MCTS which utilizes a neural network to guide the action selection for multiple agents. Algorithm 9 presents the pseudocode for the main components: `GenerateBias`, `Traverse`, `Expansion`, `Simulation`, and `Backpropagation`.

In `GenerateBias` we generate an approximation of the neural network evaluated at the initial state of the simulation that we use to increase tree search speed. In `Traverse` we modify the tree traversal and action selection process of the tree search to accomodate for multiple agents. `Expansion`, `Simulation` and `Backpropagation` are all unchanged from the original UCT [KS06]. The tree search is executed from the perspective of a single agent. Actions of this agent and others are sampled with respect to a metric, *cascaded agent action selection*, where agents choose actions in a sequential manner. The order that agents choose actions is determined by $\bar{\mathcal{A}} = [\alpha_1, \ldots, \alpha_{|\mathcal{A}|}]$, where the first agent α_1 indicates the agent who is executing the tree search. `CATS` outputs the following, $\hat{Q}, N_s, N_{s,a}$, which are used to train informed policies or to choose actions during runtime.

Bias generation

Before performing tree search, we generate biases that influence actions selected. Ideally, the bias comes from the evaluation of the informed policy created by `MIPC` each time a new state is visited as a forward pass through the neural network. Unfortunately, this term has to be evaluated $\mathcal{O}(|\mathrm{A}|^D)$ times, restricting its use for large scale multi-agent systems. To avoid this problem, we devise a fast local approximation of informed policy for each agent before tree search in `GenerateBias`. To do this, we evaluate the informed policy at nearby locations for agent i while keeping all other agents at their original state. This is done for every action in the agent's action space. Evaluations are fit using regression and the output is the *local bias* $(s_\alpha, a_\alpha) \mapsto \psi_{s_\alpha, \mathrm{a}_\alpha}(s_\alpha, \mathrm{a}_\alpha) \in (0, 1)$ evaluated at the initial state of the environment.

Cascading agent action selection

Our proposed action selection is biased by the informed policy via the local bias ψ and allows for exploration of joint action spaces. To choose a joint action for the selection process, an action is chosen for each agent sequentially. Let $\bar{\mathrm{a}}$ be a *cascading joint action*, which is an ordered list of some agents' actions. The cascading joint action starts empty and is built sequentially as agents choose actions. We introduce *cascading states* to be $s^c = \langle \mathrm{s}, \bar{\mathrm{a}} \rangle$, which contains the current state of the environment s and the set of actions already selected by agents $\bar{\mathrm{a}}$. Agents choose their actions with respect to the number of times a cascaded state has been visited, N_{s^c}, implying that the action selection process is conditioned on the selections of previous agents in the sequential process. Algorithm 9 outlines the sequential process under the `Traverse` pseudocode. The action selection metric is modified accordingly to reflect the change of $N_{\mathrm{s}} \rightarrow N_{s^c}$ and $N_{\mathrm{s,a}} \rightarrow N_{s^c,\mathrm{a}_i}$ as

$$f_{caas}(s^c, \mathrm{a}_i) = \hat{Q}_{s^c,\mathrm{a}_i} + c_1 \sqrt{\frac{\ln N_{s^c}}{N_{s^c,\mathrm{a}_i}}} + \frac{\psi_{s^c.s_i,a_I}}{\sqrt{N_{s^c,\mathrm{a}_i}}}.$$

The maximization of this function leads the agent to choose the action most recommended by $\psi_{s^c.\mathrm{s}_i,\mathrm{a}_i}$ when visiting a state-action pair for the first time because $\hat{Q}_{s^c,\mathrm{a}_i}$ is initialized at 0. The cascading state now transitions both when an agent selects an action and when all of the agents take a step in the environment, which is a modification of the `Traverse` process in the tree search.

Remark 6.2.1. *(Advantages of cascaded agent action selection): This method of action selection has several implications. The cascade effect on agents' exploration grants the ability to discover reward signals behind joint actions. If only one agent is*

allowed to explore its individual action space at a time, then it is committed to plan strictly under the predictions of other agents. When those predictions do not include actions necessary for joint cooperation, then certain reward signals will be difficult to reach. The second advantage of cascading exploration is that agents are able to take a bad prior informed policy $\hat{\pi}^k$ and create a good post informed policy $\hat{\pi}^{k+1}$ given enough time. This is balanced by the exploitive local bias, which allows for search deep into the tree for tractability. This algorithm is used to generate a dataset that will train $\hat{\pi}^{k+1}$ for the first agent only. •

Remark 6.2.2. *(Cascaded agent action selection vs joint action selection): Our proposed action selection does not improve the branching factor of the tree search when compared with the joint case. There are structural advantages that lead to efficient tree traversal when allowing agents to choose one at a time. A particular case is when agents receive some rewards based on non-joint actions or partial rewards for joint actions.* •

6.2.3 Online and offline deployment

`CATS` is meant to be run both offline and online. `CATS` is used to generate data for training informed policies offline for `MIPC` as shown in Figure 6.2, where the agent order $\bar{\mathcal{A}}$ is chosen at random. In the online case, CATS is executed on each of the agents individually during deployment and all agents take actions simultaneously. Each of the agent choose themselves as first in $\bar{\mathcal{A}}$, and the rest is chosen heuristically.

6.3 Convergence of CATS to optimal value

Here we analyze the convergence properties of CATS. Our technical analysis proceeds in two steps. We discuss how to properly model the sequence of multi-agent state transition and action selection in the algorithm execution via cascaded MDPs and then, building on this construction, we establish that the state-action value estimate determined by CATS converges to the optimal value of the MDP.

6.3.1 Cascaded MDPs

The sequential action selection process employed by f_{caas}, cf. Section 38, requires care in using standard MDPs to model the process of state transition and action selection. This is because agents choose their actions conditioned on the current tentative sequence of actions decided by agents which choose earlier in the sequence: in turn, this is information not contained in states of the originally defined MDP. To address this issue, we transform the original MDP with multiple agents and joint action selection into a *cascaded MDP*. Additionally we show that this transformation has a unique and convergent state value for every state in the new state space using value iteration with discount $0 < \gamma < 1$.

Let $\mathcal{C} = \langle \mathrm{S}^c, \mathrm{A}^c, P^c, \mathcal{R}^c, \gamma^c \rangle$, where $s^c \in \mathrm{S}^c$ contains states defined in original MDP as well as cascaded actions $\bar{\mathrm{a}}$. The notation $s^c_{j,n} = \langle \mathrm{s}_j, [\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_n] \rangle$ specifies the environment state s_j from the original MDP and the actions that are in $\bar{\mathrm{a}}$. The set of allowable actions for each agent is $\mathrm{a}_i \in \mathrm{A}_i \in \mathrm{A}^c$ and P^c, $\mathcal{R}^c$, and γ^c are dependent on one of two types of state transitions that we distinguish in $\mathcal{C}$. The first type of state transition, *intra-agent transitions*, occurs when the current state's cascaded action is not full, i.e., $s^c_{j,n}$ such that $n < N - 1$. In this situation, the next action that will be

picked will not yet complete the sequence of actions (one per agent) so this action will not yet result in a step in the environment. Picking an action results in a determined probability of transition, $P^c_{s^c_{j,n+1}|s^c_{j,n},\mathrm{a}} = 1$. The next state will contain the previously selected sequence of actions *and* the newly selected a. The reward given for this transition is always zero, $\mathcal{R}^c_{s^c_{j,n+1},\mathrm{a},s^c_{j,n}} = 0$. Furthermore, the discount factor is always $\gamma^c = 1$. The second type of state transition, *environmental transitions*, happens when the last agent in the sequence chooses an action, i.e., $s^c_{j,n}$ such that $n = N - 1$. At this point, the agents all take their promised action in the environment and the next state, $s^c_{j+1,0}$, contains changes in the environment and an empty cascaded action. This transition has the corresponding probability of transition in the original MDP, $P^c_{s^c_{j+1,0}|s^c_{j,N-1},\mathrm{a}} = \Pr_{s_{j+1}|s_j,\bar{\mathrm{a}}}$, given that $s^c.\bar{\mathrm{a}} = \bar{\mathrm{a}}$ and $s^c.s = s$. The reward under this transition is determined by the original MDP as well, $\mathcal{R}^c_{s^c_{j+1,0},\mathrm{a},s^c_{j,N-1}} = R_{s_{j+1},\bar{\mathrm{a}},s_j}$, such that $s^c.s = s$. Finally, the discount factor used is the same as the constant discount factor from the original MDP, $\gamma^c_{s^c} s = \gamma$. Figure 6.3 shows the original MDP and

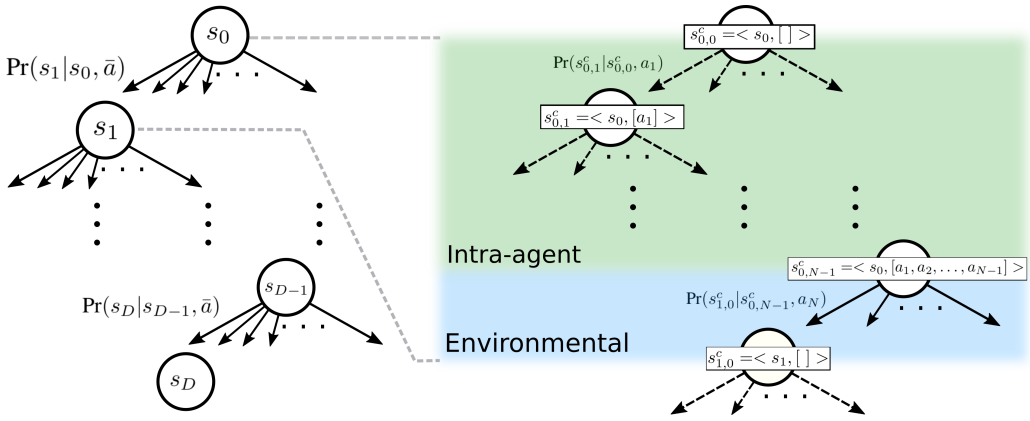


Figure 6.3: Original MDP (left) and associated cascaded MDP (right). The MDPs start at the root state s_0 and $\mathrm{s}^c_{0,0}$, respectively. The original MDP is shown to depth D. Its first step is expanded to show the additional steps of the cascaded MDP. Transition probability $\Pr(\mathrm{s}^c_{0,1}|\mathrm{s}^c_{0,0}, \mathrm{a}_1) = 1$ in the green region indicates an intra-agent state transition (dashed line), where the first agent in the sequence selects action a_1. Further down the tree in the blue region, $\Pr(\mathrm{s}^c_{1,0}|\mathrm{s}^c_{0,N-1}, \mathrm{a}_N) = \Pr(\mathrm{s}_1|\mathrm{s}_0, \bar{\mathrm{a}})$ such that $\bar{\mathrm{a}} = [\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_N]$, represents environment transitions (solid line).

the cascaded MDP. The convergence of MDPs under value iteration generally depend on having the discount factor $0 < \gamma < 1$. Our next result shows that given the constraints of the probability of transition, reward function, and the discount factor under intra-agent transitions and environment transitions, the state value converges under the Bellman operator $\mathcal{B}$ defined by

$$\mathcal{B}(V_s) = \max_a \left(r_{s,a} + \gamma \sum_{\mathrm{s}} \mathrm{Pr}_{\mathrm{s}'|\mathrm{s},a} V_{\mathrm{s}'} \right).$$

For $0 < \gamma < 1$, the Bellman operator induces V^*.

Theorem 6.3.1 (Convergence to V^*)**.** *Consider an MDP and its associated cascaded MDP, generated as described above. Let V be the estimated state value induced by the Bellman operator in a cascaded MDP. V converges to the optimal state value of the original MDP, V^*. Furthermore, for all s^c such that $s^c.s = s$, the state values from the original MDP and cascaded MDP are equivalent, $V_{s^c} = V^*_{\mathrm{s}}$.*

Proof. We examine the two types of transitions in the cascaded MDP. If $s^c_{j,n}$ such that $n < N - 1$ then an intra-agent transition occurs next. Let $\mathrm{S}^c_{j,n+1}$ be the set of all states that could occur at depth $n + 1$, then

$$\mathcal{B}(V_{s^c}) = \max_{s^c_{j,n+1}} V_{s^c_{j,n+1}}, \tag{6.1}$$

Since $r_{s^c_{j,n+1}, s^c_{j,n}} = 0$, $P^c_{s^c_{j,n+1}|s^c_{j,n},a} = 1$, and $\gamma^c = 1$. Therefore, a Bellman operation sets the state value equal to the next state value for intra-agent transitions. This implies that in a sequence of intra-agent transitions, all state values become equal to each

other under multiple Bellman operations and

$$\begin{aligned}\mathcal{B}^{N-1}(V_{s^c_{j,0}}) = \mathcal{B}^{N-1}(V_{s^c_{j,0}}) &= \mathcal{B}^{N-2}(V_{s^c_{j,1}}) \\ &= \ldots = \max_{s^c_{j,N-1} \in \mathrm{S}^c_{j,N-1}} V_{s^c_{j,N-1}},\end{aligned}$$

where $s^c_{j,N-1}$ is a state prior to all agents choosing their actions. After the last agent selects an action, an environmental transition occurs and the next state is of the form $s^c_{j+1,0}$. Under $N-1$ Bellman operations, $s^c_{j,0}$ becomes affected by the next environmental transition, which can be shown to be a contraction as follows,

$$\begin{aligned}&|\mathcal{B}^{N-1}(V_{s^c_{j,0}}) - \mathcal{B}^{N-1}(\bar{V}_{s^c_{j,0}})| \\ &\quad = |\mathcal{B}^{N-2}(V_{s^c_{j,1}}) - \mathcal{B}^{N-2}(\bar{V}_{s^c_{j,1}})| \\ &\quad = |\mathcal{B}^{N-3}(V_{s^c_{j,2}}) - \mathcal{B}^{N-3}(\bar{V}_{s^c_{j,2}})| \\ &\quad = \ldots = |\mathcal{B}(V_{s^c_{j,N-1}}) - \mathcal{B}(\bar{V}_{s^c_{j,N-1}})|,\end{aligned}$$

where the derivation of convergence for standard MDPs can be applied. For brevity, we use $\overline{\Pr}$ to denote $\Pr_{s^c_{j+1,0}|s^c_{j,N-1},a_{n+1}}$,

$$\begin{aligned}&|\mathcal{B}(V_{s^c_{j,N-1}}) - \mathcal{B}(\bar{V}_{s^c_{j,N-1}})| \\ &\quad = \max_{a_{n+1}}\{r_{s^c_{j,N-1},a_{n+1}} + \gamma \sum_{s^c_{j+1,0}} \overline{\Pr}\, V_{s^c_{j+1,0}}\} \\ &\qquad - \max_{a'_{n+1}}\{r_{s^c_{j,N-1},a'_{n+1}} + \gamma \sum_{s^c_{j+1,0}} \overline{\Pr}\, \bar{V}_{s^c_{j+1,0}}\}) \\ &\quad \leq \max_{a_{n+1}}|\{r_{s^c_{j,N-1},a_{n+1}} + \gamma \sum_{s^c_{j+1,0}} \overline{\Pr}\, V_{s^c_{j+1,0}}\}\end{aligned}$$

$$
\begin{aligned}
&\quad - \{r_{s^c_{j,N-1},a_{n+1}} + \gamma \sum_y \overline{\Pr}\, \bar{V}_{s^c_{j+1,0}}\}| \\
&= \max_{a_{n+1}} \gamma \sum_y \overline{\Pr}\, |V_{s^c_{j+1,0}} - \bar{V}_{s^c_{j+1,0}}| \\
&\leq \max_{a_{n+1}} \gamma \sum_{s'} \overline{\Pr}\, \|V - \bar{V}\|_\infty \\
&= \gamma \, \|V - \bar{V}\|_\infty \max_{a_{n+1}} \sum_y \overline{\Pr} \, = \gamma \, \|V - \bar{V}\|_\infty .
\end{aligned}
$$

Finally, because $\lim_{k\to\infty} \mathcal{B}^k(V_{s^c}) = V^*$, state values under Bellman operations converge.

□

6.3.2 Convergence to optimal state value

Next, we show that the state-action value estimate determined by CATS converges to the optimal value of the MDP. This requires careful consideration of the effect of the local bias term ψ. Our treatment follows the argumentation in [KSW06] for the convergence analysis of UCT, noting the necessary differences. First, let X_{it} be the payoff rewarded when action i is taken at time t. The average payoff is given by

$$\bar{X}_{im} = \frac{1}{m} \sum_{t=1}^{m} X_{im}.$$

Let $\mu_{im} = \mathbb{E}[\bar{X}_{im}]$ be the expected payoff for taking action i after m attempts and define

$$\mu_i = \lim_{m\to\infty} \mu_{im}. \tag{6.2}$$

Finally, let δ_{im} be the drift in the mean payoff,

$$\mu_{im} = \mu_i + \delta_{im}. \tag{6.3}$$

We make the following assumption.

Assumption 6.3.2 (Bounds on expected payoff)**.** *Fix $1 \leq i \leq K$, where $K = |\mathrm{A}^c|$. Let $\{\mathcal{F}_{it}\}_t$ be a filtration such that $\{X_{it}\}_t$ is $\{\mathcal{F}_{it}\}$-adapted and $X_{i,t}$ is conditionally independent of $\mathcal{F}_{i,t+1}, \mathcal{F}_{i,t_2}, \ldots$ given $\mathcal{F}_{i,t-1}$. Then $0 \leq X_{it} \leq 1$ and the limit of μ_{im} exists. Further, we assume that there exist a constant $C_p > 0$ and an integer N_p such that for $m \geq N_p$, for any $\delta > 0$, $\Delta_m(\delta) = C_p\sqrt{m \ln(1/\delta)}$, the following bounds hold:*

$$Pr(m\bar{X}_{is} \geq m\mathbb{E}[\bar{X}_{in}] + \Delta_m(\delta)) \leq \delta,$$
$$Pr(m\bar{X}_{is} \leq m\mathbb{E}[\bar{X}_{in}] - \Delta_m(\delta)) \leq \delta.$$

We let $\Delta_i = \mu^* - \mu_i$, where i and $*$ indicate suboptimal and optimal actions, respectively. Assumption 6.3.2 implies that δ_{it} converges to 0. Therefore, for all $\epsilon > 0$, there exists $N_0(\epsilon)$ such that if $t \geq N_0(\epsilon)$, then $|\delta_{it}| \leq \epsilon\Delta_i/2$ and $|\delta_t^*| \leq \epsilon\Delta_i/2$, where $|\delta_t^*|$ is the drift corresponding to the optimal action. In particular, it follows that for any optimal action, if $t > N_0(\epsilon)$, then $\delta_t^* \leq \epsilon/2 \min_{i|\Delta_i>0} \Delta_i$. We are ready to characterize the convergence properties of CATS.

Theorem 6.3.3 (Convergence of CATS)**.** *Consider CATS running to depth D where $K = |\mathrm{A}^c|$ is the number of actions available to each agent. Assume that rewards at*

the leafs are in the interval $[0, 1]$. *Then,*

$$|\hat{Q}_{s^c} - V^*_{s^c}| = |\delta^*_m| + \mathcal{O}\left(\frac{KND\log(m) + K^{ND}}{m}\right), \tag{6.4}$$

for any initial state s^c. *Further, the failure probability at the root converges to zero as the number of samples grow to infinity.*

Proof. The proof follows closely the steps in [KSW06] to establish convergence of UCT. To determine the expectation on the number of times suboptimal actions are taken, fix $\epsilon > 0$ and let $T_i(n)$ denote the number of plays of arm i. Then if i is the index of a suboptimal arm and assuming that every action is tried at least once, we bound the T_i using the indicator function, $\{I_t = i\}$, where l is an arbitrary integer,

$$\begin{aligned}
T_i(m) &= 1 + \sum_{t=K+1}^{m} \{I_t = i\} \leq l + \sum_{t=K+1}^{m} \{I_t = i, T_i(t-1) \geq l\} \\
&\leq l + \sum_{t=K+1}^{m} \{\bar{X}^*_{T^*(t-1)} + c_{t-1,T^*(t-1)} \leq \bar{X}_{i,T_i(t-1)} \\
&\quad + c_{t-1,T_i(t-1)}, T_i(t-1) \geq l\} \\
&\leq l + \sum_{t=K+1}^{m} \{\min_{0<z<t} \bar{X}^*_z + c_{t-1,z} \leq \max_{l \leq z_i < t} \bar{X}_{i,z_i} + c_{t-1,z_i}\} \\
&\leq l + \sum_{t=1}^{\infty} \sum_{z=1}^{t-1} \sum_{z_i=l}^{t-1} \{\bar{X}^*_z + c_{t,z} \leq \bar{X}_{i,z_i} + c_{t,z_i}\}.
\end{aligned}$$

The last term, $\bar{X}^*_z + c_{t,z} \leq \bar{X}_{i,z_i} + c_{t,z_i}$, implies one of three possible cases

$$\bar{X}^*_z \leq \mu^* - c_{t,z} \tag{6.5a}$$

$$\bar{X}_{i,z_i} \geq \mu_i + c_{t,z_i} \tag{6.5b}$$

$$\mu^* \leq \mu_i + 2c_{t,z_i}, \tag{6.5c}$$

where l represents the number of times (6.5c) occurs, and (6.5a) and (6.5b) are characterized by Assumption 6.3.2. We proceed by finding upper bounds for each of these three cases. Under Assumption 6.3.2, we can use the Chernoff-Hoeffding bounds [SSS95] $\mathbb{P}(\bar{X}_{iz} \geq \mathrm{E}[\bar{X}_{im}] + c) \leq e^{-2c^2 z}$ and $\mathbb{P}(\bar{X}_{iz} \leq \mathrm{E}[\bar{X}_{im}] - c) \leq e^{-2c^2 z}$ to bound cases (6.5a) and (6.5b). We apply $c = C\sqrt{\frac{\ln(t)}{z}} + \frac{1}{\sqrt{z}}$ to $e^{-c^2 z}$, where $C \geq \sqrt{2}$, $t > 0$, and $z > 0$ to get

$$\begin{aligned}
&\mathbb{P}(\bar{X}_z^* \leq \mu^* - c_{t,z}) \leq e^{-2c^2 z} \\
&\leq e^{-2(C\sqrt{\frac{\ln(t)}{z}}+1/z)^2 z} \leq e^{-2(\sqrt{2\frac{\ln(t)}{z}})^2 z} = t^{-4},
\end{aligned}$$

and $\mathbb{P}(\bar{X}_{i,z_i} \geq \mu_i + c_{t,z_i})$ follows similarly. To bound l we look at (6.5c) where $2c_{t,z} \leq (1-\epsilon)\Delta_i$. Solving for z yields

$$\begin{aligned}
z &\leq \frac{2(\Delta_i + C^2\log(t) - \epsilon\Delta_i)}{\epsilon^2\Delta_i^2 - 2\epsilon\Delta_i^2 + \Delta_i^2} \\
&\quad + \frac{2\sqrt{-2\epsilon C^2\Delta_i\log(t) + C^4\log^2(t) + 2C^2\Delta_i\log(t))}}{\epsilon^2\Delta_i^2 - 2\epsilon\Delta_i^2 + \Delta_i^2} \\
&\leq \frac{WC^2\log(t)}{(1-\epsilon)^2\Delta_i^2}
\end{aligned}$$

such that some constant W, which can be upper bounded for any $t \geq 2$, $0 \geq \epsilon \geq 1$, and $d > 0$. Note that this requires Assumption 6.3.2, where $t > N_p$ and $t > m > N_0(\epsilon)$. Therefore l is bounded with

$$l = \max\{z, N_0(\epsilon), N_p\} \leq \frac{WC^2(\ln(t))}{(1-\epsilon)^2\Delta_i^2} + N_0(\epsilon) + N_p.$$

Our expected T_i is then

$$\mathbb{E}[T_i(m)] \leq \frac{WCp^2(\ln(t))}{(1-\epsilon)^2\Delta_i^2} + N_0(\epsilon) + N_p$$
$$+ \sum_{t=1}^{\infty}\sum_{s=1}^{t-1}\sum_{s_\alpha=l}^{t-1}\frac{1}{t^4} \leq \frac{WC^2(\ln(t))}{(1-\epsilon)^2\Delta_i^2} + N_0(\epsilon) + N_p + \frac{\pi^2}{3},$$

which is of order $\mathcal{O}(Cp^2\ln(m) + N_0)$. The rest of the proof follows from [KSW06] where this bound is used to derive the expected regret as well as convergence of the probability of choosing suboptimal arms to 0. The statement follows via induction on the depth D^c, noting that $D^c = |A||D|$ as a result of the construction of the cascaded MDP. □

The convergence bound is worse for CATS than for UCT (i.e., the constant W in (6.4) is larger than the constant in [KSW06]) since the worst-case scenario for the local bias term has to be accounted for. In practice the local bias term helps the convergence rate, as we demonstrate in Section 6.4.

6.4 Implementation on 2D environments

In this section we test 3 environments defined in Figure 6.1 and discuss performance of several algorithms in a variety of metrics. Significant parameters of the experiments are shown in Table 6.1. The following variations of tree search and reinforcement learning algorithms are implemented.[1]

`CATS` The algorithm we propose in Section 6.2 which utilizes the informed policy generated by `MIPC`.

[1]Hyperparameters for the algorithms and environments can be found at https://github.com/aaronma37/cats_hyperparameters.

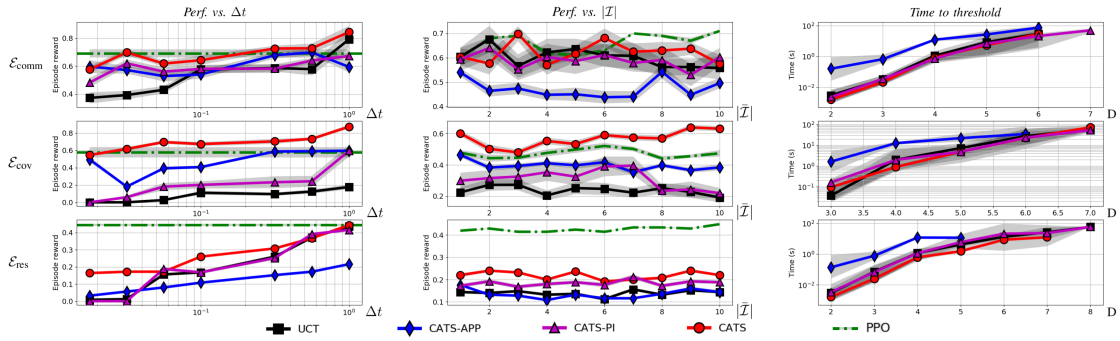


Figure 6.4: Results from the experiments in each of the environments. Each datapoint is sampled 100 times and the variance is shown as the shaded region. Variance of results in trials are affected by parameters on initial conditions and inherent properties of each of the environments. For *Perf. vs.* Δt and *Perf. vs.* $|\mathcal{A}|$ a higher value is better and for *Time to threshold* a lower value is better.

`CATS-`π Our proposed algorithm with cascading action selection but withouts the bias term from the informed policy.

`CATS-APP` Our proposed algorithm without use of the local bias approximation. This algorithm evaluates the informed policy at every newly visited state to bias tree search.

`UCT` Monte-Carlo tree search using upper confidence bound for action selection on the full joint action space.

`PPO` Proximal policy optimization (PPO), a policy-based model-free reinforcement learning algorithm [SWD+17]. In our adaptation, each agent uses the same policy to choose actions given their local state image x.

Figure 6.4 displays results for the following experiments.

6.4.1 Performance vs. allotted simulation time (*Perf. vs.* Δt)

The sum of rewards per episode vs. Δt (time allotted for tree search per step) for L number of steps in the environment during an online deployment. We find that for each of the environments, `CATS` and `CATS-APP` have strong performance given limited simulation time in the tree search. It is likely that `CATS` scales better than `CATS-APP` in this experiment because evaluation of the local bias takes less time than a forward pass through $\hat{\pi}$ at each node during the tree search. Performance of `CATS-`π and `UCT` both start off low and increase as Δt increases as expected, however the cascading action selection structure enables `CATS-`π to scale better with Δt. `PPO` does not perform tree search so it is shown at constant performance. As Δt increases, `CATS` performs equal or better than `PPO` in the environments tested.

6.4.2 Performance vs. number of agents (*Perf. vs.* $|\mathcal{A}|$)

The sum of rewards per episode vs. $|\mathcal{A}|$ for L number of steps in an online deployment is tested. As the number of agents increases, the branching factor increases exponentially. This results in a shallow tree search because of limited time allotted for simulation. `CATS` sees superior performance as $|\mathcal{A}|$ increases as it is able to effectively simulate other agents during search under time constraints. Adding agents means an exponential increase in evaluations of the informed policy in `CATS-APP`. In $\mathcal{E}_{\text{comm}}$, `CATS-APP` performs poorly and likely would have benefited if it had more time to explore joint actions during tree search. `CATS-`π yields better performance than `UCT` in some cases where the cascading action selection allows an agent to find an adequate action when determining the best joint action is difficult under the time

constraints. In some cases, PPO performs better than the tested tree search algorithms as the increased branching factor of tree search inhibits their performance.

6.4.3 Simulation time to threshold value (*Time to threshold*)

The amount of simulation time vs. depth of tree search (D) required during simulation to find an adequate solution in the environment, which is determined when $\hat{Q}$ is greater than a threshold value. This experiment measures how quickly an agent finds a satisfactory state in the MDP with respect to distance of that state, in terms of steps in the environment. CATS-APP performs worse than other algorithms because the evaluation of the informed policy for each agent at each new state becomes computationally expensive with respect to tree search depth. Performance in the other algorithms is influenced by how effectively agents are able to choose actions during tree search. PPO is omitted since it does not perform tree search.

6.5 Conclusions

We provide two major contributions, first being a scalable approach for determining an informed policy using MIPC to recycle and reduce the amount of time required for data collection. The second contribution, CATS, is a variation of Monte-Carlo tree search, which provides a balance of exploitation and exploration that utilizes prior knowledge about the environment and multi-agent scenarios. Agent explore sequentially allowing for robustness to errors in the prior informed bias enabling them to distributively determine solutions for cooperative objectives. We show that this modified MDP converges to the optimal state value and CATS estimates the optimal state value when using misinformed biases. The efficacy of CATS is shown Our al-

gorithm is compared against variations of `UCT` with a joint action spaces, where it excels when joint-actions are required for rewards but require low action space size for tractability.

Chapter 6, in full, is a reprint of the material as it appears in IEEE Robotics and Autonation Letters 5 (2) 2020, 1819-1826. Ma, Aaron; Ouimet, Mike; Cortés Jorge. The dissertation author was the primary investigator and author of this paper.

Algorithm 9: `MIPC` and `CATS`

`MIPC` $(N, \hat{\pi}_0)$
  for $k = 0$ to $|\bar{n}|$:
    $\mathcal{D} = \emptyset$
    do in parallel:
      $\mathcal{E} \leftarrow$ `GenerateEnvironment`$(\bar{n}_k)$
      $\Phi_{s,0}, N_{\mathrm{s}}, N_{\mathrm{s,a}} \leftarrow$ `CATS` $(\mathcal{E}, \hat{\pi}^k)$
      $\mathcal{D}$.append$(\Phi_{s,0}, N_{\mathrm{s}}, N_{\mathrm{s,a}})$
    $\hat{\pi}^{k+1}$.`Train` $(\mathcal{D})$
  return $\hat{\pi}^{|\bar{n}_k|}$

`CATS` $(\mathcal{E}, \bar{\mathcal{A}})$:
  for agent i in $\bar{\mathcal{A}}$
    $\psi_i \leftarrow$ `GenerateBias`$(\mathcal{E}, \hat{\pi}^k)$
  for allotted time
    `Traverse`$(s^c, \bar{\mathcal{A}}, k)$
    `Expansion`$(\ldots)$
    `Simulation`$(\ldots)$
    `Backpropagation`$(\ldots)$

`GenerateBias` $(\mathcal{E}, \hat{\pi}^k, i)$:
  for a_i in A_i:
    for allotted time
      s'_i = sample uniformly nearby s_i
      $s' = s$ such that with s'_i replaces s_i
      X.append(s'_i)
      Y.append$(\hat{\pi}_{x_i,a_i})$
    $\psi_{i,s_i,a_i} \leftarrow$ regression of X on Y
  return $\psi_i = (\psi_{i,a_1}, \ldots, \psi_{i,a_{|A|}})$

`Traverse`$(s^c, \bar{\mathcal{A}}, k)$:
  if $k < |\bar{\mathcal{A}}|$:
    s^c.ā.append$\big(\mathrm{argmax}_{\mathrm{a}_i \in \mathrm{A}_i} f_{caas}(s^c, \mathrm{a}_i)\big)$
    $k = k + 1$
  else:
    $s' \leftarrow$ evolve environment
    $s^c = \langle s', [\,] \rangle$
    $k = 0$
  if s^c is unvisited:
    return s^c
  else:
    `Traverse` $(s^c, \bar{\mathcal{A}}, k)$

Table 6.1: Parameters for each environment and experiment.

	Perf. vs. Δt			*Perf. vs.* $\lvert\mathcal{A}\rvert$			*Time to threshold*		
	$\mathcal{E}_{\text{comm}}$	$\mathcal{E}_{\text{cov}}$	$\mathcal{E}_{\text{res}}$	$\mathcal{E}_{\text{comm}}$	$\mathcal{E}_{\text{cov}}$	$\mathcal{E}_{\text{res}}$	$\mathcal{E}_{\text{comm}}$	$\mathcal{E}_{\text{cov}}$	$\mathcal{E}_{\text{res}}$
Δt	—	—	—	.1s	.1s	.1s	—	—	—
$\lvert\bar{\mathcal{A}}\rvert$	3	3	2	—	—	—	3	3	3
L	10	10	10	10	10	10	—	—	—

Chapter 7

Conclusions

To fully enable autonomy for UxVs, we have developed algorithms which are capable of planning in a large variety of scenarios. We are inspired by many state-of-the-art approaches which work in the framework of Markov chains and Markov decision processes in order to generalize the environment and system dynamics. Even though the use of reinforcement learning for robotic planning is a popular researched topic, task planning is a challenge for many reasons, especially as the state and action space become large. Furthermore, extending scenarios to the multi-agent case vastly increases the difficulty as the action and state space become joint and grow exponentially with the number of agents. In this dissertation we discuss several multi-agent algorithms that can take advantage of recent advances in computational power.

Our first approach attempts to alleviate the curse of dimensionality by geometrically splitting the environment into regions and by planning in a hierarchical sense. The hierarchical approach selects sub-environments on the top (slow level) and optimizes trajectories through those sub-environments on a low (fast level). By treating sub-environments as a separate problem we are able to formulate them as

semi-Markov decision processes. In doing so, we are able to reduce both the state and action space to a point where we can train the agents offline for quick inference of the state value during online deployment. Furthermore, being able to quickly infer the state value of a particular sub-environment increases the performance of the sub-environment selection by selecting environments that are likely to yield better rewards given the prior knowledge. This approach yields compelling results when used for planning in massive environments and presents many possibilities for future extensions. In particular, model-free reinforcement learning could be used to generate policies for generating sub-environments, as it would greatly benefit from offline training and is a non-convex optimization problem that could benefit from more efficient sampling. On the top level, during the search for a sub-environment, there are many different optimization strategies that are unexplored which may lead to better performance. In the dissertation, we have proposed a sequential approach that is based in submodularity in order to guarantee a theoretical lower bound. Other strategies may be utilized for better performance in an empirical sense. On the low level, it is difficult to choose what is the best way to find an optimal policy in the semi-Markov decision process and this is largely determined on a case-by-case basis. However a variety of reinforcement learning algorithms could be adequate.

Next, we have introduced a finite time horizon simulated annealing based approach. In this algorithm, we consider previous algorithms which attempt to plan tasks for agents by distributively sampling their action schedules with respect to a finite time horizon. We casted the task planning as a potential game, where agents are able to find a Nash equilibrium by using the 'wonderful life utility'. We presented a novel structure for the sampling matrix during the simulated annealing process.

This sampling matrix is a block matrix organized with respect to the finite time horizon. This enabled us to study the properties of the stationary distribution of the Markov chain associated to the simulated annealing as agents execute steps in the environment and the finite time horizon shifts. We present an algorithm for agents to choose their actions and show convergence to a local Nash equilibrium. To improve the performance of the agents, we trained a convolutional neural network offline in order to bias the sampling matrix. This algorithm yields great results and is easily scalable. The weakness of the algorithm is apparant when trying to plan deep into the future, as it becomes very large and it is not readily clear how to design the generation probabilities to yield the best results. One problem that arises when trying to plan deep into the future is that the agents require many more iterations in order to get close to the stationary distribution. When agents execute an action and move forward in the finite time horizon when the Markov chain is not well mixed, performance can actually be worse than if agents were to not recycle their solutions and use our suggested sampling matrix. More work can be done in the future to determine how the sampling matrix can be designed when given a fixed number of iterations that the agents can take during every step in the finite time horizon. Furthermore, in the future, machine learning could be utilized to determine optimal sampling matrices that gives the system of agents the greatest possibility of reaching the optimal Nash equilibrium within a time limit.

In Chapter 5, we are inspired by many recent algorithms that modify upper confidence bound tree search by influencing the action selection stage with a recommendation from a neural network, which we train using `MIPC`. By influencing the action selection, we are able to more efficiently sample the action space of the

agents in order exploit better actions more often. This is especially important in the multi-agent case because the joint action space becomes very large and it is imperative to sample efficiently. To fully utilize the imitation learned policy in our action selection stage, we map the state of the environment to a personalized image for every agent. We use a convolutional neural network that is trained to classify the input personalized images to a probability distribution (labels) which match previous successful action selections in upper confidence bound tree search. The motivation behind utilizing convolutional neural networks is two-fold. First, mapping the state of the environment and agents to an image is a method in which we can regulate the dimensions of the state space. Second, most of the multi-agent deployments are spatial by nature and convolutional neural networks specialize in recognizing spatial features that may indicate whether or not an action is preferable to take. In addition to the use of the imitated policy as a bias for the action selection, we have modified the action selection process to cater multiple agents by having them select in a sequential fashion. This sequential action selection modifies the Markov decision process that was originally defined. In analysis we show that convergence properties of the upper confidence bound tree search are maintained. `CATS` is trained offline to be used online. One problem that arises when trying to implement `CATS` is the required time needed for both creating synthetic data and for training the convolutional neural network. Fortunately, the data creation is a process that can be fully parallelized. In the future, we would like to implement `CATS` on UxVs and take advantage of recent advances in computational power. Microprocessers and edge-compute devices, such as TPUs, are now able to compute large neural networks. `CATS` requires many forward passes through its convolutional neural network. In order to implement `CATS` on a

real device it is imperative that the neural network is optimized for both accuracy and speed.

Chapter 7 is coauthored with Cortés, Jorge. The dissertation author was the primary author of this chapter.

Bibliography

[ACBF02] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[AmCA11] A. A. Agha-mohammadi, S. Chakravorty, and N. M. Amato. FIRM: Feedback controller-based information-state roadmap-a framework for motion planning under uncertainty. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 4284–4291, San Francisco, CA, 2011.

[ATB17] T. Anthony, Z. Tian, and D. Barber. Thinking fast and slow with deep learning and tree search. In *Conference on Neural Information Processing Systems*, pages 5360–5370, 2017.

[B+93] L.E. Blume et al. The statistical mechanics of strategic interaction. *Games and economic behavior*, 5(3):387–424, 1993.

[BBKT17] A. A. Bian, J. M. Buhmann, A. Krause, and S. Tschiatschek. Guarantees for greedy maximization of non-submodular functions with applications. In *International Conference on Machine Learning*, volume 70, pages 498–507, Sydney, Australia, 2017.

[BCK+07] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, 37(2):653–670, 2007.

[BCM09] F. Bullo, J. Cortés, and S. Martinez. *Distributed Control of Robotic Networks*. Applied Mathematics Series. Princeton University Press, 2009.

[BCP+19] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch. Dec-MCTS: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

[Bel13] R. Bellman. *Dynamic programming*. Courier Corporation, 2013.

[Ber95] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[BNS08] F. Broz, I. Nourbakhsh, and R. Simmons. Planning for human-robot interaction using time-state aggregated POMDPs. In *AAAI*, volume 8, pages 1339–1344, 2008.

[Bou96] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[Bou99] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the 16th international joint conference on Artifical intelligence (IJCAI)*, volume 1, pages 478–485, 1999.

[BPW+12] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[BSR16] A. Bai, S. Srivastava, and S. Russell. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence, IJCAI*, pages 3029–3039, New York, NY, 2016.

[BTN98] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Math. Oper. Res.*, 23:769–805, 1998.

[CABP16] A. Clark, B. Alomair, L. Bushnell, and R. Poovendran. *Submodularity in Dynamics and Control of Networked Systems*. Communications and Control Engineering. Springer, New York, 2016.

[CE17] J. Cortés and M. Egerstedt. Coordinated control of multi-robot systems: A survey. *SICE Journal of Control, Measurement, and System Integration*, 10(6):495–503, 2017.

[CGP09] M. C. Campi, S. Garatti, and M. Prandini. The scenario approach for systems and control design. *Annual Reviews in Control*, 32(2):149–157, 2009.

[CMKJ09] A.C. Chapman, R.A. Micillo, R. Kota, and N.R Jennings. Decentralised dynamic task allocation: a practical game: theoretic approach. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 915–922. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[DK11] A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *CoRR*, February 2011.

[DLM09] H. Daumés, J. Langford, and D. Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009.

[DM12] M. Dunbabin and L. Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics & Automation Magazine*, 19(1):24–39, 2012.

[DPH+15] J. Das, F. Py, J. B. J. Harvey, J. P. Ryan, A. Gellene, R. Graham, D. A. Caron, K. Rajan, and G. S. Sukhatme. Data-driven robotic sampling for marine ecosystem monitoring. *The International Journal of Robotics Research*, 34(12):1435–1452, 2015.

[GM04] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[GOL98] L. El Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998.

[GS07a] S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning*, pages 273–280. ACM, 2007.

[GS07b] P. R. Goundan and A. S. Schulz. Revisiting the greedy approach to submodular set function maximization. *Optimization online*, pages 1–25, 2007.

[GSL+14] X. Guo, S. Singh, H. Lee, R.L. Lewis, and X. Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Conference on Neural Information Processing Systems*, pages 3338–3346, 2014.

[Haj88] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[HF00] E. A. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *International Conference on Artificial Intelligence Planning Systems*, pages 130–139, Breckenridge, CO, 2000.

[HGEJ17] A. Hussein, M.M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

[HLKT19] P. Hernandez-Leal, B. Kartal, and M.E. Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[How60] R.A. Howard. *Dynamic programming and Markov processes*. M.I.T. Press, 1960.

[KGKG15] B. Kartal, J. Godoy, I. Karamouzas, and S. J. Guy. Stochastic tree search with useful cycles for patrolling problems. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1289–1294. IEEE, 2015.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KS06] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, volume 6, pages 282–293. Springer, 2006.

[KSW06] L. Kocsis, C. Szepesvári, and J. Willemson. Improved Monte-Carlo search. 2006.

[LA87] P.J.M.V Laarhoven and E.H.L Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[LK00] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on Algorithmic Foundations of Robotics*, pages 293–308, Dartmouth, NH, March 2000.

[Lov91] W. S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[MAS09] J. R. Marden, G. Arslan, and J. S. Shamma. Cooperative control and potential games. *IEEE Transactions on Systems, Man & Cybernetics. Part B: Cybernetics*, 39:1393–1407, 2009.

[MB96] A. K. McCallum and D. Ballard. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester. Dept. of Computer Science, 1996.

[MC16] A. Ma and J. Cortés. Visibility-based distributed deployment of robotic teams in polyhedral terrains. In *ASME Dynamic Systems and Control Conference*, Minneapolis, MN, October 2016. DSCC2016-9820.

[MC19] A. Ma and J. Cortés. Distributed multi-agent deployment for full visibility of 1.5D and 2.5D polyhedral terrains. *Journal of Intelligent and Robotic Systems*, 2019. Submitted.

[ME10] M. Mesbahi and M. Egerstedt. *Graph Theoretic Methods in Multiagent Networks*. Applied Mathematics Series. Princeton University Press, 2010.

[MGPO89] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21(1):59–84, 1989.

[MKS+15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[MOC17] A. Ma, M. Ouimet, and J. Cortés. Dynamic domain reduction for multi-agent planning. In *International Symposium on Multi-Robot and Multi-Agent Systems*, pages 142–149, Los Angeles, CA, 2017.

[MOC19] A. Ma, M. Ouimet, and J. Cortés. Cooperative dynamic domain reduction. In *Distributed Autonomous Robotic Systems: The 14th International Symposium*, volume 9 of *Springer Proceedings in Advanced Robotics*, pages 499–512. Springer, New York, 2019.

[MOC20a] A. Ma, M. Ouimet, and J. Cortés. Exploiting bias for cooperative planning in multi-agent tree search. *IEEE Robotics and Automation Letters*, 2020. To appear.

[MOC20b] A. Ma, M. Ouimet, and J. Cortés. Hierarchical reinforcement learning via dynamic subspace search for multi-agent planning. *Autonomous Robots*, 2020. To appear.

[MS96] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[NWF78] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[OA16] F. A. Oliehoek and C. Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer, New York, 2016.

[OAmAH15] S. Omidshafiei, A. A. Agha-mohammadi, C. Amato, and J. P. How. Decentralized control of partially observable Markov decision processes using belief space macro-actions. In *IEEE Int. Conf. on Robotics and Automation*, pages 5962–5969, Seattle, WA, May 2015.

[PR98] R. E. Parr and S. Russell. *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley Berkeley, CA, 1998.

[PR10] S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Robotics Research*, pages 293–305. Springer, 2010.

[PRHK17] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6059–6066. IEEE, 2017.

[PT87] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[Put14] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[RCN18] F. Riccio, R. Capobianco, and D. Nardi. Q-CP: learning action values for cooperative planning. In *IEEE Int. Conf. on Robotics and Automation*, pages 6469–6475, Brisbane, Australia, 2018.

[SAH+19] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[SHM+16] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[SK06] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, 2006.

[SLA+15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, Lille, France, 2015.

[SPS99] R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[SSS95] J.P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[SSS+17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[SWD+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[TK04] G. Theocharous and L. P. Kaelbling. Approximate planning in POMDPs with macro-actions. In *Conference on Neural Information Processing Systems*, pages 775–782, 2004.

[WMG+17] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Conference on Neural Information Processing Systems*, volume 30, pages 5285–5294, 2017.

[You98] H. P. Young. *Individual Strategy and Social Structure: an Evolutionary Theory of Institutions*. Princeton University Press, 1998.