Multiagent Planning and Learning Using Random Decompositions and Adaptive Representations

by

Nazim Kemal Ure
M.Sc., Defense Technologies
Istanbul Technical University (2010)
B.Sc., Aeronautical Engineering
Istanbul Technical University (2008)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2015

Signature redacted

Author ....
Department of Aeronautics and Astronautics
January 29, 2015

Signature redacted

Certified by ....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Certified by ... .
Girish Chowdhary
Assistant Professor of Mechanical and Aerospace Engineering

Certified by Signature redacted ....
Sertac Karaman
Assistant Professor of Aeronautics and Astronautics

Accepted by........ Signature redacted.......................
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Multiagent Planning and Learning Using Random Decompositions and Adaptive Representations

by

Nazim Kemal Ure

Submitted to the Department of Aeronautics and Astronautics
on January 29, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Multiagent planning problems are ubiquitous in engineering. Applications range from control of robotic missions and manufacturing processes to resource allocation and traffic monitoring problems. A common theme in all of these missions is the existence of stochastic dynamics that stem from the uncertainty in the environment and agent dynamics. The combinatorial nature of the problem and the exponential dependency of the planning space on the number of agents render many of the existing algorithms practically infeasible for real-life applications. A standard approach to improve the scalability of planning algorithms is to take advantage of the domain knowledge, such as decomposing the problem to a group of sub-problems and exploiting decouplings among the agents, but such domain knowledge is not always available. In addition, many existing multiagent planning algorithms rely on the existence of a model, but in many real-life situations models are often approximated, wrong, or just unavailable. The convergence rate of the multiagent learning process can be improved by sharing the learned models across the agents. However, many realistic applications involve heterogeneous teams, where the agents have dissimilar transition dynamics. Developing multiagent learning algorithms for such heterogeneous teams is significantly harder, since the learned models cannot be naively transferred across agents. This thesis develops scalable multiagent planning and learning algorithms for heterogeneous teams by using embedded optimization processes to automate the search for decouplings among agents, thus decreasing the dependency on the domain knowledge.

Motivated by the low computational complexity and theoretical guarantees of the Bayesian Optimization Algorithm (BOA) as a meta-optimization method for tuning machine learning applications, the developed multiagent planning algorithm, Randomized Coordination Discovery (RCD) extends the BOA to automate the search for finding coordination structures among the agents in Multiagent Markov Decision Processes. The resulting planning algorithm infers how the problem can be decomposed among agents

based on the sampled trajectories from the model, without needing any prior domain knowledge or heuristics. In addition, the algorithm is guaranteed to converge under mild assumptions and outperforms the compared multiagent planning methods across different large-scale multiagent planning problems.

The multiagent learning algorithms developed in this thesis use adaptive representations and collaborative filtering methods to develop strategies for learning heterogeneous models. The goal of the multiagent learning algorithm is to accelerate the learning process by discovering the similar parts of agents transition models and enable the sharing of these learned models across the team. The proposed multiagent learning algorithms Decentralized Incremental Feature Dependency Discovery (Dec-iFDD) and its extension Collaborative Filtering Dec-iFDD (CF-Dec-iFDD) provide improved scalability and rapid learning for heterogeneous teams without having to rely on domain knowledge and extensive parameter tuning. Each agent learns a linear function approximation of the actual model, and the number of features is increased incrementally to automatically adjust the model complexity based on the observed data. These features are compact representations of the key characteristics in the environment dynamics, so it is these features that are shared between agents, rather than the models themselves. The agents obtain feedback from other agents on the model error reduction associated with the communicated features. Although this process increases the communication cost of exchanging features, it greatly improves the quality/utility of what is being exchanged, leading to improved convergence rate.

Finally, the developed planning and learning algorithms are implemented on a variety of hardware flight missions, such as persistent multi-UAV health monitoring and forest fire management scenarios. The experimental results demonstrate the applicability of the proposed algorithms on complex multiagent planning and learning problems.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

First of all, I would like to thank my advisor Jonathan P. How for being an excellent mentor throughout my years at MIT. His views on how to perform research and write papers had a huge influence on me. I feel lucky for being a part of his research team. I would also like to thank my committee members Girish Chowdhary and Sertac Karaman. Girish, whom I also had the pleasure to co-author a few papers, always pointed me on the right track when I got lost and has been a very supportive friend. Sertac, whom I had known for years, has been very helpful and asked a lot of interesting questions that made me think about how to structure my research. He also has been a role model for me on how to be a successful young academician. I am very grateful for knowing him. I would also like to thank Jared Speck, my Mathematics minor advisor, whom provided me with valuable guidance during my studies. I would also like to thank Boeing Research and Technology for funding my PhD studies.

Without my wonderful labmates, the time at MIT would have been much more stressful and overwhelming. I would like to thank Tuna Toksoz for being my closest friend, room mate and colleague in my first 2 years at MIT. Josh Redding and Mark Cutler have also been great friends to me. Alborz Geramifard has been more than helpful when I was starting to develop my own research, I can't thank him enough for the lengthy discussions we had. I would like to thank Trevor Campbell and Luke Johnson for the challenging technical discussions we had. I will also remember the meals and chats with Vishnu Deseraju, Sameera Ponda, Georges Aoude, Buddy Michini and Dan Levine with much joy. Also thanks to Tom Walsh, Christopher Amato and Ali-agha Mohammadi for sharing their wisdom and experience with me and providing valuable comments on my research.

I would like to thank my friends at Boston and Istanbul for the amazing time we had and the support they provided through tough times. I think Pelin Arslan knows me better than anyone, she is the most amazing friend one can have. Same goes for Fulden Buyukozturk, who is one of the most interesting persons I have ever met. I am so glad that I shared a flat in my last 6 months at Boston with Emre Koyuncu, who is a great friend and one of the funniest and wisest guys I ever met. And of course, I would like to thank Asli Sirman, who never ceases to amaze me, she is my beacon of happiness and holds a very special place in my heart. A big thanks also goes to my friends in Turkey, Alper Karahan, Onur Saydam, Berker Altepe, Ufuk Cetinkaya and Merve Akca, you guys have been very supportive of me and thanks for all the amazing time we spent during my vacations in Turkey.

While doing my PhD, I have been fortunate enough to pursue an alternative career as a musician. A huge thanks goes to Your Pain is Endearing and Achroma Rising fans, as well as all the people who viewed and shared my bass guitar videos. You guys are amazing! I would also like to thank my bandmates, Jason Eick, Matt Philippo, Justin Brink, Anthony Simone and Rob Robles, traveling with these guys all over the country and playing music was an unforgettable experience. I would also like to thank Ahmet Saracoglu and Eren Basbug for all the awesome time we had and the music we wrote/played together.

I also want to mention that my previous advisor at Istanbul Technical University Gokhan Inalhan has been the single most important person in shaping my academic career. Without his guidance I don't know where would I be now. I also would like to thank Ibrahim Ozkol for his valuable advice and wonderful courses he taught.

Finally, I would like to thank my family for their love and support. I am so lucky to have Cihan Deniz Ure as my brother, I would like to thank him for all the courage and support he provided over the years. I would like to thank my parents Dilek Ure and Mehmet Celal Ure for being the most amazing parents ever. They always encouraged me to pursue my dreams, whether it is academia or music, I am where I am because of them. Words cannot express how grateful I am to them.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

The design and study of multiagent systems is a widely acknowledged interdisciplinary engineering problem, drawing attention of researchers from distinct fields, such as computer science [1], electrical engineering [2], aerospace engineering [3] and operations research [4]. The engineering of multiagent systems for cooperative scenarios can be defined as the development of methodologies for coordinating multiple entities towards achieving a common goal. An important subset of the multiagent design problem is addressing planning and learning problems. The planning problem involves optimizing the behavior of agents in an uncertain environment based on the known model of the mission dynamics. On the other hand, the learning problem consists of modelling the dynamics of the environment by processing observations collected by individual agents.

Both of these problems have common and distinct challenges, such as the scalability of the algorithms with the number of agents, development of the data and model sharing protocols under communication constraints and the automation of the model selection process. In the subsequent sections we motivate the study of multiagent systems by focusing on real world applications (Section 1.1), then we give the technical definitions of the multiagent planning and learning problems (Section 1.2). The Section 1.3 identifies some of the critical challenges inherent to these problem formulations. The Section 1.4 presents the literature review of the related work and emphasizes the existing gaps. Finally, the Section 1.5 presents the outline of the research conducted for addressing these existing gaps and the contributions of the thesis.

1.1 Motivation

Many robotic missions involve teams of mobile robots operating in an uncertain environment with the objective of achieving a common goal or trying to maximize a joint reward.

Examples include planetary science missions [5] , cooperative pursuit evasion [6], persistent surveillance with multiple Unmanned Aerial Vehicles (UAVs) [7], reconnaissance missions [8], and supply chain event management [9]. Cooperation between agents is a recurring theme in these missions. For instance, in a multi-robot exploration mission, individual agents should coordinate their actions based on the actions of other agents to efficiently cover the search domain. Another example is the cooperative pursuit evasion mission, where agents should collaboratively decide their positions to limit an evader's maneuvers.

Another line of multiagent system applications involve stationary agents operating in a highly dynamic environment, where the agents must execute actions that regulate the environment dynamics. Examples include distributed vehicle monitoring [10], air traffic control [11], network management [12], electricity distribution management [13] and meeting scheduling [14]. There are also applications in which the mobile and stationary agents must work together, such as control of manufacturing processes [15]. Similar to the missions with mobile agents, cooperation between agents is critical in these applications. Consider a multiagent traffic flow regulation problem [16], in which each agent controls a traffic light at an intersection and the objective is to maximize the flow rate of the mobile vehicles while trying to minimize the traffic jams and the number of collisions. Non-cooperative solutions are simply unacceptable in a such domain, since collisions are very likely to happen if the traffic lights are not coordinated.

Technically all these missions can be formulated as stochastic sequential decision making problems. The uncertain elements of the missions may stem from many different sources, such as the impact of external disturbances on the dynamical model of the robot [17], the rate change and amplitude of package requests in a network [18], or the uncertainty in the target motion model in a cooperative target tracking scenario [19]. Hence planning algorithms require a model of the uncertainty in order to hedge against the stochastic dynamics of the environment. However, in many real world applications, such models may not be available *a priori* and mismatch between the planning model and the actual dynamics of the environment might lead to catastrophic performance [20]. Hence, models need to be learned/updated during the mission to mitigate the effects of the model mismatch. Thus in real word applications, the planning problem cannot be separated from the model learning problem, these two classes of algorithms should work harmoniously.

To summarize, it can be concluded that both multiagent planning and learning problems arise in many technological applications and the integrated solutions to such problems should be investigated to improve the quality of these implementations and to open doors to new applications.

1.2 Problem Statement

This Section gives a brief technical formulation of the planning and learning problems of interest. A deeper technical discussion of these problems and tools for analyzing them will be provided in Chapter 2.

1.2.1 Multiagent Planning Problem

Markov Decision Processes (MDPs) [21] are a common framework for analyzing stochastic decision making problems. We will utilize this framework to study multiagent planning problems.

An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the transition model, $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is the reward model and $\gamma \in [0,1)$ is a discount factor. $\mathcal{T}(s,a,s')$ denotes the probability of ending up at state $s' \in \mathcal{S}$ given that the current state and action is $s \in \mathcal{S}, a \in \mathcal{A}$. Multiagent Markov Decision Processes (MMDPs) [22] are specific types of MDPs that involve multiple agents, where the action space is factorized as $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \cdots \times \mathcal{A}_{n_a}$ and n_a is the number of agents. The objective of the planning problem is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ such that the cumulative discounted reward is maximized for given initial state s^0,

$$\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r^k | s^0, a^k = \pi(s^k)\right], \tag{1.1}$$

where k is the discrete time step, $\mathbb{E}[.]$ is the expectation operator and the expectation is taken over the distribution of next states s^{k+1}. Further information on the assumptions on the structure of the MMDP formulation and obtaining a policy that maximize the Eq. 1.1 will be provided in Chapter 2.

1.2.2 Multiagent Learning Problem

In this thesis it is assumed that the reward model $\mathcal{R}$ of the MDP is available to the designer. The multiagent learning problem consists of estimating the unknown transition model $\mathcal{T}$ from the observed state transitions. In many multiagent applications of interest, the agents cannot directly change the states of the other agents, that is, the transition models are independent[23]. The coupling between agents usually enters the system through the reward model. We will assume that the transition dynamics are independent among the agents and the transition model can be factorized as $\mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2 \cdots \times \mathcal{T}_{n_a}$. Each agent maintains and updates it's own estimate of the individual transition model $\tilde{\mathcal{T}}_i$. Let $o^k = \{s^k, a^k, s^{k+1}\}$ denote the state, action and next state at time k. The learning process consists of two stages; the estimation step,

$$\tilde{\mathcal{T}}_i^k \leftarrow e_i(\tilde{\mathcal{T}}_i^{k-1}, o^k) \quad (1.2)$$

where e_i is the estimation law that updates the estimation $\tilde{\mathcal{T}}_i^k$ based on the past estimation $\tilde{\mathcal{T}}_i^{k-1}$ and the current transition observation o^k. The second stage is the communication step, where each agent broadcasts their estimated models by using a communication function c_i,

$$c_i(\tilde{\mathcal{T}}_i^k), \quad (1.3)$$

where the output of the c_i might be saturated due to the constraints on the throughput of the communication network [24]. For a variety of different multiagent communication architectures also see [23].

The objective of the learning problem is to design the estimation laws e_i and the communication laws c_i such that the estimates $\tilde{\mathcal{T}}_i^k \to \mathcal{T}_i$ as $k \to \infty$, preferably with an exponential rate.

1.3 Challenges

Planning and learning with MMDPs involve a variety of challenges [25]. This thesis addresses three particular challenges that are critical to the design of multiagent planning and learning algorithms.

1. *Scalability:* The "curse of dimensionality" is a widely recognized problem in machine learning and planning research, which refers to the exponential blow-up in the computational complexity of the problem with the increased dimensionality [26]. Multiagent systems inherently suffer from this problem, because the joint size of the state space is almost always exponential in the number of agents. In the language of Section 1.2, this refers to the fact that the cardinality of $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{T}$ being exponential in n_a. For instance, consider a mission where the agents operate in a gridworld and let the state of each agent be represented by it's current location. The size of the state space is simply m^{n_a}, where m is the number of grids. Hence the problem size increases exponentially in the number of agents. Many existing MDP solvers [27] and transition model estimators [28] have polynomial complexity in the size of the state-action space, which corresponds to having exponential complexity in the number of agents, rendering these algorithms computationally intractable for large-scale problems. This is a crucial limitation for the application of multiagent planning and learning algorithms

to real world problems.

2. *Heterogeneous Teams:* Homogeneous teams refer to the multiagent problems where the agents have the same or similar transition dynamics, hence $\mathcal{T}_i \approx \mathcal{T}_j, i, j = 1, \ldots, n_a, \imath \neq j$. This assumption usually simplifies both the planning and the learning problem significantly. For instance, the function approximation methods [29] and dimensionality reduction methods [30] work much more efficiently for the multiagent planning problems with homogeneous teams. Similarly, solving multiagent learning problems become significantly easier, since all the agents are trying to estimate the same parameters. The bulk of the literature in multiagent machine learning examines the problem of multiple agents with homogeneous transition dynamics trying to infer a single model [31]. However, many real world applications require the agents to learn different or agent-dependent models, such as the impact of external disturbances on the agent health [32] or modelling of the arrival rate of packages in a specific part of the network [33]. In such problems, the learning process might be accelerated substantially by sharing information/models among agents. Learning with heterogeneous teams is far more challenging, because it involves the need to assess similarity of the learned models among agents and transferring the relevant parts of the individual models across the team to improve the convergence rate of the learning process.

3. *Dependency on Domain Knowledge and Manual Tuning:* Many planning and learning algorithms contain parameters that must be tuned to obtain a good performance in a specific domain. These parameters can be as simple as a scalar learning rate in a reinforcement learning algorithm [34] and sometimes they can be as complex as a set of arbitrary basis functions for an approximation algorithm [35]. In many applications, these parameters are set by domain experts based on the heuristics and domain knowledge obtained from past experiences. However, many algorithms contain a large number of parameters that require an extensive tuning process that could be beyond the capabilities of a domain expert, even if they are available. Furthermore, the number of required parameters typically increases with the complexity of the problem, so the simple parameter tuning methods might become time consuming. Thus, it is desirable to automate the parameter selection process for the planning and learning algorithms without relying on extensive domain knowledge.

1.4 Literature Review

This Section gives a review of the related work on planning and learning for multiagent systems. Since the literature on the subject is rather large, review will mainly focus on algorithms that are concerned with addressing challenges described in Section 1.3.

Related Work on Centralized Multiagent Planning with MDPs

Dynamic Programming (DP) [36] is one of the most popular methods for solving an MDP, which involves computing the value function as a solution to the Bellman equation [37]. Exact DP methods, such as value iteration and policy iteration [38] work by sweeping through the planning space by performing Bellman backups and they are guaranteed to converge to the optimal solution asymptotically [39]. However, for large-scale planning spaces, such as those associated with multiagent missions, both memory and computational complexity increases exponentially in the number of dimensions, which renders the exact approaches infeasible for such domains. In the last 20 years, significant effort has been exerted to develop approximate algorithms that can yield near-optimal solutions with feasible computation and memory requirements [29]. For example, Asynchronous Dynamic Programming aims to lower the computational complexity by performing planning on a subset of the planning space rather than sweeping the whole space, which can be performed by selecting randomized subsets [39] of the planning space or sampling trajectories from the model [40]. However, sampling trajectories from a large-scale planning space is a non-trivial problem, since subsets of the planning space that are more relevant to the optimal policy are not known *a priori*. Hence, the trajectories sampled from the model should *explore* the state space in an efficient manner, that is the algorithm should not get stuck to a specific subset of state-space but should take exploratory actions to move less frequently visited parts, while also taking actions that exploit the current policy to improve the plan. The problem of finding a balance between such actions is usually referred to as the *exploration-exploitation* dilemma [28]. Many different approaches were developed to handle this dilemma, such as randomized exploration [29] and incorporation of knownness and memory functions [41]. However, handling exploration in large-scale planning spaces is still an open problem.

Approximate Dynamic Programming (ADP) [29] methods were developed to address the issue of scalability by approximating the value/policy function by a finite set of basis functions, so that the approximate problem has fewer unknown parameters compared to the original problem. Once the basis set is chosen, approximate versions of value and policy iteration [39] can be applied to obtain a sub-optimal solution. Linear function approximation [35] methods received particular attention due to their theoretical guarantees [42]. Nonlinear approximators such as radial basis functions [29] and neural networks [43] have also been studied because of their capabilities to approximate a large class of functions. Note however that ADP methods mentioned so far require the designer to hand-code the set of approximation basis, and such information usually relies on domain expertise.

Recently, significant effort has been applied into automating the basis function selection process based on the observed/simulated data. Examples are, adaptive tile coding [44], incremental feature dependency discovery [45], orthogonal matching pursuit [46] and kernel based approximators [47]. Overall, these approximate techniques were shown to improve the scalability and convergence rate substantially for a wide variety of MDPs and relaxed the constraints on specifying a fixed set of basis functions a priori. However, in practice these methods do not scale well to large-scale multiagent missions, because the process of basis function automation slows down significantly in large-scale planning spaces.

Structural MDP decomposition algorithms take advantage of the structure of the problem formulation to lower the size and dimension of the planning space to enable faster computation time [48]. This process usually involves investigating the factorization of the state-action space and then exploiting the independence between these factors. Factored MDPs [49] represent the original MDP with a Dynamic Bayesian Network [50] (DBN), where the state transitions are represented in a more compact manner to avoid exponential blow-up of the planning space. Both policy iteration [51] and linear programming approaches [52] were developed for solving factored MDPs efficiently. A highly scalable multiagent planning algorithm with factored MDPs was proposed by Guestrin and Parr [53], where each agent solves its individual MDP and the joint return optimization is achieved via use of a coordination graph. The main drawback of the aforementioned approaches is the assumption that the structure of the DBN and the coordination graph is known to the designer. Recent research has focused on estimating the structure of factored MDPs, which is still an active area of research [54]. Ref. [55] demonstrated that statistical tests can be applied to learn structure of factored MDPs and [56] showed that Dynamical Credal Networks can be used as a tool to estimate factored MDP structures. Although not studied from multiagent planning point of view, Doshi [57] applied nonparametric inference techniques to learn structure of DBNs. Kok and Valassis [58] proposed an algorithm for learning the structure of the coordination graphs greedily based on statistical tests, however no theoretical analysis was done and the algorithm was shown to be effective for only a small number of moderately sized problems.

Another popular MDP structure decomposition technique is inducing hierarchies. The work in [59] shows how a large-scale MDP can be decomposed into a hierarchy of sub-MDPs, where sub-MDPs at the bottom of the hierarchy can be solved by neglecting states of MDPs at the top. Several works extended this formulation [60] and developed efficient algorithms than can solve hierarchical MDPs much faster compared to the original flat MDP. The options framework [61] is closely related to the hierarchical MDPs, where a collection of actions and state transitions are abstracted to entities that can be executed as macro actions. Both planning with options [62] and learning options [63, 64] attracted many researchers in recent years, however automating the discovery of such structures is still an open problem.

It should also be noted that, we are interested in solving centralized multiagent planning problems in this Thesis. There is a bulk literature on decentralized multiagent planning (See [23, 65, 66]). These problems come with additional challenges, such as partial observability of the agent states and enforcing the policy consensus among agents. Although decentralized planning is a very interesting research direction, these problems are beyond the scope of this Thesis.

Also, it should be noted that there are methods that can construct a policy without the knowledge of a model. These methods are usually referred to as model-free reinforcement learning algorithms [28, 29]. These algorithms process the state transition samples collected from the interactions with the environment and then compute a policy without explicitly learning the MDP model. However in practice, compared to the model-based approaches, these algorithms usually need a lot of samples to converge. In addition, model-based approaches also end up learning the model of the MDP along with the policy, which can be used for other purposes. Hence, we favor the model-based approaches over the model-free approaches. That is, we are interested in developing planning/learning architectures that first learns a model of the MDP from the transition samples and then use the learned model to compute a policy. In the next subsection we review the multiagent model learning methods.

Related Work on Multiagent Learning

Fusion of multiagent systems and machine learning methods is an active area of research. A general survey on these methods can be found in [67]. Classical multiagent learning literature focuses on the problem of a group of agents trying to learn a single model, which can be formulated as a parameter/model consensus problem [31]. This problem has been investigated extensively in different settings and efficient algorithms in Bayesian framework for both perfect and limited communication have been developed [68, 69]. However, extensions to the heterogeneous learning setting have not been much investigated. Game Theory is a popular framework to study multiagent systems [70], flexibility of the game theoretic modeling framework enabled a thorough analysis of both homogeneous and heterogeneous team learning in cooperative and competitive scenarios [71, 72]. However, studies have mainly focused on the theoretical aspects, such as convergence to Nash equilibria and practical aspects such as scalability and ability to work under limited communication were largely ignored. Homogeneous learning have been investigated by application of evolutionary computation techniques [67], important results include applications to team-mate modeling [73], competitive learning [74] and credit assessment [75]. Overall, algorithms demonstrated good scalability properties. However, heterogeneous learning and performance under limited communication have not been much investigated. Multiagent Reinforcement learning (MARL) algorithms solve multiagent planning problems without the knowledge of the model beforehand. A general survey on multiagent reinforcement learning can be found on [76]. MARL algorithms typically learn a joint model across the agents, hence the model heterogeneity challenge is not explicitly addressed.In addition, many MARL algorithms usually suffer from scalability issues due to exponential blowup of the learning space in number of agents. Overall, MARL algorithms suffer from the same scalability issues that comes with curse of dimensionality, and many of the existing algorithms are strictly dependent on the domain knowledge and manual tuning.

Although they are not directly related to multiagent systems, fields of transfer learning and multi-task learning also study similar problems in terms of model transfer among different tasks. Transfer learning [77] is the study of how to carry a model learned from a specific problem to a similar unsolved problem in order to accelerate the learning in the new problem. Transfer learning is an active area of research and found many applications ranging from multi-task learning [78] and collaborative filtering [79] to reinforcement learning [80]. Although transfer learning algorithms are not developed specifically for solving heterogeneous multiagent learning problems, they can be used as a tool to study similarities between models learned by different agents and how to handle the sharing of models between agents in an online manner under communication constraints. To the best of our knowledge, no prior research exist on establishing the connection between online multiagent learning and transfer learning/collaborative filtering algorithms.

1.5 Overview of The Research and Contributions

The contributions of this Thesis focuses on addressing the challenges outlined in Section 1.3. The Thesis aims to develop planning and learning algorithms that are scalable to large-scale multiagent problems, applicable to teams with heterogeneous transition models and have embedded meta-optimization layers for self-tuning their parameters, so that the resulting algorithms have little or no dependency on the domain knowledge.

The first contribution of the thesis is the development of a planning algorithm that performs coordination structure search for solving large-scale MMDPs. This work develops a novel MMDP structure decomposition algorithm where the decomposition process is automated by the use of a probabilistic meta-optimization layer. The algorithm, named

Randomized Coordination Discovery (RCD), utilizes the framework of the Bayesian Optimization Algorithm [81], which guides the decomposition search by generating samples from probabilistic graphical models defined on the planning space. This meta-optimization layer automatically discovers the coordination structures that would lead to computationally feasible plans with high returns. Regarding the challenges in the multiagent planning, this thesis makes the following contributions,

- **Scalability:** It is shown that the RCD algorithm has linear complexity in the number of agents in the problem and the largest factor in the coordination factor graph per iteration. This property makes the algorithm scalable to large-scale multiagent missions, contrary to the many existing MDP solvers, which have exponential complexity in the number of agents.

- **Heterogeneous Teams:** The RCD algorithm does not make any symmetry assumptions in the coordination structure and performs a randomized search over all possible structures. Since heterogeneous teams are more likely to have an asymmetric coordination structure, this property makes the RCD algorithm suitable for working with heterogeneous teams.

- **Asymptotic Convergence:** It is proven that the RCD is guaranteed to converge asymptotically. This property is not always guaranteed for existing representation search algorithms and it ensures that the coordination search will not end up oscillating between different coordination structures.

- **Dependency on Domain Knowledge:** Unlike the existing decomposition based MMDP solvers, the RCD algorithm does not require the designer to encode a fixed coordination structure beforehand. The algorithm utilizes a randomized search process to discover the coordination structure that is most likely to result in a high return policy.

- **Comparative Simulation Results:** The simulation results show that, given the same computational resources, the RCD algorithm outperforms alternative decomposition based MMDP solvers up to an order of magnitude on several challenging large-scale multiagent planning problems.

The second contribution of the thesis is the development of a family of distributed learning algorithms that utilize feature transfer among agents to enable rapid learning under communication constraints. The algorithms decompose the learning problem across the team and enable agents to exchange partial models with each other to improve the convergence rate of the learning process. The algorithms, named Decentralized Incremental Feature Dependency Discovery (Dec-iFDD) and Collaborative Filtering Decentralized Incremental Feature Dependency Discovery (CF-Dec-iFDD), utilize the iFDD algorithm [35] for automated model complexity adjustment. Each agent learns a linear function approximation of the actual model, and the number of features is increased incrementally to adjust the model complexity based on the observed data. These features are compact representations of the key characteristics in the environment dynamics, so it is these features that are shared between agents, rather than the models themselves. Regarding addressing the challenges in multiagent learning, this Thesis makes the following contributions,

- **Scalability:** The developed algorithms decompose the multiagent learning problem into a group of single agent learning problems that are executed concurrently. Hence the algorithm does not have any exponential dependency on the number of agents, making it applicable to large-scale scenarios.

- **Heterogeneous Teams:** The CF-Dec-iFDD algorithm makes no assumption on the homogeneity of the models of the agents. Instead, each agent maintains a partial estimate of other agent's models, which are built by receiving feedback on from other agents. The transferred features are selected based on these partial models, which might be different for every agent. Hence, the algorithm can work with heterogeneous teams, which contrasts with most other multiagent learning methods that assume that the agents are solving the same learning problem.

- **Dependency on Domain Knowledge:** The developed algorithms are built upon the iFDD algorithm, which starts the learning process with a small set of binary features and builds up a more complex representation by taking conjunctions of these initial features. The conjunctions are added only whenever the observed error exceeds a pre-specified threshold, hence the model complexity is automatically adjusted based on the gathered observations. Thus, the designer does not need to hand-code a fixed set of features for each agent's representations; iFDD discovers these features on the fly. This property of iFDD renders Dec-iFDD and CF-Dec-iFDD less dependent on the domain knowledge and free of extensive manual parameter tuning. Note that earlier iFDD was criticized for being dependent on a set of given initial features and the computational complexity does not scale well with the number of agents. These problems does not occur in the developed algorithms, since the multiagent learning problem is decomposed into a collection of single agent learning problems. There exists guidelines for specifying a set of good initial features for the single agent systems [45]. In addition, iFDD was shown to be computationally efficient for the single agent learning problems [35].

- **Asymptotic Convergence:** Asymptotic convergence of the iFDD algorithm in the context of reinforcement learning was proven in previous work. The proofs in this Thesis extend these guarantees to the approximate model learning and shows that iFDD model learning will always converge under mild assumptions.

- **Comparative Simulations Results:** Numerical results on the thesis considers a variety of large-scale multiagent learning problems and compares the learning rate of the CF-Dec-iFDD, Dec-iFDD and alternative approaches. Results confirm that when the team is heterogeneous, CF-Dec-iFDD has the fastest convergence rate among the compared approaches.

The third and the final contribution of the thesis focuses on the experimental verification of the proposed planning and learning algorithms on realistic multi-UAV mission scenarios. Experimental verification of these algorithms poses several challenges, such as learning with measured data, coping with the environmental disturbances, and operating under real time constraints. This work involves integration of the developed planning and distributed learning algorithms and the implementation of this framework on multi-UAV missions that require persistence coverage, health management, communication relay and cooperation. The contribution is the verification and demonstration of the proposed algorithms on flight experiments to confirm that the developed framework is able to compute policies in real time and learn using measured data. Specifically, the thesis makes contributions to the following experimental subjects,

- **Long Endurance Missions:** A persistent multi-UAV surveillance mission is conducted. The experiment involves real time policy computation and learning of sensor failure dynamics via iFDD. With the aid of autonomous recharge stations that recharge/swap UAV batteries, the mission was kept running fully autonomous for 3 hours, enabling collection of sufficient data to learn sensor models.

- **Emulating wind Disturbance:** Another persistent multi-UAV surveillance mission with a mixture of real/virtual agents were conducted. Two vertical fans were mounted on the ceiling, which made an impact on the battery life of the real UAVs. By using the Dec-iFDD learning algorithm, UAVs were able to learn the coupling between the fuel dynamics and the fan locations, and modified their policy accordingly. The experiment demonstrated that the realism of flying outdoors and emulation of external disturbances can be brought into indoors via the use of vertical fans.

- **Emulating sensor noise:** A multi-UAV forest fire management experiment was conducted. With the help of ceiling mounted laser projectors, a forest fire animation with stochastic dynamics was animated on the testbed ground. A quadrotor UAV mounted with an onboard camera collected images of the fire and used the CF-Dec-iFDD algorithm to learn the fire dynamics. This experiment demonstrated that the sensor noise, such as the noise introduced by the camera, can be emulated in indoor experiments with the help of projected animations.

1.6 Layout of The Thesis

Technical background associated with the developed work is given in Chapter 2. Next, Chapter 3 provides the details on the development of the multiagent planning algorithm RCD, along with the theoretical and simulation results. Chapter 4 studies both single and multiagent learning with adaptive representations and provides details of the work on iFDD, Dec-iFDD and CF-Dec-iFDD algorithms, their convergence properties and simulation results. Chapter 5 gives an overall view of the MIT's RAVEN testbed and then provides experimental flight results for integrated planning/learning algorithms developed in this Thesis. Finally, Chapter 6 gives the concluding remarks and associated future work.

Chapter 2

Background

This Chapter gives a brief background on planning and learning in MMDPs along with an introduction to the Bayesian Optimization Algorithm (BOA), Incremental Feature Dependency Discovery (iFDD) and the Matrix Completion Algorithm. These planning and learning tools will serve as a basis for the algorithms that are going to be developed in the later chapters of the thesis.

2.1 Planning with Markov Decision Processes

This section presents the formal definitions of MDPs and MMDPs, as well as the basic algorithms to perform planning with these models.

2.1.1 Markov Decision Processes

A Markov Decision Process is a framework for formulating sequential stochastic decision making problems [21],

Definition 1 (Markov Decision Process (MDP)) *A Markov Decision Process (MDP) is a tuple,*

$$<\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R},\gamma>,$$

where, $\mathcal{S}$ is a finite set of state configurations, $\mathcal{A}$ is a finite set of actions, $\mathcal{T}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\rightarrow[0,1]$ is a transition model, $\mathcal{T}(s,a,s')$ denotes the probability of ending up in state $s'\in\mathcal{S}$, given that the agent was in state $s\in\mathcal{S}$ in the previous timestep and took action a, $\mathcal{R}:\mathcal{S}\times\mathcal{A}\rightarrow\mathbf{R}$ is a reward model, $\mathcal{R}(s,a)$ denotes the reward obtained by the agent at the current time step, $\gamma\in[0,1)$ is a discount factor that balances the future rewards versus present rewards. We *denote the discrete timestep by k, and let $s^k \in \mathcal{S}$ and $a^k \in \mathcal{A}$ denote the state and action at time k respectively.*

In particular, we will study Multiagent Markov Decision Processes (MMDPs), which are defined as follows,

Definition 2 (Multiagent Markov Decision Process (MMDP)) *A Multiagent Markov Decision Process (MMDP) is a tuple,*

$$< n_a, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma >,$$

where $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma >$ forms an MDP with the factorized action space,

$$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \mathcal{A}_{n_a},$$

where $\mathcal{A}_i$ denotes the local action sets of agents and $n_a \in \mathbf{Z}$ is the number of agents.

Remark 1 *An MMDP is just an MDP where the set of actions are factorized across multiple agents.*

In many real life scenarios, the state space and transition models of the agents are also factorized, which enables the design of more computationally efficient algorithms. This thesis focuses on a specific class of MMDPs referred to as factored MMDPs (F-MMDPs), which admits a such factorization. F-MMDPs are defined as follows.

Definition 3 (Factored Multiagent Markov Decision Process (F-MMDP)) *A Factored Multiagent Markov Decision Process (F-MMDP) is an MMDP that admits the following factorizations,*

Factored State Space: *The state space is factorized as follows*

$$\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_{n_a} \times \mathcal{S}_1^e \times \ldots \times \mathcal{S}_{n_e}^e,$$

where $\mathcal{S}_i$ is the individual set of states for the agent i, $\mathcal{S}^e = \mathcal{S}_1^e \times \ldots \times \mathcal{S}_{n_e}^e$ is the set of states for the external variables and n_e is the number of factors for the external variables.

Factored Transition Model: *The transition model is factorized as follows,*

$$\begin{aligned}
\mathcal{T} &= \mathcal{T}_1 \times \ldots \times \mathcal{T}_{n_a} \times T^e, \quad \textit{where}, \\
T^e &= \mathcal{T}_1^e \times \ldots \times \mathcal{T}_{n_e}^e, \\
\mathcal{T}_i &: (\mathcal{S}_i \times \mathcal{S}^e) \times \mathcal{A}_i \times \mathcal{S}_i \to [0,1], i = 1, \ldots, n_a
\end{aligned}$$

$$\mathcal{T}_i^e \quad : \quad \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1], i = 1, \ldots, n_e.$$

Remark 2 *In an F-MMDP, the transition model of individual agents states depend only on the local state, local action and the external states. Whereas the dynamics of the external states might depend on the both external states and the state of the agents.*

Example 2.1.1 *Consider a multiagent multi-target tracking problem, where a group of agents try to minimize the total distance between them and multiple targets with stochastic dynamics. This problem can be cast as an F-MMDP, where $\mathcal{S}_i$ is the location the agent i and $\mathcal{S}_j^e$ is the location of the target j. Note that the next state distribution of each agent depends only on its own position and action, whereas the next state distribution of the each target might depend on their local states plus states of the tracking agents.*

Next we define the trajectory, policy, return and value function,

Definition 4 (Trajectory) *A trajectory z sampled from an MDP is a sequence of state-action pairs $< s^k, a^k >$, $k = 0, \ldots, n_{hor}$, where $s^{k+1} \sim \mathcal{T}(s^k, a^k, .)$ and n_{hor} is the length (horizon) of the trajectory.*

Definition 5 (Policy) *A policy π is a mapping from $\mathcal{S}$ to $\mathcal{A}$. For an F-MMDP, the policy can be factorized as follows,*

$$\pi = \pi_1 \times \ldots \times \pi_{n_a},$$

where $\pi_i : \mathcal{S} \to \mathcal{A}_i$ is the local policy for agent i. We will denote space of all policies by Π.

Remark 3 *Note that the policy π_i depends on the joint space $\mathcal{S}$ and not only on the individual state space of the agent $\mathcal{S}_i$. The problems of latter form are in the realm of Decentralized MDPs, which is beyond the scope of this work.*

Definition 6 (Return) *Infinite horizon discounted return (or simply the return) $R^\pi(s)$: $\mathcal{S} \to \mathbf{R}$ of a policy π is,*

$$R^\pi(s) = \mathop{\mathbf{E}}_{s^{k+1} \sim \mathcal{T}(s^k, a^k, .)} \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}\left(s^k, a^k\right) \middle| a^k = \pi\left(s^k\right) \right] \tag{2.1}$$

where $s^0 = s$.

Definition 7 (Value Function) *Value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbf{R}$ associated with the policy π is,*

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \mathop{\mathbf{E}}_{s^{k+1} \sim \mathcal{T}(s^k, a^k, .)} [R^\pi(s^{k+1})]. \tag{2.2}$$

The optimal planning problem is defined as follows,

Definition 8 (Optimal Policy and Value Function) *For a given MDP, the optimal policy π^* is defined as follows,*

$$\pi^* = \operatorname{argmax} Q^{\pi},$$

the corresponding value function Q^{π^*} is called the optimal value function. A value function Q is called δ–optimal if

$$\|Q - Q^{\pi^*}\| \leq \delta$$

The optimal planning problem is computing the optimal value function Q^{π^*} and the corresponding optimal policy π^* for a given F-MMDP. This problem is well studied and there exists polynomial time algorithms (for instance Value Iteration and Policy Iteration [39]). However, the iteration complexity of these algorithms scale up exponentially with the increasing n_a, so it is intractable to solve the optimal planning problem for most cases. As such, this thesis focuses on the sub-optimal planning problem; that is given the constraints on computational resources, try to find the tightest δ that would yield an δ–optimal value function.

2.1.2 Approximate Planning with MDPs

As mentioned in the Section 2.1.1, exact planning is infeasible for most large-scale MDPs. The current research mainly focuses on approximate methods, where either the value function is represented approximately or the process to update the value of (s, a) pairs are approximated. A common approach is to sample trajectories from the MDP and estimate the value function only on the (s, a) pairs observed from these trajectories. This approach yields significant computational savings and often works well in practice [35].

This subsection presents a generic approximate MDP-solver, which is a variant of the Real Time Dynamic Programming algorithm [82]. Extension of the algorithm to F-MMDPs will be treated in the Chapter 3. This algorithm will be the main tool for solving F-MMDPs in this thesis, due to its convergence properties and low computational complexity. Before giving the definition of the algorithm, we first define the ϵ–greedy policy.

Definition 9 (ϵ–Greedy Policy) *An ϵ–greedy policy π^{ϵ} is a random function which returns $a = \pi(s)$ with probability ϵ and a uniformly random action from $\mathcal{A}$ with probability $1 - \epsilon$.*

The Trajectory Based Policy Iteration (TBPI) algorithm is presented in the Algorithm 1. TBPI algorithm samples trajectories from the MDP following an ϵ–greedy policy that is set

Algorithm 1: Trajectory Based Policy Iteration Algorithm

```
Input: MDP $\mathcal{M}$, Number of iterations $n_{traj}$, Initial state $s^0$,Trajectory length $n_{hor}$,
       exploration schedule $\epsilon(k)$, number of Monte-Carlo samples $n_{mc}$
Output: Policy $\pi$
// Optimistic Initialization
1  $\tilde{Q}^0 = R_{\max}$
2  $l \leftarrow 0$
   // Loop over trajectories
3  while $l \leq n_{traj}$ do
       // Loop over state-action pairs in each trajectory
4      while $k \leq n_{hor}$ do
5          Compute $a^k \leftarrow \pi^{\epsilon(k)}(s^k) = \arg\max Q^l$ w.p. $\epsilon(k)$
6          $\tilde{Q}^{l+1} = \tilde{Q}^l + \tilde{\mathbf{E}}_{n_{mc}}\left(\mathcal{R}(s^k, a^k) + \gamma\tilde{Q}^l(s'_i, \pi_i(s') - \tilde{Q}(s^k_i, a^k_i))\right)$.
7          Sample $s^k \sim \mathcal{T}(s^k, a^k, .)$.
8          $k \leftarrow k + 1$
9      $l \leftarrow l + 1$
10 return $\pi^l$
```

according to the current estimate of the value function $\tilde{Q}^l$, then the algorithm applies Monte-Carlo Bellman update to each s, a pair in the trajectory. The $\tilde{\mathbf{E}}_{n_{mc}}$ denotes the Monte-Carlo expectation with n_{mc} samples.

The TBPI algorithm is guaranteed to converge asymptotically to the optimal value function, given that every state-action pair is visited infinitely often, which can be ensured theoretically by picking an appropriate exploration schedule $\epsilon(k)$ [29]. Iteration complexity of the algorithm is $\Theta(|\mathcal{S}||\mathcal{A}|n_{traj}n_{mc})$ [83].

The main advantage of using TBPI is to avoid sweeping through all the state-action pairs. By applying a Monte-Carlo Bellman update to only the trajectories sampled from the model, we construct a policy that performs well on state transitions that are more likely to occur during the execution of the scenario. Although more advanced exploration algorithms, such as KWIK [84] can be used to handle the exploration-exploitation dilemma, we use the ϵ-greedy approach with optimistic initialization to handle this dilemma [28] to save memory and computation power.

2.2 Bayesian Optimization Algorithm

The Bayesian Optimization Algorithm (BOA) [81] is a probabilistic framework for solving black-box optimization problems by guiding the search for the global optima using probabilistic graphical models [85], built by performing inference on the samples obtained from the

Algorithm 2: Bayesian Optimization (BOA) algorithm

```
Input: The evaluation function $f$, Probabilistic Graphical Model $p(x;\theta)$, Initial
       estimate for the model parameter $\theta^0$, number of iterations $n_{iter}$, number of
       generated samples $n_{sample}$.
Output: Solution $x$
1 Generate uniformly random $x^0 \in \mathcal{X}$, where $|x^0| = n_{sample}$
2 Evaluate $y^0 \leftarrow f(x^0)$
3 for $i = 0$ to $n_{iter-1}$ do
4     Sample $x_i$ from the model $p(\theta^i)$, where $|x^i| = n_{sample}$
5     Evaluate $y^i \leftarrow f(x^i)$
6     Build Model $p(x, \theta^{i+1})$ using samples $(x^{i+1}, y^{i+1})$
```

objective function. Let $\mathcal{X}$ be the search space and let the objective function be $f : \mathcal{X} \to \mathbb{R}$. The problem is to find the global maxima

$$\max_x f(x) \qquad \text{s.t. } x \in \mathcal{X} \tag{2.3}$$

The pseudocode for the generic BOA is given at the Algorithm 2. The BOA can be described by the following steps,

- *Initialization (Lines 1-2):* The algorithm is initialized with selecting uniform random points x^0 from the search space $\mathcal{X}$ and computing the corresponding values y^0. Note that although uniform random selection is used in practice, any alternative method is valid as long as n_{sample} points are generated, such as sampling from the distribution $p(x;\theta^0)$.

- *Sampling and Evaluation (Lines 4-5):* At this step, the model $p(x;\theta^i)$ is sampled to generate n_{sample} number of new points. Then these samples are evaluated with the objective function to generate y^{i+1}.

- *Model Building (Line 6):* The new parametric probabilistic model $p(x;\theta^{i+1})$ is built using the samples x^{i+1} and their evaluated values y^{i+1}. Any probabilistic inference method is applicable at this step. One of the most common approaches is to truncate the set x^{i+1} based on their evaluations, such as keeping the first K-values with highest score and discarding the rest [81]. After the number of samples is truncated, inference methods such as maximum-likelihood estimation can be used to build $p(x;\theta^{i+1})$. If one defines a prior distribution $p(\theta)$ over the model parameters, Bayesian inference methods such as MAP estimation [26] can also be used. Alternatively, instead of discarding samples, we prefer a weighted-maximum-likelihood approach to involve all of the samples in the estimation process. Let,

$$l(\theta) = \prod_{j=1}^{n_{sample}} p(x_j^{i+1}; \theta)^{f(x_j^{i+1})/m} \tag{2.4}$$

represent the weighted likelihood, where m is a normalization constant. Setting $\theta^{i+1} = \arg\max l(\theta)$ ensures that the new distribution will have more probability mass on the points x with higher score.

The main objective of the algorithm is to converge to a probabilistic model that generates points x with higher values of $f(x)$ with high probability. This objective is achieved via the coupled processes of iteratively down-weighting points x with lower values from the sampled set. The BOA algorithm is shown to have better scalability properties in practice, compared to a lot of alternative black-box optimization methods (for a detailed analysis of BOA and comparisons to the other algorithms see [81]).

BOA belongs to a family of probabilistic search algorithms called Estimation of Distribution algorithms (EDA). Although any parametric distribution can be used as the model $p(x; \theta)$, for multivariate discrete variables, graphical models [85] are the most popular choice, because graphical models can embed the independences between random variables, which might result in computational savings during the inference step. In the Chapter 3, we will use a class of graphical models called factor graphs for representing the sampling distribution.

2.3 Approximate Model Learning

As mentioned in the Chapter 1, many existing planners including, the MDP solver described in the Subsection 2.1.2, need the transition model $\mathcal{T}$ to solve the planning problem. When the transition model is not available, it can be estimated from the sampled trajectories z (See Def. 4 by an estimation/learning algorithm. This Section provides the background on the exact and approximate transition model estimation.

2.3.1 Exact Learning

Let $\mathcal{T}_i : \mathcal{S} \to \mathbb{R}, i = 1, \ldots, n_a$ be a group of unknown functions that need to be estimated from samples, where $\mathcal{S}$ is a finite dimensional discrete state space and n_a is the number of agents. The multiagent learning problem consists of estimating the unknown functions $\mathcal{T}_i$ from the noisy samples $(s_{i,j}, t_{i,j})$, $j = 1, \ldots, n_d$, where n_d is the number of samples, $s_{i,j} \in \mathcal{S}$, $t_{i,j} = \mathcal{T}_i(s_j) + w$ and w is a random noise with bounded variance and zero mean. Each agent maintains and updates it's own estimate $\hat{\mathcal{T}}_i$. The learning process consists of two stages; the learning step,

$$\hat{\mathcal{T}}_i^k \leftarrow e(\hat{\mathcal{T}}_i^{k-1}, (s_{i,k}, t_{i,k})), \tag{2.5}$$

where k is the step number, e is the estimation law that updates the current estimate $\hat{\mathcal{T}}_i^k$ based on the past estimate $\hat{\mathcal{T}}_i^{k-1}$ and the observed noisy sample $(s_{i,k}, t_{i,k})$. The second stage is the communication step, where agents broadcast their estimated models/features (the concept of feature will be explained in the subsection 2.3.2) by using a communication function c,

$$c(\hat{\mathcal{T}}_i^k), \tag{2.6}$$

where the output of the c might be saturated due to the constraints on the throughput of the communication network [24]. We will use $N(i)$ to denote the set of agents that agent i can exchange information with. We will use f_{cap} to represent the upper limit on model parameters that can be exchanged at each communication step.

The objective of the multiagent learning problem is to design estimation law e and communication law c such that estimates $\hat{\mathcal{T}}_i^k \rightarrow \mathcal{T}_i$ as $k \rightarrow \infty$, preferably with an exponential rate.

2.3.2 Learning with Linear Function Approximation

In many practical scenarios, $\mathcal{S}$ is high dimensional, and since $|\mathcal{S}|$ goes up exponentially with the number of dimensions, it is computationally intractable to store $\hat{\mathcal{T}}_i$ exactly. Thus learning algorithms usually employ a lower dimensional approximation $\tilde{\mathcal{T}}_i \approx \mathcal{T}_i$. Linear function approximation is a popular choice due to its theoretical guarantees [28],

$$\tilde{\mathcal{T}}_i = \sum_{j=1}^{n_f} \theta_{i,j}\phi_{i,j}(s), \tag{2.7}$$

where $\theta_{i,j} \in \mathbb{R}$ are weights, $\phi_{i,j} : \mathcal{S} \rightarrow \mathbb{R}$ are basis functions (which are commonly referred to as *features*) and n_f is the number of features. Note that $\tilde{\mathcal{T}}_i$ is a linear approximation to $\mathcal{T}_i$ in the subspace spanned by ϕ_j. We denote $\theta_i = [\theta_{1,1}, \ldots, \theta_{1,n_f}]$ and $\phi_i = [\phi_1, \ldots, \phi_{1,n_f}]$ as the weight vector and feature vector correspondingly, so that we can write $\tilde{\mathcal{T}}_i = \theta_i^T \phi_i$ in vector notation.

The standard approach to update $\hat{\tilde{\mathcal{T}}}_i$ online from noisy samples $(s_{i,k}, t_{i,k})$ is to use the stochastic gradient approach [86],

$$\theta_i^{k+1} = \theta_i^k + \alpha^k \Delta_i^k(s_{i,k}, t_{i,k})\phi_i(s_{i,k}), \tag{2.8}$$

where $\alpha^k \in [0,1]$ is the learning rate used for mitigating with the noise at iteration k, and $\Delta_i^k(s_{i,k}, t_{i,k})$ is the estimation error at step k,

$$\Delta_i^k = t_{i,k} - \theta_i^{k^T} \phi_i^k(s_{i,k}). \tag{2.9}$$

It can be shown that under mild assumptions on α^k and the richness of the obtained samples [87], $\hat{\hat{\mathcal{T}}}_i \to \Pi_{\phi_i}\mathcal{T}_i$ as $k \to \infty$, where Π_{ϕ_i} is the projector operator that projects $\mathcal{T}_i$ to the subspace spanned by ϕ_i.

2.3.3 Incremental Feature Dependency Discovery (iFDD)

Stochastic Gradient Descent algorithms usually employ a fixed set of features. However, the quality of the approximation depends heavily on the choice of these features, which are usually not known beforehand. Incremental Feature Dependency Discovery (iFDD) is an online algorithm that expands the set of features ϕ_i based on the estimation error Δ^k at each step. The algorithm was initially developed to estimate value functions in the context of reinforcement learning [35] and was later adapted to estimate transition models [87].

iFDD works as follows: the algorithm is initialized with a set of binary features (that is $\phi_{i,j} : \mathcal{S} \to \{0,1\}$). The basic idea is to expand the feature set by taking conjunctions of the existing features where the estimation error persists. Let ϕ_i^+ be the set of features that are formed by taking conjunctions of the existing features at the state $s_{i,k}$,

$$\phi_i^+ = \{\phi_{i,j} \wedge \phi_{i,l} | j \neq l, \phi_{i,j}, \phi_{i,l} \in \phi_i\}. \tag{2.10}$$

This set is referred to as the list of potential features. The estimation error associated with the potential feature $f \in \phi_i^+$ is given as

$$\Delta^k(f) = t_{i,k} - [\theta_{i,j}^k, \theta_{i,l}^k]^T [\phi_{i,j}(s_{i,k}), \phi_{i,l}(s_{i,k})], \tag{2.11}$$

where $f = \phi_{i,j} \wedge \phi_{i,l}$. The relevance of each potential feature $\eta^k(f), f \in \phi_i^+$ is their absolute accumulated estimation error,

$$\eta^k(f) = \sum_{m=0}^{k} |\Delta^m(f)|. \tag{2.12}$$

iFDD adds f to the set of existing features whenever its relevance (Eq. 2.12)passes a predetermined threshold $\zeta > 0$. The weight of the new feature f is set as $\theta_{i,f} = \theta_{i,j} + \theta_{i,l}$.

2.4 Matrix Completion

Collaborative Filtering (CF) studies the problem of estimating multiple models by fusing information across different input sets. One of the biggest applications of CF is the web recommendation systems used by the companies such as Amazon and Netflix [88]. These systems frequently estimate which items are likely to be bought by which users based on the analysis of similarity of item ratings provided by the users. Matrix completion is one of the popular approaches in CF due to the existence of computationally efficient algorithms. Define the total number of users in the database as n_u and the total number of items as n_p. Let $M_{i,j} \in [0,1]$ denote the rating of the product i provided by user j. Usually $|M_{i,j}| \ll n_p \times n_d$ and the missing entries are estimated by solving the following optimization problem [89],

$$\begin{aligned} &\min \|X\|_* \\ &\text{s.t, } X_{i,j} = M_{i,j}, \end{aligned} \tag{2.13}$$

where $\|\cdot\|_*$ is the nuclear norm. This optimization problem is convex and can be solved in real time for reasonably sized matrices. We will refer to the matrix with missing entries M as the rating matrix and the solution of the optimization problem X as the prediction matrix.

2.5 Summary

This Chapter provided the background on mathematical tools, which are going to provide a basis for the algorithms developed later in the thesis. The TBPI algorithm (Algorithm 1) will be used in conjunction with the BOA (Algorithm 2) in the novel multiagent planning algorithm developed in the Chapter 3. The iFDD algorithm (Subsection 2.3.3) and Matrix Completion (Section 2.4) will serve as the building blocks for the multiagent learning algorithms developed in the Chapter 4.

Chapter 3

Multiagent Planning with Randomized Decompositions

This Chapter provides the development of the novel multiagent planning algorithm, Randomized Coordination Discovery (RCD). As discussed in the Sections 2.1.2 and 1.4, many existing approximate MDP solvers fail to scale up with the number of agents in multiagent scenarios. The coordination structure decomposition based methods [53, 90] yield the best scalability but require the existence of a fixed coordination graph structure beforehand. The main premise of the RCD is the discovery of useful coordination structures through a randomized search process.

The Chapter is organized as follows, Section 3.1 provides a general overview of the RCD algorithm. Next, Section 3.2 provides the definition of the Coordination Factor Graphs and how to perform planning with them on F-MMDPs. The Section 3.3 describes the RCD algorithm. The Section 3.4 proves the asymptotic convergence of the RCD algorithm and draws some links to the theory of linear function approximations. Finally, the Section 3.5 compares the empirical performance of the RCD with other alternative approximation search algorithms on 3 different multiagent planing domains.

3.1 Overview of The Developed Approach

Fig. 3-1 provides an outline of the Randomized Coordination Discovery (RCD) algorithm developed in this thesis, which builds upon the framework of Coordination Graphs (CGs) [53, 90]. In a nutshell, the algorithm performs a randomized search over the space of coordination graphs in order to discover graphs that yield high return policies.

The standard decomposition based MMDP solvers assume a fixed coordination graph structure beforehand. This structure is usually obtained from domain knowledge or param-

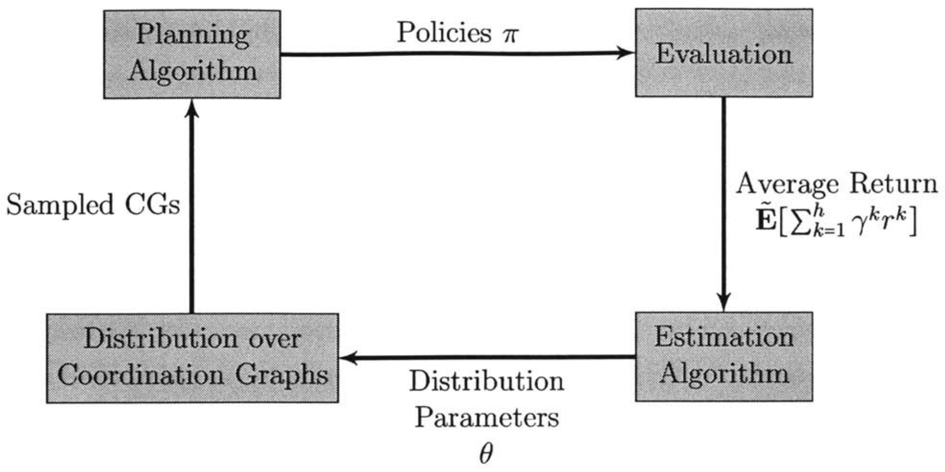

Planning Algorithm
Policies π
Evaluation
Average Return $\tilde{\mathbf{E}}[\sum_{k=1}^{h} \gamma^k r^k]$
Estimation Algorithm
Distribution Parameters θ
Distribution over Coordination Graphs
Sampled CGs

Figure 3-1: Diagram representation of the Randomized Coordination Discovery (RCD) algorithm. Each box in the diagram represents a different subroutine that works together with others to solve the MMDP problem.

eter tuning. The planning algorithm uses this pre-specified coordination graph to compute a multiagent policy. In contrast, RCD doesn't assume a fixed coordination graph structure, and keeps generating new coordination graphs based on the planning performance. RCD (Fig. 3-1) defines a parametric probability distribution over the space of Coordination Graphs. At the start of each iteration, a number of CGs are sampled from this distribution. Next, these sampled CGs are sent to the planning algorithm, which computes a policy corresponding to each CG. Then these policies are evaluated by computing the corresponding discounted returns, and the return-CG pairs are sent to the estimation algorithm. The estimation algorithm updates the sampling distribution by using the return-CG pairs. The updated distribution puts more probability weight to the CGs with higher returns. As a result, CGs with higher returns are sampled with higher probability in the next step.

The main contribution of this work is the randomized coordination graph search architecture displayed in the Fig. 3-1 and the use of Bayesian Optimization Algorithm (BOA) to efficiently search over the space of coordination graphs. By automating the Coordination Graph search with RCD, we obtain good suboptimal solutions to large-scale multiagent planning problems without the need for domain expert inputs. This is an improvement over the majority of the works mentioned in the Section 1.4, which assumed a fixed structure of Coordination Graphs. We also show in simulations that the developed algorithm is superior to the existing adaptive approximation techniques in terms of convergence speed and solution quality.

3.2 Multiagent Planning with Coordination Factor Graphs

We first discuss the Coordination Graphs (CFGs), which serves as the main building block of the RCD algorithm. Instead of formalizing coordination graphs as Markov Random Fields(MRF)s with each node acting as the local value function of the individual agents as in Parr's [53] and Kok's [90] work, we define CFGs as factor graphs where the state-action variables are shared across different factors. The number of factors are not necessarily equal to the number of agents, hence a more flexible decomposition can be obtained compared to the previous work on coordination graphs. The formal definitions are as follows,

Definition 10 (Factor Graph) *Let $X = \otimes_{i=1}^{n} X_i$ be a collection of variables. A factor graph is a tuple (F, V, E), where $F : X \to \mathbf{R}$, (V, E) is an bipartite graph with vertices decomposed as $V = X \cup f$, $f = \{f_1, \ldots, f_m\}$. A sum factor graph represents F from (V, E) as,$F = \sum_{i=1}^{m} f_i(\bar{x}_i)$, where as a product factor graph recovers F as, $F = \prod_{i=1}^{m} f_i(\bar{x}_i)$, where $\bar{x}_i = \{x_j \in X_j | e_{j,i} \in E\}$.*

Now we can define Factor Graphs in the context of F-MMDPs.

Definition 11 (Coordination Factor Graph) *Let $\mathcal{M}$ be an F-MMDP and let $\bar{Q} \approx Q$ represent an approximation of the value function of $\mathcal{M}$. A coordination Factor Graph (CFG) is a sum factor graph where $F = \bar{Q}$, $X = \mathcal{S} \times \mathcal{A}$, $f_i = \bar{Q}_i(\bar{s}_i, \bar{a}_i)$.*

Example 3.2.1 *An example CFG is displayed at Fig. 3-2. The global Q function is decomposed into a sum of 3 factors,*

$$Q = Q_1(\mathcal{S}_1, \mathcal{A}_1, \mathcal{S}_1^e) + Q_2(\mathcal{S}_2, \mathcal{A}_2, \mathcal{S}_3, \mathcal{A}_3) + Q_3(\mathcal{A}_3, \mathcal{S}_2^e) \tag{3.1}$$

There are two evident advantages of using an CFG to approximate the Q function. First, the number of parameters are greatly reduced. For instance if we set $n = |\mathcal{S}_i|$ and $m = |\mathcal{A}_i|$, the CFG in Fig. 3-2 needs only $n^2m + 2n^2m^2 + nm = \mathcal{O}(n^2m^2)$ parameters, whereas the exact Q function needs $\mathcal{O}(n^5m^3)$ parameters. The second advantage comes in when we want to compute $\max_{a \in \mathcal{A}} Q$, which is discussed next.

The following maximization problem occurs frequently in the design of MDP solving algorithms,

$$\max_{a \in \mathcal{A}} \bar{Q},$$

since the $\mathcal{A}$ can be high dimensional, this optimization problem is usually intractable for large-scale F-MMDPs. If the approximate value function $\bar{Q}$ is represented via an CFG,

$$\max_{a \in \mathcal{A}} \sum_{i=1}^{m} \bar{Q}_i(\bar{s}_i, \bar{a}_i). \tag{3.2}$$

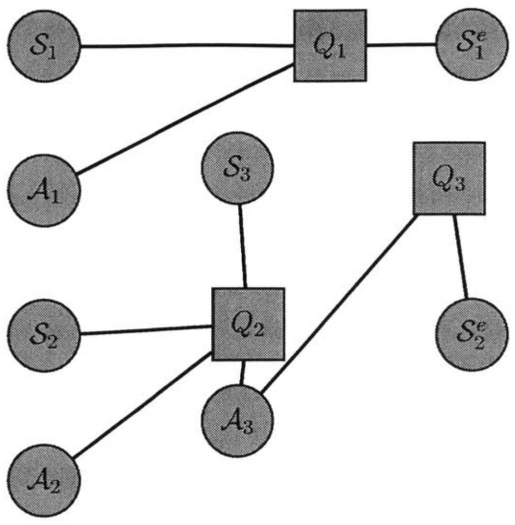


Figure 3-2: A CFG for a a 3-agent F-MMDP with 2 external variables and 3 factors.

Algorithm 3: Optimize Tree Factor Graph via Max-Sum

```
    Input: Tree Factor Graph $(F, V, E)$, vertices are labelled from leaves to the root
    Output: Maximizing variables $x_i^* \in X_i$
    // Initialize
1   forall the $x_i, f_j \in$ leaf nodes do
2       $\mu_{f_j \to x_i} = f_j(x)$
3       $\mu_{x_i \to f_j} = 1$
    // Pass messages from leaves to the root
4   forall the $x_i, f_j \notin$ leaf nodes do
5       $\mu_{f_j \to x_i} = \max_{x_1} \dots \max_{x_M} f_j(x, x_1, \dots, x_M) \sum_{k=1}^{M} \mu_{x_k \to f_j}(x_k)$
6       $\mu_{x_i \to f_j} = \sum_{f \in ne(x_i) \setminus f_j} \mu_{f_j \to x_i}(x_i)$
7   return $x_i^*$
```

The structure in CFG enables the use of techniques from the Machine Learning community to solve the optimization problem in Eq. 3.2 [91]. It can be shown that solving Eq. 3.2 is equivalent to performing inference on a probabilistic model using belief propagation algorithms. In particular, the algorithm is exact when the CFG has a tree structure [85]. Algorithm 3 presents the max-sum algorithm, which can be used to maximize a factor graph with a tree structure. Note that when we have a fixed state $s \in \mathcal{S}$ and if we pass $\sum_{i=1}^{m} Q_i(\bar{s}_i, \bar{a}_i)$ to the Algorithm 3, it will return the $a^* \in \mathcal{A}$, that is the solution to Eq. 3.2.

The TBPI algorithm (Algorithm 1) can be extended to the case where Q is represented by a CFG. The key step is to re-define the the Bellman update per factor as follows,

$$\tilde{Q}_i^+ \leftarrow \tilde{Q}_i + \mathop{\mathbf{E}}_{s' \sim \mathcal{T}(s,a,.)} \left(\frac{\mathcal{R}(s,a)}{n} + \gamma \tilde{Q}_i(s_i', \pi_i(s') - \tilde{Q}_i(s_i, a_i)) \right),$$

Algorithm 4: Coordinated Trajectory Based Value Iteration Algorithm

```
   Input: F-MMDP $\mathcal{M}$, Coordination Factor Graph $(Q, V, E)$, number of iterations
          $n_{traj}$, Initial state $s^0$,Trajectory length $n_{hor}$
   Output: Policy $\pi$
   // Optimistic Initialization
 1 $\tilde{Q}^0 = R_{\max}$
 2 $l \leftarrow 0$
   // Loop over trajectories
 3 while $l \leq n_{traj}$ do
       // Loop over state-action pairs in each trajectory
 4     while $k \leq n_{hor}$ do
 5         Compute $a^k = \pi(s^k) = \arg\max Q^l$ using Algorithm 3
 6         forall the $i \in [1, \dots, m]$ do
 7             $\tilde{Q}_i^{l+1} = \tilde{Q}_i^l + \tilde{\mathbf{E}}_{n_{mc}} \left( \frac{\mathcal{R}(s^k, a^k)}{n} + \gamma \tilde{Q}_i^l(s_i', \pi_i(s') - \tilde{Q}_i(s_i^k, a_i^k)) \right)$
 8         Sample $s^k \leftarrow\sim \mathcal{T}(s^k, a^k, .)$
 9         $k \leftarrow k + 1$
10     $l \leftarrow l + 1$
11 return $\pi^l$
```

where $\pi(s') = \text{argmax}\tilde{Q}(s,a)$. Combining Algorithm 3 with the Bellman update in Eq. 3, recovers the Coordinated Trajectory Based Policy Iteration (C-TBPI) algorithm, presented in Algorithm 4.

3.3 Randomized Coordination Discovery (RCD)

The Algorithm 4's performance relies heavily on the specified CFG. If the CFG is sparse and the approximation error is low, Algorithm 4 will return a policy with low optimality gap with little computation. However, specifying a good CFG beforehand usually requires domain expertise, which is not available in many practical situations. Hence it is reasonable to develop an algorithm that can efficiently search over possible CFG structures to meet the computation and performance demands. Randomized Coordination Discovery (RCD) Algorithm performs this task by combining BOA and C-TBPI. The main idea behind the RCD algorithm is to apply Algorithm 2 to automate the selection of CFG for the Algorithm 4.

We define the distribution over CFGs as a distribution over the edges $e_{i,j}$ on the graph. Since an edge $e_{i,j}$ takes values $\{0,1\}$, this renders each $e_{i,j}$ a Bernoulli random variable. Thus the sampling distribution is of the form,

$$P(e_{i,j};\theta), i = 1, \dots, n, j = 1, \dots, m, e_{i,j} \in \{0,1\}. \tag{3.3}$$

Example 3.3.1 *Depending on how we define the coupling between the variables $e_{i,j}$, we have a family of different probabilistic models to choose from. The Fig. 3-3 represents different forms of couplings between $e_{i,j}$ and the corresponding graphical models. Forcing each edge to be probabilistically independent from others (see Fig. 3-3b) results in a simple model, where the joint probability distribution is decomposed as,*

$$P(e_{i,j};\theta) = P(e_{1,1};\theta)P(e_{1,2};\theta)\dots P(e_{4,2};\theta).$$

This model has only 8 *parameters, hence updating the distribution is computationally very cheap. However due to negligence of dependency between the variables, this model does not capture the true distribution. On the other extreme, we have the fully connected model (See Fig. 3-3b), where there is no conditional independencies. This model captures all the possible dependencies between the edges, but it requires a lot of samples to update the parameters. This model has* 8×2^7 *parameters. We can also specify a model that captures some dependencies, as in Fig. 3-3d. This model is more expressive compared to the model in Fig. 3-3b whereas more sample efficient compared to the model in Fig. 3-3c.*

Since the model structure has a huge impact on the approximation error and sample complexity of the BOA algorithm [81], we also learn the structure of the model from the data. Structure learning for graphical models is a common practice in Machine Learning research and there are already well-established algorithms for learning the structure of Bernoulli graphical models, like the models displayed in Fig. 3-3.

We use the l_1 regularized logistic regression algorithm developed by Wainwright et al. [92] to learn the structure of the Markov Network $P(e_{i,j};\theta)$. This algorithm converts the structure learning problem to a convex optimization problem and the regularization parameter can be tuned to obtain the desired sparsity on the graph.

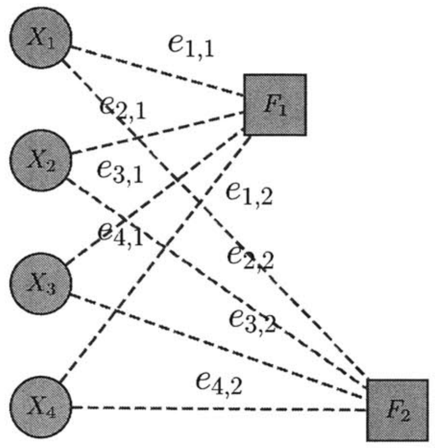
The algorithm assumes that the $P(x;\theta), x \in \{0,1\}$ has an underlying pairwise Markov network and a set of edges (V,E) with a joint Ising distribution,

$$p(x;\theta) = \exp\left(\sum_{s\in V}\theta_s x_s + \sum_{(s,t)\in E}\theta_{s,t}x_s x_t - \Psi(\theta)\right), \tag{3.4}$$

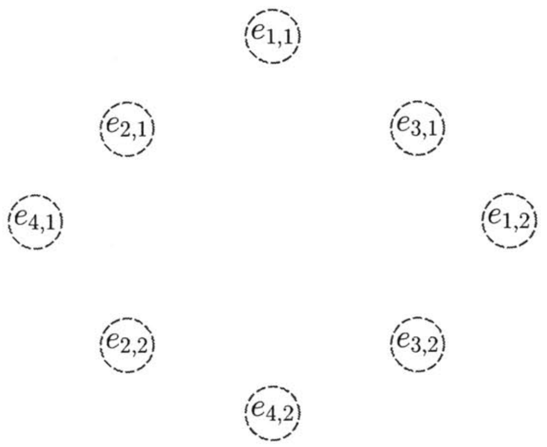
where $\Psi(\theta)$ is the log partition function. The optimization problem is cast as the minimization of the negative weighted log-likelihood. Let $x^i, i = 1,\dots,n_{sample}$ be the observed samples,

$$\hat{\theta}^{\lambda} = \underset{\theta\in\mathbf{R}^{|V|}}{\operatorname{argmin}} -\frac{1}{n_{sample}}\sum_{i=1}^{n_{sample}} R(x^i)\log[p(x^i;\theta)] + \lambda\|\theta\|_1, \tag{3.5}$$

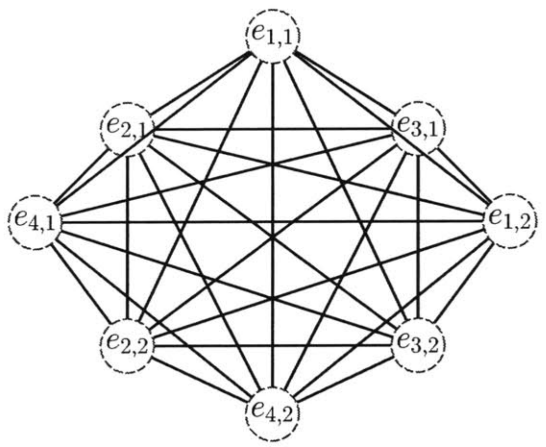
where λ is a regularization parameter and $R(x^i)$ is the discounted return of the policy of



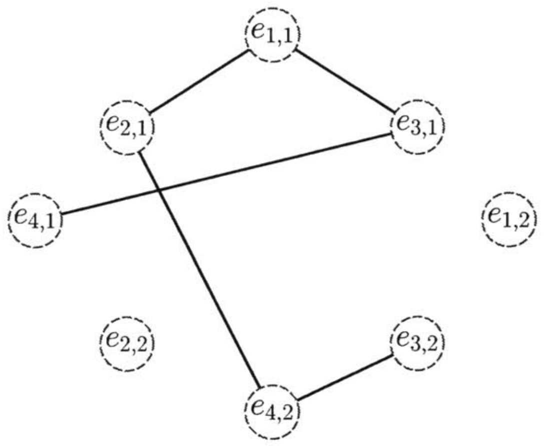
(a) A factor graph with 4 nodes and 2 factors. Edge $e_{i,j}$ represents the edge that connects the node X_i with the factor F_j.



(b) Edges on Fig. 3-3a represented as independent Bernoulli variables.



(c) Edges on Fig. 3-3a represented as joint Bernoulli variables via a fully connected graph.



(d) Edges on Fig. 3-3a represented as joint Bernoulli variables via a sparse graph.

Figure 3-3: Different representations of dependency between edges of factor graph.

C-TBPI Algorithm (Algorithm 4) computed using the CFG with the edge structure x^i.

Returning to Fig. 3-1, RCD algorithm can be summarized as the C-TBPI (Algorithm 4) as the planning algorithm, the distribution over the coordination graphs is given by Eq. 3.4 and the Estimation algorithm is BOA (Algorithm 2). We will refer the complete algorithm as RCD-TBPI from now on.

Table 3.1: Iteration complexity of RCD compared to other approaches.

Algorithm	Iteration Complexity
TBPI	$\Theta\left(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert n_{traj} n_{hor}\right)$
C-TBPI	$\Theta\left(\max_i \lvert\bar{\mathcal{S}}_i\rvert\lvert\bar{\mathcal{A}}_i\rvert n_{traj} n_{hor}\right)$
RCD-TBPI	$\Theta\left(\max_i \lvert\bar{\mathcal{S}}_i\rvert\lvert\bar{\mathcal{A}}_i\rvert n_{traj} n_{hor} (n_{agent} + n_{ext}) n_{factor}\right)$

3.4 Analysis

This Section investigates the iteration complexity and convergence properties of the RCD-TBPI algorithm.

3.4.1 Computational Analysis

The iteration complexity of the algorithms presented in this Chapter are given in the Table 1. The complexity difference between TBPI and C-TBPI highlights the usefulness of CFGs. For large-scale multiagent systems, $|S||A|$ is usually exponential in the number of agents and hence TBPI becomes computationally intractable. On the other hand, the complexity of C-TBPI depends only on the size of the largest factor $|\bar{S}_i||\bar{A}_i|$. Provided that the largest factor grows with polynomial rate in the number of agents, C-TBPI will also have polynomial complexity in the number of agents, as opposed to the exponential complexity of TBPI. In many practical scenarios, the maximal number of coordinating agents does not need to grow with the number of total agents, hence it is possible to obtain good planning performance with sparse coordination graphs with this property [53, 90].

However, as mentioned in the Section 3.3, the optimality gap of the policy returned by C-TBPI relies heavily pre-specified CFG. The Table 5.2 also shows that using RCD algorithm to automate the CFG search adds to the complexity, but it is only a linear factor in the number of agents. This low growth is due to fact that the l_1 logistic regression is a convex optimization problem that can be solved in linear time in the number of edges and because sampling from a sparse graphical model can be done with low computation. Hence the RCD-TBPI still has polynomial complexity in the number of agents, but relaxes the constraint of having to use a pre-specified CFG.

3.4.2 Convergence Analysis

This Section investigates the convergence properties of the RCD algorithm. First, the convergence of the stand-alone BOA algorithm with weighted maximum likelihood estimation is shown (Lemma 3.4.1). Then the convergence of the RCD algorithm follows (Theorem 3.4.2). We also show that there is a corresponding linear function approximation for every CFG (Lemma 3.4.3), which links our results to the well-known results in approximate dynamic programming with linear approximations.

The convergence of the classical BOA algorithm was established in [81] for a family of cost functions with a specific structure. Pelikan proved finite time convergence guarantees for these classes of problems. The following proof extends the work of Zhang and Muhlenbein [93], by considering the weighted likelihood estimation for discrete probability distributions.

Lemma 3.4.1 (Asymptotic Convergence of BOA) *Let $f : A \to [0,1], A \subset \mathbf{Z}^n$ be a positive multivariable discrete function. Define $A^* \subset A$ as the set of global optima, that is if $x \in A^*$ then $f(x) > f(y), y \in A \setminus A^*$ and $f(x) = f(y), y \in A^*$. Let BOA algorithm (Algorithm 2) be initialized with f and the sampling distribution $p(x;\theta^0)$. Let the model building step performed with weighted maximum likelihood approach (Eq. 2.4). Then as $n_{sample}, n_{iter} \to \infty$, $\theta^k \to \theta^*$ and $p(x;\theta^*)$ generates samples from the set A^* with probability one.*

Proof First we show that the weighted maximum likelihood estimation (Eq. 2.4) ensures that the probability of sampling a point $x \in A$ is proportional to its score. At the k^{th} iteration,

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmax}} \prod_{j=1}^{n_{sample}} p(x_j^{k+1};\theta)^{f(x_j^{i+1})}, x^{k+1} \sim p(x;\theta^k).$$

Hence, as $n_{sample} \to \infty$,

$$p(x;\theta^{k+1}) = \frac{p(x;\theta^k)f(x)}{\mathbf{E}_k[f(x)]}, \tag{3.6}$$

where $\mathbf{E}_k[f(x)]$ is the mean score, computed by summing the objective function over the probability mass at k^{th} iteration,

$$\mathbf{E}_k[f(x)] = \sum_{x\in A} p(x;\theta^k)f(x), \tag{3.7}$$

note that $\mathbf{E}_k[f(x)] \geq 0$ since f is a positive function. Hence we can show that,

$$\mathbf{E}_{k+1}[f(x)] = \sum_{x\in A} p(x;\theta^{k+1})f(x) = \sum_{x\in A} \frac{p(x;\theta^k)f(x)^2}{\mathbf{E}_k[f(x)]}. \tag{3.8}$$

Thus combining Eq. 3.7 and Eq. 3.8,

$$\begin{aligned}
\mathbf{E}_{k+1}[f(x)] - \mathbf{E}_k[f(x)] &= \sum_{x \in A} \frac{p(x;\theta^k) f(x)^2}{\mathbf{E}_k[f(x)]} - \sum_{x \in A} p(x;\theta^k) f(x), \\
&= \sum_{x \in A} \frac{p(x;\theta^k) f(x)^2}{\mathbf{E}_k[f(x)]} - \frac{\left(\sum_{x \in A} p(x;\theta^k) f(x)\right)^2}{\mathbf{E}_k[f(x)]}, \\
&= \frac{\sum_{x \in A} f(x)^2 p(x;\theta^k) - \mathbf{E}_k[f(x)]^2}{\mathbf{E}_k[f(x)]} \\
&= \frac{\mathbf{E}_k[(f(x)^2] - \mathbf{E}_k[f(x)]^2}{\mathbf{E}_k[f(x)]}.
\end{aligned}$$

By using the definition of the variance [94], we can see that nominator is $\mathbf{E}_k[(f(x)^2] - \mathbf{E}_k[f(x)]^2 = \mathbf{E}_k[(f(x) - \mathbf{E}_k[f(x)])^2)]$, which is the variance of $f(x)$ with respect to the probability distribution $p(x;\theta^k)$. Since the variance is always non-negative, it is seen that $\mathbf{E}_{k+1}[f(x)] - \mathbf{E}_k[f(x)] \geq 0$. Hence the sequence $\mathbf{E}_k[f(x)]$ in the Eq.3.7 is monotonically increasing with respect to the step k.

Since f is bounded on A, by the monotone convergence theorem [95], $\mathbf{E}_k[f(x)]$ converges to a limit $\mathbf{E}_k[f(x)] \to g^*$. Next, we need to show that $g^* = f^* = f(x), x \in A^*$. We will proceed with using proof by contradiction. Assume that $g^* \leq f^*$. Then, since $\mathbf{E}_k[f(x)] \to g^*$ is a monotonically increasing sequence and $g^* \leq f^* = \sup_{x \in A} f(x)$,

$$\lim_{k \to \infty} \frac{f(x)}{\mathbf{E}_k[f(x)]} = \frac{f^*}{g^*} \geq 1,$$

Also,

$$p(x;\theta^k) = p(x;\theta^0) \frac{f(x)^k}{\mathbf{E}_{k-1}[f(x)]\mathbf{E}_{k-2}[f(x)] \ldots \mathbf{E}_0[f(x)]}.$$

Thus, as $k \to \infty$,

$$\lim_{k \to \infty} p(x;\theta^k) \geq p(x;\theta^0) \lim_{k \to \infty} \left[\underbrace{\frac{f(x)}{\mathbf{E}_k[f(x)]}}_{\geq 1} \right]^k \to \infty,$$

which is impossible, since $p(x;\theta^k)$ is a probability distribution and thus $\sup_{x \in A} p(x;\theta^k) \leq 1$ for all k. Hence, $f^* = g^*$, which ensures that the limit $p(x;\theta^*)$ generates solutions from the set of optimal points A^* with probability one in the limit. ■

The convergence of RCD-TBPI builds upon the convergence properties of BOA, TBPI and sample complexity of l^1 logistic regression.

Theorem 3.4.2 (Asymptotic Convergence of RCD) *Let $\mathcal{M}$ be an F-MMDP and let*

RCD algorithm (see Fig. 3-1) be set-up with the coordinated TBPI algorithm (Algorithm 4) as the planner and the l^1 logistic regression algorithm (Eq. 3.5) for updating the CFG structure. Let $n_{factor} > 1$ be the number of factors in the CFG and let n_{agent} and n_{ext} denote the number of agents and the external variables. Assume the following,

A1) The number of samples $n_{samples}$ per iteration are greater than

$$6d^6 \log d + 4d^5 \log(n_{factor}(n_{agent} + n_{ext})),$$

where d is the maximum neighborhood size in the graph.

A2) Exploration schedule $\epsilon(k)$ of TBPI satisfies the infinite exploration conditions [29].

Under the assumptions above, RCD algorithm converges to a distribution $p(x;\theta^*)$.

Proof The convergence of BOA was established in the Lemma 3.4.1. In order to extend this result to the convergence of RCD-TBPI, we need to prove two statements. First, we need to ensure that the l_1 logistic regression algorithm consistently estimates the new model $p(x;\theta^{k+1})$ from the sampled CFGs, that is, the Eq. 3.6 is satisfied. Second, we need to ensure that C-TBPI algorithm converges.

Wainwright proved the consistency of graphical model selection with l_1 logistic regression in [92], assuming that the number of samples are greater than, $d^6 \log d + 2d^5 \log p$, where p is the number of nodes in the graph. The number of nodes in the RCD are the number of edges in the CFG, which is upper bounded by $(n_{factor}(2n_{agent} + n_{ext})$. Thus as long as the Assumption A1 is satisfied, the solution to the optimization problem in Eq. 3.5 will satisfy Eq. 3.6.

As noted in the Section 2.1.2, the TBPI algorithm is guaranteed to converge under the assumption A2, so the returns $R(x^i)$ associated with the sampled CFG with edges x^i will be consistent. ∎

The result above asserts that RCD-TBPI will converge to a solution. However, this result does not give any upper bounds on the sub-optimality gap. It is possible to recover a bound by linking this result to the theory of linear function approximations.

Lemma 3.4.3 (CFG is a Linear function Approximation) *Let (F, V, E) be a coordination factor graph that is used for approximating the value function $\tilde{Q} \approx Q : \mathcal{S} \times \mathcal{A} \to \mathbf{R}$. Then there exists a set of binary basis functions $\phi_i : \mathcal{S} \times \mathcal{A} \to [0,1], i = 1, \dots, n_\phi$ and a set of weights $\theta_i \in \mathbf{R}, i = 1, \dots, n_\phi$ such that $\tilde{Q} = \sum_i^{n_\phi} \phi_i \theta_i$.*

Proof Let $|\bar{\mathcal{S}}_i| = n_i$ and $|\bar{\mathcal{A}}_i| = m_i$. Hence, $\bar{Q}_i(\bar{s}_i, \bar{a}_i)$ is a tabular function that can take $n_i \times m_i$ different values. Define the following projection functions,

$$\psi_i^s : \mathcal{S} \to \mathcal{S}_i, \psi_i^a : \mathcal{A} \to \mathcal{A}_i,$$

such that $\psi_i^s(s) = s_i$ and $\psi_i^a(a) = a_i$. For each factor $\bar{Q}_i(\bar{s}_i, \bar{a}_i)$, we can define the following basis functions,

$$\phi_{i,j,k}(s,a) = \begin{cases} 1, & \text{if } \psi_i^s(s) = j, \psi_i^s(a) = k \\ 0, & \text{else} \end{cases},$$

for $i = 1, \ldots, n_i$ and $j = 1, \ldots, m_i$. Hence,

$$\bar{Q}_i = \sum_{j,k} \theta_{i,j,k} \phi_{i,j,k}(s,a).$$

Thus we can write the global value function approximation $\bar{Q}$ as,

$$\bar{Q} = \sum_i \bar{Q}_i = \sum_{i,j,k} \theta_{i,j,k} \phi_{i,j,k}(s,a).$$

Since projections ψ_i^s, ψ_i^a are linear functions, so are the basis functions $\phi_{i,j,k}$. Hence it is shown that $\bar{Q} \approx Q$ can be represented as a linear function approximation.

Linear function approximations are well studied in the Dynamic Programming/Reinforcement Learning literature (e.g., see [28, 29, 39]). By representing the CFG as a linear function approximation via Lemma 3.4.3, it is possible to use the results of existing theorems built upon the framework of linear function approximations.

Remark Assume that the distribution that BOA converges has a probability mass of 1.0 on a single CFG. Then, combining Theorem 3.4.2 with the Lemma 3.4.3 shows that the representation used by RCD-TBPI will converge to a fixed linear value function approximation $\bar{Q} = \sum_i \bar{\theta}_i \bar{\phi}_i$ asymptotically. The well-known result from [96] puts an upper bound on the approximation error introduced by a linear function approximation, when the basis functions ϕ_i are fixed. Since the representation used by RCD-TBPI converges to a fixed representation asymptotically, we can use the result from [96], to show the following upper-bound on the approximation error introduced by RCD-TPBI in the limit,

$$\|\sum_i \bar{\theta}_i \bar{\phi}_i(s,a) - Q^*(s,a)\| \leq \frac{1}{1-\gamma} \|\Pi_{\bar{\phi}} Q^*(s,a) - Q^*(s,a)\|,$$

where Q^* is the optimal value function, $\Pi_{\bar{\phi}}$ is the projection operator that projects the function to the space spanned by $\bar{\phi}$, and the norm $||.||$ is the norm weighted with respect to the stationary distribution of the MDP.

3.5 Simulation Results

This Section provides the simulation results for the RCD-TBPI algorithm across three different multiagent planning problems and compares its performance against three alternative planning approaches.

3.5.1 Domains

The selected test domains are as follows.

- **Distributed Sensor Network [90]:** In this problem a group of static agents coordinate to detect the movements of two targets that have stochastic dynamics. The simulations include 40 agents. Each agent is responsible for a set of overlapping cells on a grid. The state space consists of locations of two targets x, y positions on the grid - the agents themselves do not have any states. The action space for each agent consists of the binary decision variable "attempt to detect" or "do nothing", agents can also select which cell they are attempting to detect. The targets start at a fixed location on the grid and at each steps move to one of the neighboring cells with uniform probability. Every time an agent chooses the "attempt to detect" action and a target is located on that cell, the team gets a positive reward, with probability 0.25. This probability goes up by a factor of +0.25 if another agent responsible for that cell also chooses the action "detect". If the agent chooses the "attempt to detect" action and the target is not present in that cell, team gets a negative reward. The size of the planning space for this problem is approximately 10^{12} state-action pairs.

- **Urban Traffic Control [16]:** In this problem each agent controls a traffic light at an intersection, and the objective is to maximize the traffic flow while preventing conflicts. We considered a scenario with 40 traffic lights (agents) and 100 vehicles. Each vehicle enters and leaves the network randomly and follows a semi-random driving policy while in the network. The driving policy involves stopping at red lights, where the red and green lights are controlled by the agents. Agents get an award based on the time each vehicle spends in the network. Hence the traffic jams leads to diminished rewards. The size of the planning space for this problem is approximately 10^{22} state-action pairs.

- **Forest Fire Management [97]:** This mission involves a group of UAVs managing a forest fire. The stochastic forest fire spread model is from Boychuk [98]. The fire spread dynamics are affected by a number of factors, such as the wind direction, fuel left in the location and the vegetation. A total of 16 UAVs are present in the mission. Actions are traveling within a fixed distance in the forest or dropping water to kill the fire at the current location. The team receives a negative reward every time the fire spreads into a new location. The size of the planning space for this problem is approximately 10^{42} state-action pairs.

3.5.2 Compared Approaches

The following approaches were compared with RCD:

- **iFDD+ [99]:** iFDD+ applies linear function approximation directly to the value function. The method is adaptive in the sense that it starts with a fixed number of binary basis functions and grows the representation by adding new basis functions formed by taking conjunctions of the basis functions from the initial set. This algorithm was included in the comparisons because it is a the state-of-the-art adaptive function approximation technique in approximate DP/RL.

- **Fixed CG [100]:** In order to emphasize the value of automating the coordination graph search, an approach that involves a fixed Coordination Graph is also included in the results. We use intuition/domain knowledge to fix a CG beforehand, and use the same structure at every planning step.

- **Sparse Greedy Discovery: [90]:** This method is the most similar algorithm to the RCD, in the sense that it does not assume a fixed CG beforehand and adjusts the CG structure on the fly. The algorithm is initialized with the graph structure used in the Fixed CG approach. Then the algorithm applies χ^2-independence tests to greedily add/remove new edges to the coordination graph based on the rewards received at each step. Although this approach is also computationally cheap, it is not guaranteed to converge, and greedily adding/removing edges does not always improve the overall performance.

3.5.3 Results

For all domains, the algorithms were evaluated 30 times, and the average cumulative cost (negative reward) were computed per iteration. Each algorithm was allocated the same CPU

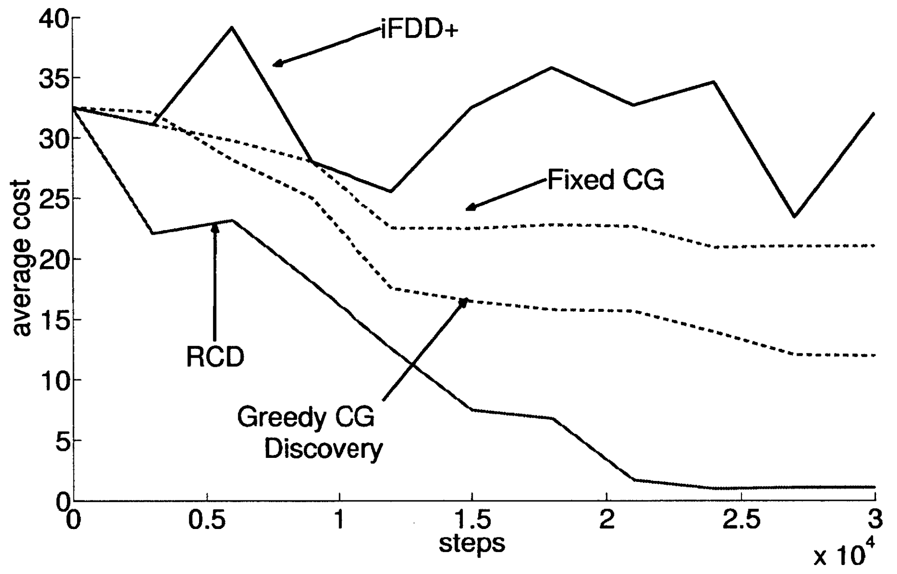


Figure 3-4: Comparison of performance of four planning algorithms on the distributed sensor network problem with 40 sensors.

time per iteration to ensure a fair comparison. Results are given in Figs. 3-4, 3-5 and 3-6. It can be seen that, across all domains the RCD algorithm outperformed the competing algorithms by a fair margin.

Specifically, in the sensor network domain (See Fig. 3-4), iFFD+ did not converge at all, however the other approaches yielded results with varying success. Fixed CG approach converged to a policy that yields consistent target detections, while the Sparse Greedy Discovery approach improved upon the fixed CG and eventually converged to a policy with lesser cost. RCD algorithm outperformed the Sparse Greedy Approach by converging to a policy with a much lower cost in approximately same number of iterations.

In the traffic regulation domain (See Fig. 3-5), the Sparse Greedy Discovery did not converge, and was discarded from the plot for a clearer presentation. iFDD+ performed much better in this domain compared to the sensor network and outperformed the fixed CG approach. This is most likely due to fact that it is more difficult to intuitively find a useful CG in complex domains. RCD converged to a structure/policy that has significantly lower cumulative cost.

In the forest fire management domain (See Fig. 3-6, Sparse Greedy Discovery again did not converge, while iFDD+ resulted in a poorly performing policy. Although the fixed CG approach yielded in a passable performance, RCD once again outperformed the competing approaches, in terms of both the convergence rate and the cost.

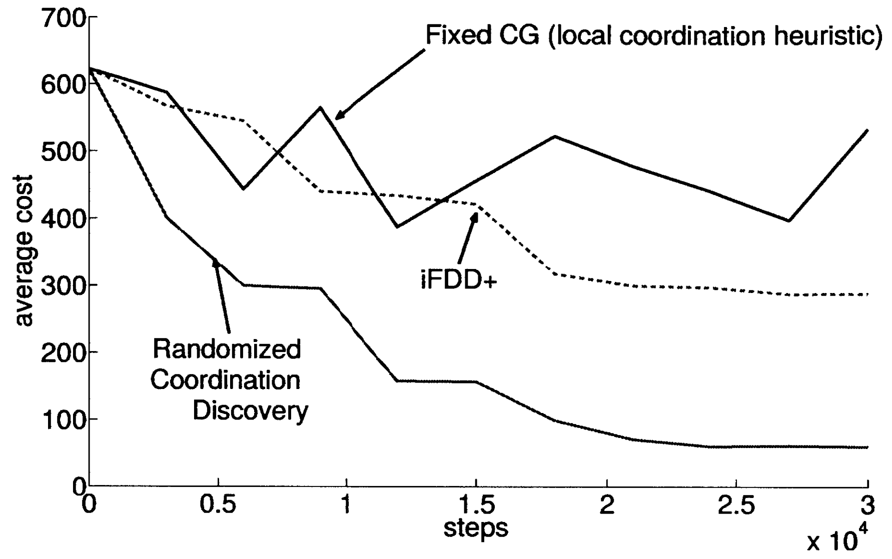


Figure 3-5: Comparison of performance of three planning algorithms on the traffic regulation domain with 40 agents.

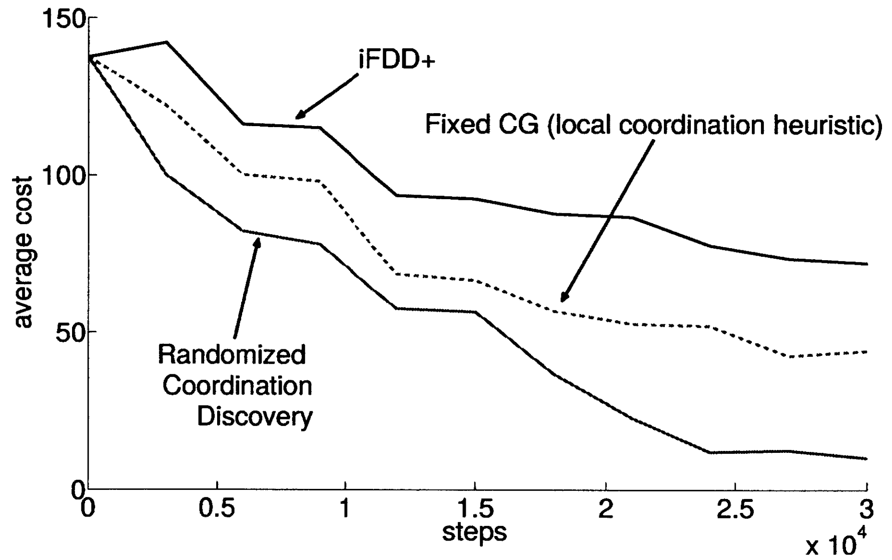


Figure 3-6: Comparison of performance of three planning algorithms on the fire fighting domain with 40×40 grid size and 16 agents.

Note that across all domains, RCD converged to a better solution compared to the fixed CG approach. This shows that for the considered problems, it was not easy to handcode a fixed CG that yields good performance. Hence, even for the case where domain expertise is available, a more efficient solution can be obtained through utilization of the RCD algorithm.

Finally, we would like to note that although the RCD algorithm removes the constraint on fixing the CG structure, it is not completely parameter free. The designer still needs to tune the number of factors and the sparsity penalty λ for l_1 logistic regression. For the complex multiagent planning problems considered in this Section, the effort required to tune these parameters were much less compared to tuning the CG structure directly.

3.6 Summary

This Chapter presented the development of the novel multiagent planning algorithm, RCD. A detailed discussion of the algorithm along with the convergence guarantees were provided. The simulation results reinforced the idea that automating the coordination structure search is useful and results in policies with significantly higher returns than the alternative approaches. Note that the RCD assumes the full knowledge of the transition model of the F-MMDP. In the next Chapter we are going to develop algorithms that can estimate transition models online by processing observations.

Chapter 4

Multiagent Learning with Approximate Representations

The previous Chapter developed algorithms for solving F-MMDPs with known transition and reward models. However, as discussed in Chapter 1, in many scenarios these models are not available. This thesis assumes that the reward model is available or can be estimated via another algorithm (such as using inverse reinforcement learning [101]) and focuses on estimation/learning of transition models using multiple measurements/observations obtained by the agents. Since the exact representation of the transition model is intractable for large-scale missions, the developed methods focus on learning with approximate representations.

Layout of the Chapter is as follows. Section 4.1 provides the development, analysis and simulation results for single agent learning with iFDD, which provides a basis for the multiagent learning algorithms developed in the later sections. Section 4.2 discusses the development of the extension of iFDD to multiagent problems and provides the Dec-iFDD algorithm. Finally, the Section 4.3 provides the analysis and simulation results for the CF-Dec-iFDD algorithm, which enables closed loop learning for highly heterogeneous teams.

4.1 Single Agent Learning with iFDD

The developed multiagent learning algorithms in this thesis decomposes the global learning problem as a collection of single agent learning problems that can be executed concurrently. Hence, before starting to attack the multiagent learning problem, it is necessary to develop an efficient single agent learning algorithm.

This section utilizes the iFDD algorithm (Section 2.3.3) to learn the transition model of a single agent. The motivation for using iFDD comes from not being constrained by a fixed representation, since the iFDD adjusts the representation complexity based on the observed estimation error. First, the problem statement and the estimation law is discussed and next the asymptotic convergence guarantees are proven. The Section concludes with the comparative simulation results across three different domains.

4.1.1 State Based Uncertainty Representation

In many scenarios it is unnecessary to estimate the full transition model $\mathcal{T}$. Usually, uncertainty in the missions is centered around the probability of occurrence of a single event, which forms a subset of elements of the transition model $\mathcal{T}$. We will denote this probability of the unknown event as p, which usually depends on the state of the MDP $s \in \mathcal{S}$.

If the mapping p is constant over all states, it corresponds to the *MDP with an unknown parameter* framework, which has been extensively studied in [20]. In a more general setting, p does not necessarily map each state to the same probability value, resulting in the *MDP with state-correlated uncertainty* framework. Solving such MDPs is the central theme of this Section.

From a practical point of view, p usually denotes occurrence probability of some event E. Here an event E refers to a set of state transitions $\langle s, a, s' \rangle, s' \sim \mathcal{P}_s^a$, that satisfy a certain condition. For instance, in the context of the robot navigation problem, an event may be defined as all state transitions where GPS signal was not received. Since the probability of the event occurrence depends on the state (*i.e.*, robot's location), then the problem needs to be formulated as an MDP with a state correlated uncertainty. In this setting, $p(s)$ can be written as the following set:

$$p(s) = P(\langle s, a, s' \rangle \in E | s). \tag{4.1}$$

For brevity, we only consider a single event that is action independent. For example in the robot navigation problem, the event is the GPS signal failure and it is assumed to be independent of the immediate action taken by the robot. Consequently we have removed the dependence of p on E and a in Eq 4.1. The extension is straight forward.

4.1.2 Learning State Based Uncertainties

An approximate representation of the uncertainty can be formed using linear function approximation with binary features [29] as follows

$$p(s) \approx \hat{p}^k(s) = \phi^{\mathsf{T}}(s)\theta^k, \tag{4.2}$$

where $\hat{p}^k(s)$ is the approximate representation at k^{th} step and $\phi(s)$ is the vector of features [1]. Each component ϕ_j is a binary feature characterized by a mapping from state to a binary value; $\phi_j(s) : s \to \{0,1\}, j = 1, ..., m$, where m is the total number of features and $\theta^k \in \mathbb{R}^m$ is the weight vector at step k. A feature ϕ_j is called *active* at state s if $\phi_j(s) = 1$. Set of active features at state s is given by $\mathsf{A}(s) = \{j | \phi_j(s) = 1\}$. Hence, Eq. 4.2 can be written as:

$$\hat{p}^k(s) = \sum_{j \in \mathsf{A}(s)} \theta_j^k,$$

where θ_j^k denotes the j^{th} component of θ^k.

New estimates are formed by updating the weight vector at each step. For that purpose, define a Bernoulli random variable $\Psi(s)$ with parameter $p(s)$. That is, $\Psi(s)$ is equal to 1 with probability $p(s)$ and zero otherwise. Let z^k be the k^{th} experienced trajectory with length N_{exec}, obtained from interacting with the environment. Define $s^{k,l}$ as the state at the l^{th} time-step of the k^{th} trajectory where $l = 1, ..., N_{exec}$. Let $\theta^{k,l}$ denote the corresponding weight vector. Define $\zeta(s^{k,l})$ to be the value that random variable $\Psi(s^{k,l})$ takes. This value can be gathered from z^k based on the occurrence of the event that is defined in Eq. 4.1. The weight vector can be updated applying a gradient descend type update as follows

$$\begin{aligned} \theta^{k,l+1} &= \theta^{k,l} - \frac{1}{2}\alpha^{k,l}\frac{\partial}{\partial\theta^{k,l}}[p(s^{k,l}) - \hat{p}^{k,l}(s^{k,l})]^2 \\ &= \theta^{k,l} + \alpha^{k,l}[p(s^{k,l}) - \hat{p}^{k,l}(s^{k,l})]\phi(s^{k,l}), \end{aligned}$$

where $\alpha^{k,l}$ is a scalar step-size parameter. Since p is unavailable to the algorithm, it is replaced by its estimate $\zeta(s^{k,l})$. Define the sampled estimation error at state s as $\Delta p^{k,l}(s) = \zeta(s) - \hat{p}^k(s) = \zeta(s) - \phi(s)^T\theta^{k,l}$. Then, the final form of the parameter update law is

$$\theta^{k,l+1} = \theta^{k,l} + \alpha^{k,l}\Delta p^{k,l}(s^{k,l})\phi(s^{k,l}). \tag{4.3}$$

Eq. 4.3 is a variant of the well studied stochastic gradient descent (SGD) algorithm [102]. Since the structure of p is not known beforehand, quality of the resulting approximation found by SGD depends strongly on the selected set of features.

[1] Our methodology can be extended to state-action correlated uncertainties by introducing feature vectors $\phi(s, a)$.

4.1.3 Adaptive Function Approximation using iFDD

Adaptive function approximators also modify the set of features based on the observed data based on the following general update rule:

$$\begin{aligned} \hat{p}^{k,l}(s) &= \phi^{k,l}(s)^{\top}\theta^{k,l}, \\ \phi^{k+1,l+1}(s) &= h(z^k, \theta^{k,l}, \phi^{k,l}), \end{aligned} \tag{4.4}$$

where h is the representation expansion function that adds new features to the feature vector based on sampled trajectories, weight vector, and previous set of features. Based on the successful results in representing high dimensional value functions, we employ iFDD [35, 45] (also see Section 2.3.3) as the adaptive function approximator for our framework to represent the uncertainty. The basic idea of iFDD is to expand the representation by adding conjunctions of the initial features based on an error metric, thus reducing the error in parts of the state space where the feedback error persists. The general outline of the algorithm is as follows: given a set of initial binary features, when performing the update for state $s \in z^k$, a conjunction set $\phi^c(s) = \{\phi_j(s) \wedge \phi_k(s) | j, k \in \mathsf{A}(s)\}$ is formed. These features are referred as candidate features. If the sum of sampled estimation error $\Delta p(s)$ over active candidate features exceeds some pre-determined threshold ξ, these conjunctions are added to set of features. The evaluation function learner (ELF) algorithm [103], expands the representation akin to iFDD that we use, yet candidate features are selected based on a limited set of heuristically selected features.

4.1.4 Convergence Analysis

This Section investigates the asymptotic properties of iFDD combined with the SGD algorithm presented in the previous section (iFDD-SGD). For the analysis, we consider the estimation part, assuming that the iteration number k is fixed and that each state $s^{k,l}$ and its corresponding random variable $\Psi(s^{k,l})$ is sampled by following an exploratory policy that turns the underlying MDP into an ergodic Markov chain.[2] For brevity, superscript k will be dropped from notation since it is fixed. We provide a theoretical proof showing that iFDD-SGD asymptotically converges to a solution with probability one using existing results on stochastic gradient descent theory [102, 104]. Moreover, we show that if p can be captured through the representation class, iFDD-SGD converges to this point. Throughout the section we assume the standard diminishing step-size parameter for Eq. 4.3.

[2]Here we are restricting ourselves to the class of MDPs that can be turned into an ergodic Markov Chain.

Preliminaries

Define $\mathcal{V}$ as space of all functions of the form $v : \mathcal{S} \to \mathbb{R}$. Define tabular representation as a set of features of the form $\phi_j(s_i) = \delta_{ij}$, where δ_{ij} is the Kronecker delta function and $s_i \in \mathcal{S}, i = 1, 2, ..., |\mathcal{S}|$. Note that tabular representation forms a basis for this space, since any $v \in \mathcal{V}$ can be represented as $v = \sum_{j=1,2,...,|\mathcal{S}|} v(s_j)\phi_j$. It can be easily shown that ϕ_j are orthogonal to each other, hence the $\dim(\mathcal{V}) = |\mathcal{S}|$. Let $f = \{\phi_j \in \mathcal{V}, j = 1, \cdots, n\}$ be a set of n linearly independent features. Define $\Phi_f \in \mathbb{R}^{|\mathcal{S}| \times n}$ to be the feature matrix with elements $\Phi_{(i,j)} = \phi_j(s_i), i = 1, 2, ..., |\mathcal{S}|, j = 1, 2, ..., n$. It can be shown that $\mathbf{span}(\Phi_f)$ is a subspace of $\mathcal{V}$ [35], hence the orthogonal projection operator $\Pi^{\mathcal{F}} : \mathcal{V} \to \mathcal{F}$ is well defined, where $\mathcal{F}$ is the subspace $\mathbf{span}(\Phi_f)$. The weight matrix used for the projection operator is a diagonal matrix with the steady state distribution of states as its non-zero elements. More details about the matrix description of the projection operator can be found in [35]. Moreover, define $\Omega(f)$ as the set of all possible conjunctions of the elements of $\phi_j \in f$ and let $\Omega(\mathcal{F})$ be the corresponding subspace. Define $C(s)$ as the total number of non-zero features at state s. Finally define D_{ϕ_i} as the set of features constituting feature ϕ_i. For instance if $\phi_i = \phi_j \wedge \phi_k \wedge \phi_l$ then $D_{\phi_i} = \{\phi_j, \phi_k, \phi_l\}$. If ϕ_i belongs to set of initial features then, $D_{\phi_i} = \{\phi_i\}$.

Convergence of iFDD-SGD

Lemma 4.1.1 *Under the ergodicity and diminishing step-size assumptions, using SGD with fixed feature representation f, $\hat{p}_l$ defined in Eq. 4.4 converges to $\Pi^{\mathcal{F}}p$ as $l \to \infty$, with probability one.*

Proof Ergodicity assumption simply turns the problem into a least-squares parameter estimation problem for p, with infinite amount of data. With the diminishing step-size assumption, it is sufficient to invoke the results of Thm 5.3.1 in [102] showing that stochastic approximation algorithm SGD will converge to least-squares solution asymptotically. That is $\hat{p}_l = \Pi^{\mathcal{F}}p$ as $l \to \infty$. ∎

The following lemma states that, when a new feature, which is constructed by taking conjunctions of existing features is added to the feature set, it will only be activated at the parts of the state space with more than one active feature. This conclusion will simplify the development of the convergence theorem.

Lemma 4.1.2 *Let $g \in \Omega(f)$ be added to the set of existing features by iFDD. Then $\forall s \in \mathcal{S}$ where $C(s) \leq 1$ before adding g, then $\phi_g(s) = 0$.*

Proof If $C(s) = 0$, proof is trivial since no feature is active at state s including ϕ_g. Let $C(s) = 1$, and let ϕ_j be the corresponding active feature. if $D_{\phi_g} \subset D_{\phi_j}$, then $\phi_g(s) = 0$ due to sparse activation property of iFDD explained in subsection 4.1.3. Assume that $D_{\phi_g} \not\subset D_{\phi_j}$, then there exists a $\phi_i \in f$ such that $\phi_i \in D_{\phi_g}$ and $\phi_i \notin D_{\phi_j}$. Since only ϕ_j is active at s, $\phi_i(s) = 0$, which in turn implies that $\phi_g(s) = 0$. ■

Theorem 4.1.3 *Under the ergodicity and diminishing step-size assumptions, $\hat{p}^l$, using SGD (Eq. 4.3) with iFDD representation with an initial set of features f and discovery threshold $\xi > 0$, converges to $\Pi^{\Omega(f)}p$ as $l \to \infty$ with probability one.*

Proof Since the number of conjunctions are finite, there exists a point at time, after which the set of features is unchanged. Let us call this terminal fixed set of features $f^\dagger$ and the corresponding subspace $\mathcal{F}^\dagger$. We show that the claim holds for all possible $\mathcal{F}^\dagger$:

- $(\Pi^{\mathcal{F}^\dagger}p = p)$: This means the representation is rich enough to capture the exact p vector. Lemma 4.1.1 shows that in this case $\hat{p}^l$ converges to $\Pi^{\mathcal{F}}p$ as $l \to \infty$, with probability one.

- $(\Pi^{\mathcal{F}^\dagger}p \neq p)$: This means $p \notin \mathcal{F}^\dagger$. Define $\mathcal{S}^- \subseteq \mathcal{S}$ as the set of states for which $\Delta p(s) = p(s) - \hat{p}(s) \neq 0$. We argue that $\forall s \in \mathcal{S}^-, C(s) \leq 1$. Assume this is not true and let $e(s)$ denote the cumulative sampled estimation error at state s. Due to ergodicity of the underlying Markov chain, state s will be visited infinitely many times, hence after some time $e(s)$ will exceed any pre-defined threshold ξ, and since $C(s) > 1$ iFDD algorithm would expand $f^\dagger$ with the active features at state s. This contradicts that $f^\dagger$ is the terminal fixed representation.

  Now, it is sufficient to show that $\Pi^{\mathcal{F}^\dagger}p = \Pi^{\Omega(\mathcal{F})}p$. We prove this part by showing that the projection of the asymptotic residual (*i.e.*, $\delta p = p - \Pi^{\mathcal{F}^\dagger}p$), on any unexpanded feature vector (*i.e.*, $\phi_q \in \Omega(f) \setminus f^\dagger$) is null. To do so, it is sufficient to show that $\delta p^\top \phi_q = \sum_{s \in \mathcal{S}} \delta p(s)\phi_q(s) = 0$. Since $\forall s \notin \mathcal{S}^- \Rightarrow \delta p = 0$, we can write the summation as: $\sum_{s \in \mathcal{S}^-} \delta p(s)\phi_q(s)$. On the other hand, Lemma 4.1.2 showed that $\forall \phi_q \in \Omega(\mathcal{F}) \setminus \mathcal{F}^\dagger, \forall s \in \mathcal{S}^-$, if feature q is added to the representation, then $\phi_q(s) = 0$. Hence $\forall \phi_q \in \Omega(f) \setminus f^\dagger, \delta p^\top \phi_q = 0$. Therefore adding any of the remaining features to $f^\dagger$ does not help the representation to reduce the residual error further down. So as $l \to \infty, \hat{p} \to \Pi^{\mathcal{F}^\dagger}p = \Pi^{\Omega(\mathcal{F})}p$. ■

Theorem 4.1.3 proves that given an initial set of features f, iFDD-SGD asymptotically converges to the best possible approximation with probability one. The analysis also provides guidance on how to select the set of initial features. If f is chosen such that $\dim(\Omega(\mathcal{F})) \geq |\mathcal{S}|$, iFDD-SGD's approximation will be exact asymptotically with probability one. For instance, consider the following process for selecting an initial set of features for an MDP with finite state space. Let s be represented by a d dimensional vector, where s_i corresponds to the i^{th} component. Hence $s = (s_1, s_2, \cdots, s_d)$. Let $\{v_i^1, v_i^2, \cdots, v_i^{n_i}\}$ denote the distinct values that s_i can take. Then initial features can be selected selected as follows:

$$
\begin{aligned}
f &= \begin{bmatrix} \phi_{11} & \dots & \phi_{1n_1} & \phi_{21} & \dots & \phi_{2n_2} & \dots & \phi_{dn_d} \end{bmatrix}^{\mathsf{T}}, \\
\phi_{ij}(s) &= \begin{cases} 1 & s_i = v_i^j \\ 0 & \text{otherwise} \end{cases}, i = 1, \dots, d, j = 1, \dots, n_i,
\end{aligned}
\tag{4.5}
$$

amounting to a total of $m = \sum_{i=1}^{d} n_i$ features. Geramifard et al. [35] demonstrated that, for such an initialization, $\Omega(f)$ satisfies $\dim(\Omega(\mathcal{F})) \geq |\mathcal{S}|$.

4.1.5 Simulation Results

In order to assess the algorithm's applicability to planning problems, the performance of iFDD learning is tested within a model based planning framework. At the k^{th} iteration, the simulator model has an estimate $\hat{p}^k$ of the the parameter from the iFDD learning algorithm. The planner interacts with the simulator for N_{plan} steps in order to design a policy π^k. This policy is executed in the actual environment for N_{exec} steps, where it is expected that $N_{exec} \ll N_{plan}$ because collecting samples is much more expensive from the real world compared to the simulation. The resulting trajectory z^k, which is of length N_{exec}, is used by the parameter estimator to obtain the new estimate $\hat{p}^{k+1}$ with which the simulator model is updated. In the next iteration, the planner computes an improved policy π^{k+1}, and the loop continues. The TBPI algorithm (See Section 2.1.2) was used as the planner.

The first two domains are classical RL problems: 1) gridworld and 2) block building problems motivated by [105]. Both domain descriptions are extended to include state correlated uncertainty. The third domain is a PST mission with state-correlated uncertainty in fuel dynamics. Structure of this domain is more complex and is included to validate our approach on a large-scale stochastic UAV mission planning problem. In all domains, γ was set to 0.9 and the threshold for iFDD (ξ) was set to 1.

Domain Descriptions

Gridworld Navigation with Sensor Management This domain consists of a robot navigating in a 10×10 gridworld shown in Fig. 4-1. The task for the robot is to reach the goal ($\star$) from the starting position ($\bullet$). Possible actions are $\{\leftarrow, \rightarrow, \uparrow, \downarrow\}$. There is 20% chance

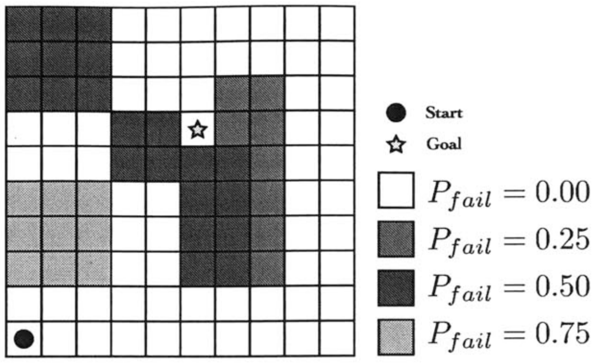


Figure 4-1: The gridworld navigation problem: the task is to reach ☆ from •. The robot has two sensors: a GPS and a camera. Using the GPS is preferred but unreliable depending on the location of the robot. p_{fail} represents the failure probability of the GPS.

that the robot moves to one of the adjacent grids that was not intended. Reward is +10 for reaching the goal and zero for all movement actions. In addition, the robot carries two sensors for navigation: a camera and a GPS. The GPS is the preferred tool for navigation with no additional cost, although the probability of receiving the GPS signal is location dependent shown in Figure 4-1 as p_{fail} highlighted with four colors. If the GPS signal is not received, the robot uses the camera, incurring an additional −1 reward, while receiving the exact position. The goal of the adaptive planner is to estimate the structure of the p_{fail} through interactions and modify the plan accordingly. The size of the state-action space of the domain is about 800.

Block Building In this domain, the objective is to distribute 5 blocks to 5 empty slots on a table shown in Figure 4-2. Initially all blocks are located under the table. The set of possible actions is defined by picking any of the blocks on or under the table and put it in any of the 5 slots. The episode is finished when there is no blocks under the table. The reward at the end of an episode is equal to the hight of the tallest tower minus one. However, placing a block on the top of the others involves p_{fall} probability of dropping the block under the table, which is increased by the size of the tower (*i.e.*, the longer the tower is, the harder is to place another block on top of it) and decreased by the number of blocks in adjacent slots. Let n_{slot} be the number of blocks in the destination slot, and n_{adj} be the maximum number of blocks in adjacent towers. Define $\bar{p}_{\text{fall}} = 0.1 \times (n_{slot} - n_{adj})^2 + 0.1$. Then p_{fall} is calculated as:

$$p_{\text{fall}} = \begin{cases} 0 & \bar{p}_{\text{fall}} \leqslant 0 \text{ or } n_{slot} = 0 \\ \bar{p}_{\text{fall}} & 0 < \bar{p}_{\text{fall}} < 1 \\ 1 & \bar{p}_{\text{fall}} \geqslant 1 \end{cases}.$$

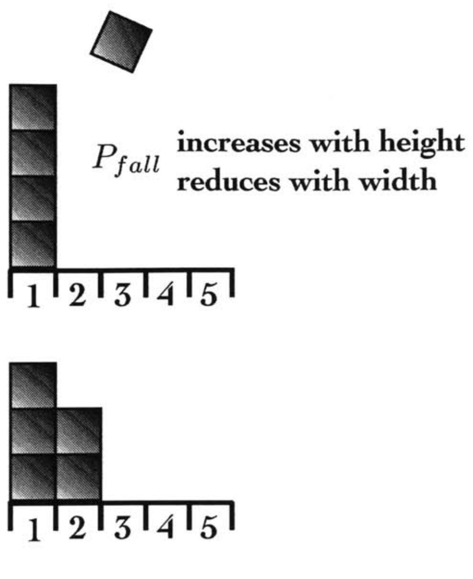


Figure 4-2: Block building domain: the task is to populate the table with blocks using the least number of slots. However as the number of blocks on a slot is increased, the probability of a block falling from the table increases accordingly. Given the discount factor of 0.9, the bottom figure shows the optimal solution.

The optimal solution is shown in the bottom of Figure 4-2. The state-action size for this domain is approximately 15×10^3.

Persistent Search and Track Mission The persistent search and track is a multi-agent UAV mission planning problem, where a group of UAVs perform surveillance on a group of targets, while maintaining communication and health constraints [106]. Outline of the mission is displayed in Figure 4-3. It should be emphasized that, although this is a multi-agent domain, decision making process is completely centralized and the state-action space consists of combination of all possible states-action pairs of each UAV. Each UAV's individual state is given by its location, fuel, and health. The health is defined by two bits indicating the functionality of the sensor and the actuator. There are three available actions for each UAV: {Advance, Retreat, Loiter}. The objective of the mission is to travel to the surveillance node and put surveillance on two targets, while one UAV stays at communication to provide the data link with the base. Each UAV starts with 10 units of fuel and burns one unit for all actions with probability p_{nom} and 2 units with probability $1 - p_{\text{nom}}$. Since UAVs are subject to more aggressive maneuvers in the surveillance area, p_{nom} should decrease in that region. Similarly when a UAVs health is degraded, maneuverability and fuel consumption degrade accordingly. Therefore p_{nom} is a state correlated uncertainty shown in the Table 4.1. If a UAV runs out of fuel, it crashes and can no longer continue the mission. UAVs are also subject to

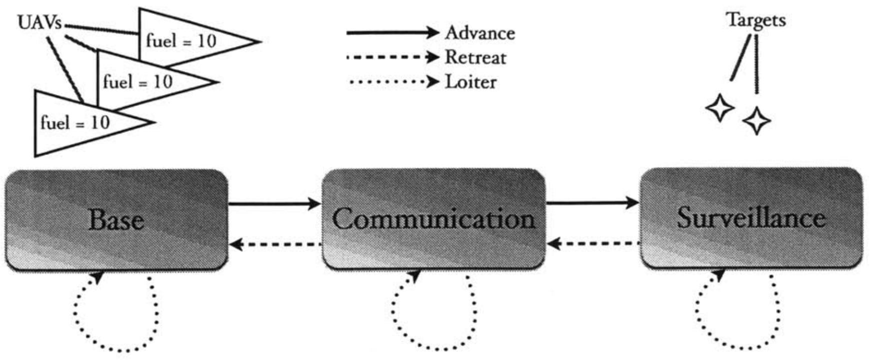


Figure 4-3: Persistent Search and Track Mission. The goal is to maintain surveillance on two targets on the right, while facilitate a data link with the base by having a UAV with a functional sensor loitering in the communication area.

Table 4.1: Probability of nominal fuel burn for UAVs across different locations and health status.

	Health Status		
Location	No Failure	Sensor Failed	Actuator Failed
Base	1.0	1.0	1.0
Communication	0.95	0.92	0.86
Surveillance	0.88	0.80	0.75

sensor and actuator failures at each transition step with probability $p_{\text{fail}} \in [0, 1]$. A UAV with failed sensor cannot perform surveillance whereas a UAV with failed actuator cannot perform neither surveillance nor communication. When a UAV returns to the base, it is refueled and its failures are repaired. Hence the objective of the planner is to maximize the number of time steps that two UAVs with working sensors are located in the surveillance region and having one UAV with a working actuator in the communication region. The complete MDP description of the mission can be found in [107]. This domain has approximately 19×10^6 state-action pairs.

Results

Figures 4-4, 4-5 and 4-6 plot the results. The X-axis is the number of iterations. Each iteration is a complete cycle of the learning-planning process. Each algorithm was executed 30 times and for each execution, after completion of each iteration the resulting policy was evaluated on the actual domain over 30 runs amounting to 900 samples per data point. The mean of the cumulated reward and standard deviation is plotted accordingly on the Y-axis.

Five different representation was used to compare various model estimation methods. In

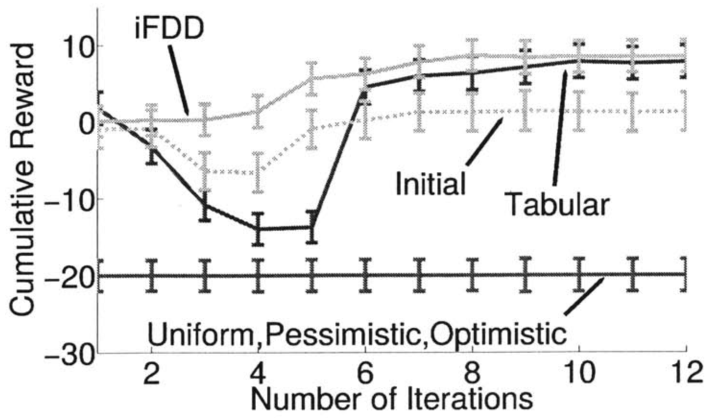


Figure 4-4: Comparison of five estimation methods combined with TBVI for the Gridworld domain. X-axis: number of iterations of the process shown; Y-axis represents the performance of the resulting policies.

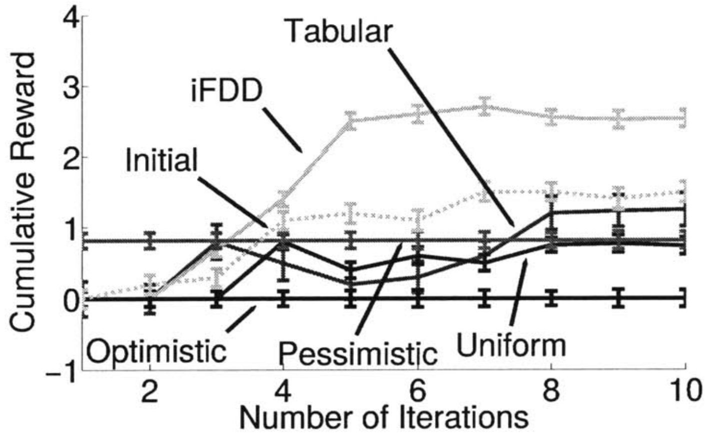


Figure 4-5: Comparison of five estimation methods combined with TBVI for the Block Building domain. X-axis: number of iterations of the process shown; Y-axis represents the performance of the resulting policies.

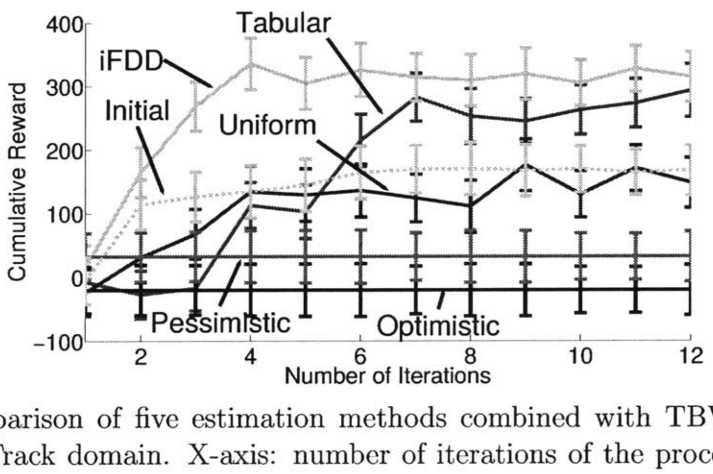


Figure 4-6: Comparison of five estimation methods combined with TBVI for the Persistent Search and Track domain. X-axis: number of iterations of the process shown; Y-axis represents the performance of the resulting policies.

order to emphasize the value of adaptation, fixed model estimators were also included in the evaluation. *Fixed Model Optimistic* and *Fixed Model Pessimistic* approaches assumed that the unknown parameter is fixed to 0 and 0.6 correspondingly across all states and did not update this value. *Uniform* representation ignored the state-correlation in the problem by utilizing a single fixed feature which was active at all times and was adapted based on observations. *Tabular* representation stored a distinct feature for each state, resulting in a number of of parameters equivalent to the size of the state space. Initial features for iFDD were selected using Eq. 4.5. To emphasize the value of discovering new features, results with the fixed initial features were also included in the plots.

Gridworld For adaptive planning experiment, planning and execution horizons were set to $N_{plan} = 8000$ and $N_{exec} = 100$. Performance of various estimation schemes using TBVI planner are displayed in Figure 4-4. In this case uniform estimation converged to the same policy obtained by any fixed model planners. This is due to fact that with a uniform uncertainty the optimal policy is to move directly towards the goal. This explains the poor performance of optimistic, pessimistic, and uniform estimators. Both tabular and iFDD estimators had the capability to capture the p_{fail} accurately. We conjecture that the dip on the performance of the tabular estimator is due to policy switching. Initially the agent followed the shortest route to the goal. As more samples were obtained, the agent explored the safe route from the right side, yet it required many samples to master the new route. The iFDD estimator performed substantially better early on compared to tabular estimator due to generalization of the features mentioned in Section 2.3.3. In this experiment iFDD started with 22 features and expanded on average a total of 150 features. iFDD representation also performed better than representation that uses only initial features, which emphasizes the importance of expanding the representation.

Block Building In this domain $N_{plan} = 6000$ and $N_{exec} = 30$. Trajectories were capped at 200 steps. Results are given in Figure 4-5. The optimistic model achieved 0 performance, because it assumed that $p_{\text{fall}} = 0$. Hence it kept trying to build a single tower which resulted in frequent collapses. The pessimistic approach placed 4 blocks into 4 slots and placed the final block on one of the placed blocks, achieving performance of 1. Uniform estimation eventually converged to the same policy as the pessimistic model. Both Tabular and iFDD estimators had the capability to capture the model exactly. While tabular estimator outperforms previous methods, iFDD estimator learned the task substantially faster and reached very close to the optimal policy shown in bottom part of Figure 4-2, using on average $\sim 10^3$ features.

Persistent Search and Track In this domain $N_{plan} = 10^5$ and $N_{exec} = 1000$. Results are shown in Figure 4-6. Both fixed model approaches perform poorly. The optimistic model assumed no uncertainty in the fuel dynamics, which resulted in an overly aggressive policy with frequent crashes. The pessimistic model assumed high uncertainty in the fuel dynamics, which resulted in a conservative policy that called UAVs back to the base early, resulting in a poor performance. Even though uniform estimation outperformed both fixed estimators, its performance did not improve after a number of trajectories due to its inability to capture the state-correleated uncertainty. Representation with initial features outperformed uniform, optimistic and pessimistic approaches, yet was not competitive. A similar trend to results presented before was observed between Tabular and iFDD estimators – the iFDD estimator settled around the best found accumulated reward among all methods much faster then the tabular estimator due to its capability to represent the uncertainty with fewer parameters. In particular, iFDD was ~ 2.5 times more sample efficient than tabular estimation according to Figure 4-6. In this experiment, iFDD expanded a total of $\sim 10^4$ features on average. The size of the parameter for the tabular representation was equivalent to the size of the state space which was larger by about 70 times.

4.2 Extension to Multiagent Learning: Decentralized iFDD (Dec-iFDD)

Although the iFDD model learning algorithm from the previous Section was shown to be an effective method for solving a variety of learning problems, the algorithms convergence rate slows down considerably for large-scale multiagent planning scenarios. One possible solution to this challenge is to decompose the learning to be per agent rather than operating in the joint state space of all agents, by using methods from distributed learning[31] and transfer learning[77]. Such methods accelerate the speed of the learning process by sharing model parameters. However these techniques have their own set of challenges, such as how to share learned model parameters under limited communication. In particular, in the *heterogeneous* learning setting[67], where the agents are learning different models, sharing model information may actually harm the overall system performance. Such heterogeneous team settings are common in UAV missions, where the dynamics or the operational space of each agent is different from each other.

This Section introduces the Dec-iFDD algorithm[108], which decomposes the problem to per agent to relax the computational complexity, and allows agents to share features with each other to improve their corresponding models.

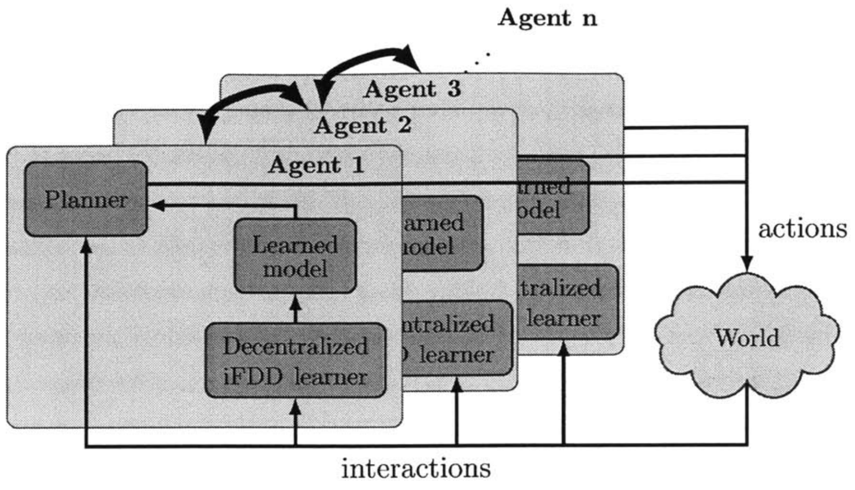


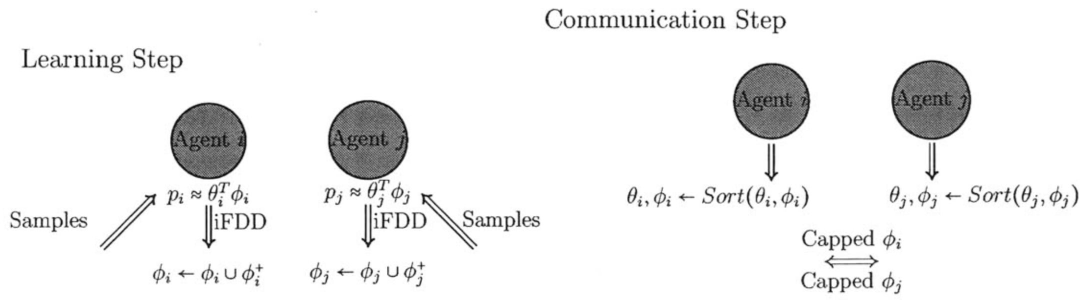
Figure 4-7: The integrated planning and learning architecture used in this Chapter employs decentralized collaborative online learning to learn the models of mission dynamics. These learned models are fed to a planning algorithm. Arrows in the figure indicate that the agents communicate with each other their representations of the uncertainty.

To address the problem of planning with learned models in large-scale heterogeneous teams, an integrated planning-learning approach was employed, displayed in Figure 4-7. The basic idea of the framework is enabling each agent to have its own decentralized learning algorithm (The Dec-iFDD algorithm) and communicate these learned models across each other and to a planning algorithm. The planning algorithm can be also decentralized depending on the mission formulation and requirement. Here the term decentralized is used to describe that each agent obtains and processes samples independently. Only during communication steps agents start interacting with each other to communicate model parameters. There is no single centralized processing unit for learning the models.

The Dec-iFDD algorithm sorts the features ϕ^i for each agent based on their corresponding weights θ^i and allows each agent to only share their most heavily weighted features. The maximum number of shared features can be tuned based on the communication constraints. This sharing methodology addresses the problem of model sharing with communication constraints and improves the convergence speed of the planning performance. The next Section presents the formal presentation of the algorithm.

4.2.1 The Algorithm

In many realistic scenarios the model of the uncertainty is not only state dependent, but is also agent dependent. Let n represent the number of agents and let $p^i(s)$ represent the


Learning Step
Agent i
Agent j
Samples
$p_i \approx \theta_i^T \phi_i$
iFDD
$\phi_i \leftarrow \phi_i \cup \phi_i^+$
$p_j \approx \theta_j^T \phi_j$
iFDD
Samples
$\phi_j \leftarrow \phi_j \cup \phi_j^+$
Communication Step
Agent i
Agent j
$\theta_i, \phi_i \leftarrow Sort(\theta_i, \phi_i)$
$\theta_j, \phi_j \leftarrow Sort(\theta_j, \phi_j)$
Capped ϕ_i
Capped ϕ_j

(a) Each agent maintains and updates an independent iFDD representation during individual learning steps

(b) Each agent ranks their features based on the corresponding weight and broadcast a capped number of top ranked features. Weight updates are handled individually for transferred features

Figure 4-8: The diagram representation of Dec-iFDD algorithm.

state dependent uncertainty functions that needs to be learned to model the uncertainty in the MDP of the i^{th} agent. A naive generalization of SGD-iFDD to the decentralized setting can be obtained by running a separate SGD-iFDD for each UAV. In this setting, each UAV updates its own representation based on the individual observations. This approach does not leverage the ability of the agents to collaborate. In particular, even though the environment may affect each agent in a different way the underlying features used to represent the environment should be common, as the agents all operate in the same environment. The main idea behind the Decentralized iFDD (Dec-iFDD) algorithm presented in this subsection is increasing the efficiency in learning by allowing agents to share the iFDD discovered features with each other under communication constraints. The pseudocode of the Dec-iFDD algorithm is provided in Algorithm 5.

The line by line description of the Algorithm 5 is as follows, the algorithm takes number of agents n, the set of binary features for each agent ϕ_{init} and the cap on shared features ϕ_{cap} as the input. At each step, agents interact with the environment to receive observations (line 10), and then each agent applies the standard iFDD algorithm independent of each other to update it's current set of features ϕ^i as well as the corresponding weights θ^i (line 11). When the current step is a sharing step, each agent sorts it's feature based on the weights (line 5), and then the first ϕ_{cap}/n number of features with the largest weights are broadcast in the network (line 6 and 7). Then all the agents update their feature set with the broadcasted features and then set the weights for their new features (line 8). A diagram representation of the algorithm is shown in Fig. 4-8.

There are two striking properties of the algorithm. First, note that the algorithm only allows agents to share features and not the weights each other. This is especially important for the heterogeneous teams, where the agents may share common features due to operating

Algorithm 5: Dec-iFDD Model Learning Algorithm

```
Input: Number of Agents $n$, Set Initial Binary Features $\phi_{init}$, Shared Feature Cap $\phi_{cap}$
Output: Estimated model $\hat{p}^i$ for each agent $i = 1, \ldots, n$
1  Initialize $\phi^i = \phi_{init}$ Initialize $\theta^i = 0$
2  foreach Step do
3      foreach Agent do
4          if Step = Sharing Step then
5              $\phi'^i \leftarrow$ Sort Features($\phi^i, \phi_{cap}, \theta^i$)
6              Broadcast($\phi'^i$)
7              $\phi^+ \leftarrow$ Listen()
8              $\phi^i \leftarrow \phi^i \cup \phi^+$, $\theta^i \leftarrow$ Update Theta($\phi^i$)
9          else
10             $s, a, s' \leftarrow$ Observe Transition
11             $\phi^i, \theta^i \leftarrow$ Expand Representation($s, a, s'$)
```

in the same environment, but each feature is weighted differently due to heterogeneity of the team. Secondly, the algorithm takes the communication limits into consideration with capping the total number of shared features in the network by the parameter ϕ_{cap}, and forces agents to only share the more heavily weighed features in their representations at each sharing step. The implications of these two properties are shown in the following simulations.

4.2.2 Simulation Results

For each multiagent MDP, we consider the learning of a single state-correlated uncertainty $p^i(s)$ per agent, where $i = 1, .., n$ indexes the agents. For all missions it is assumed that a nominal model $p^{\text{nom}}(s)$ and the underlying $p^i(s)$ for each agent was generated by randomly perturbing this nominal model. A team is called *homogeneous*, when the underlying $p^i(s)$ for each agent is within 5% limits of the nominal model. A team is called *heterogeneous*, when the underlying $p^i(s)$ for each agent is within 20% limits of the nominal model.

For both heterogeneous and homogeneous teams, we compare 4 different learning approaches. *The centralized learner with homogeneity assumption* assumes that $p^i(s)$ are the same for all agents, and therefore concatenates observations from all agents and processes these by the iFDD algorithm to generate a single model. Then all agents plan using this single model. Alternative to the centralized learner, the Dec-iFDD algorithm (Algorithm 5) with 3 different feature sharing caps (FSC), $\{0, 10, 100\}$, was compared. Note that FSC$= 0$ corresponds to the case where each agent learns completely independently and shares nothing with each other.

Two domains are inspected, the multi-agent block building problem, which is the multi-

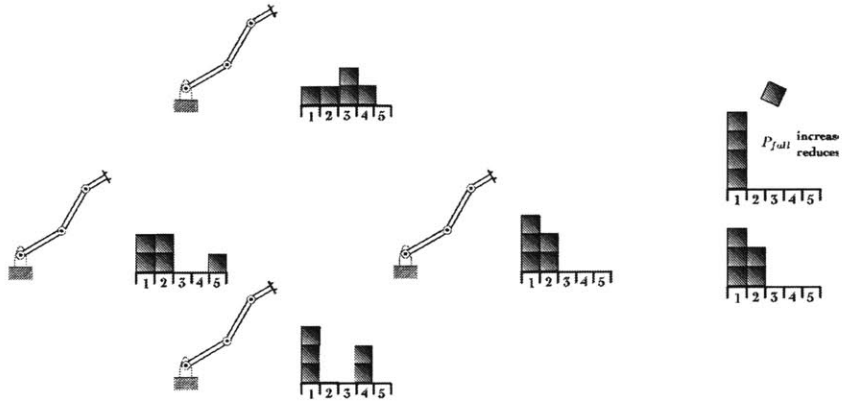


Figure 4-9: The multiagent blocksworld problem with 4 agents. The baseline probability model correlates the configuration and the probability of blocks falling

agent extension of a classical AI problem and the PST mission, which was previous used in the simulation results of the Section 4.1. The main objective of these simulation results is to show that, if the team is heterogeneous, the centralized learning can actually hurt the performance and it is outperformed by distributed learning approach.

Multiagent Block Building Problem

This problem formulation is motivated by classical blocksworld problem that appears as a benchmark in many different AI applications [105]. Problem consists of picking up the blocks on a table and arranging them into a specific configuration, usually a tower. The stochastic dynamics are introduced to the system though p_{fall}, which denotes the state-independent probability that a block may fall from the top of the tower back to the table. This formulation of the problem have been solved by many different planning algorithms in the past [45]. The authors have formulated an alternate version of the problem[87] with state-dependent p_{fall}. In this formulation, probability of the block falling down increases as the tower gets higher and decreases with the presence of adjacent blocks. The objective of the planning/learning problem is to learn the structure of $p_{\text{fall}}(s)$ and build the optimal block configuration.

This Chapter presents a decoupled multi-agent version of the blocksword problem with state dependent uncertainty. As shown in Fig. 4-9, the problem consists of coordinating $n = 4$ robotic arms for building the blocks. The dynamic of each block building problem is completely decoupled from each other, thus the planning can be performed independently. For this mission learning is executed at every 300 steps for each robot, and the MDP that corresponds to the learned models were solved by using the value iteration algorithm[39]. All results are averaged over 30 runs.

The nominal model is given as follows. Let n_{slot} be the number of blocks in the destination slot, and n_{adj} be the maximum number of blocks in adjacent towers. Define $\bar{p}_{\text{fall}} = 0.1 \times (n_{slot} - n_{adj})^2 + 0.1$. Then p_{fall} is calculated as:

$$p_{\text{fall}}^{\text{nom}} = \begin{cases} 0 & \bar{p}_{\text{fall}} \leqslant 0 \text{ or } n_{slot} = 0 \\ \bar{p}_{\text{fall}} & 0 < \bar{p}_{\text{fall}} < 1 \\ 1 & \bar{p}_{\text{fall}} \geqslant 1 \end{cases} .$$

The results for the homogeneous team is given in Fig. 4-10. Results show that the centralized algorithm required ≈ 30% less number of steps to converge compared to the Dec-iFDD algorithm with the largest FSC. It can be concluded that for this homogeneous scenario, using the same weights θ and features ϕ for all the agents resulted in a good approximation.

The results for the heterogeneous team are given in Fig. 4-11. The results show that centralized learning algorithm is unable to converge in this case because the algorithm generates a single model by processing all the observations, but the underlying samples are generated by substantially different models. The Dec-iFDD algorithm on the other hand, learns individual models for each agent and hence the agents can adapt their plans to their specific models. In addition, as the FSC increases, agents are able to share more relevant features with each other and the convergence rate of the overall performance increases significantly.

Persistent Search and Track with UAVs

For simulations, a large scale PST simulation with 10 collaborating UAVs was considered. It was assumed that the state-correlated actuator failure probability $p_a(s)^i$ and state-correlated nominal fuel burn rate for each agent i is unknown at the beginning of the mission, and must be learned through interactions with the environment. The baseline probability of actuator failure model that is used in the simulations is given at Table 4.2 and the baseline nominal fuel burning rate is given at Table 4.3. State-dependency of the fuel burning rate was motivated from the possibility of having non-uniform wind in the mission environment and thus each region induces a different fuel burning rate. Table 4.3 shows the probability of incurring nominal fuel burn in the respective states. Non-nominal fuel burn rate is set as twice the nominal fuel burn rate.

The Group Aggregate Dec-MMDP (GA-Dec-MMDP) algorithm [7] was used to replan after every model update. Similar to how the Dec-iFDD algorithm decomposes the learning

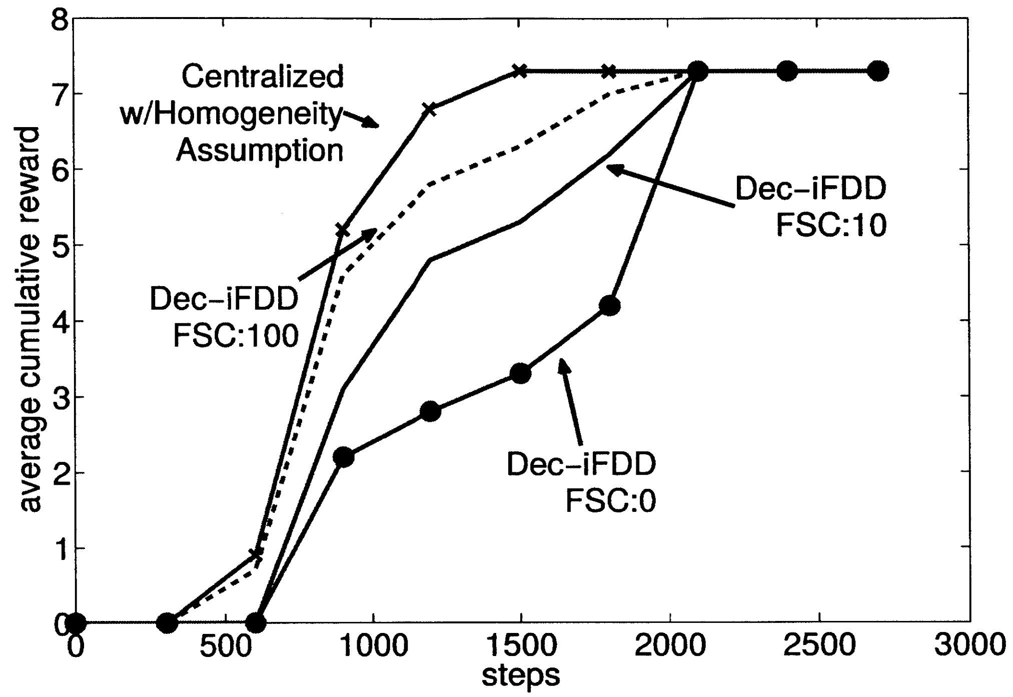


Figure 4-10: Comparison of centralized approach versus decentralized approaches with different feature sharing caps (FSC) for the multiagent blocksworld scenario where model perturbation is 5% from the nominal model

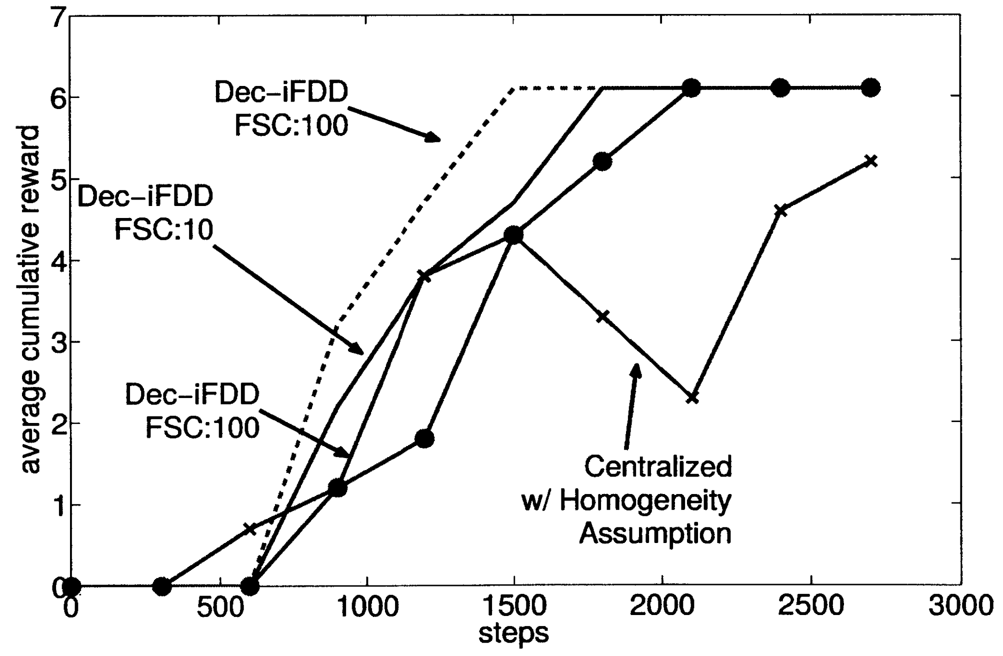


Figure 4-11: Comparison of centralized approach versus decentralized approaches with different feature sharing caps (FSC) for the multiagent blocksworld scenario where model perturbation is 20% from the nominal model

Table 4.2: Baseline probability of actuator failure across different locations and fuel levels.

	Fuel Level	
Location	High Fuel Level	Low Fuel Level
Base	0.0	0.0
Communication	0.05	0.1
Surveillance	0.2	0.3

Table 4.3: Baseline probability of nominal fuel rate across different locations and health states.

	Health		
Location	Healthy	Sensor Fail	Actuator Fail
Base	0.0	0.0	0.0
Communication	0.95	0.92	0.86
Surveillance	0.95	0.92	0.86
Surveillance (Windy)	0.90	0.80	0.75

problem to per agent, the Ga-Dec-MMDP planner decomposes the planning problem per agent to enable online planning in large-scale multiagent missions. The development, and additional simulation and flight results for the GA-Dec-MMDP algorithm are available[7, 108].

Figure 4-12 compares the cumulative cost performance of several instances of the Dec-iFDD algorithm with different feature sharing caps for almost homogeneous UAV team. In this case, it can be seen that as the cap on features to be shared is increased, the performance of the algorithm approaches that of the centralized algorithm. This result reinforces the intuition that the decentralized algorithm can do no better than a centralized one when the agent models of uncertainty are homogeneous. Figure 4-13 on the other hand, shows the performance of the Dec-iFDD algorithm for the same set of feature caps but in a network of collaborating UAVs with more heterogeneity (20% perturbation from a nominal model). In this case, it can be seen that planning using the output of the decentralized algorithm results in much less cumulative cost, because the planner is able to adjust the plan to suite the capability of each individual agent.

In order to demonstrate the proactive behavior of the resulting policy, two different mission metrics were averaged over 100 simulation runs and results are displayed on Table 4.4. It is seen that the proposed planning-learning approach leads to policies with less number of total failures in the expense of commanding vehicles to return to base more frequently. This behavior is due to ability of Dec-iFDD to learn a specific model for each agent, opposed to centralized planner.

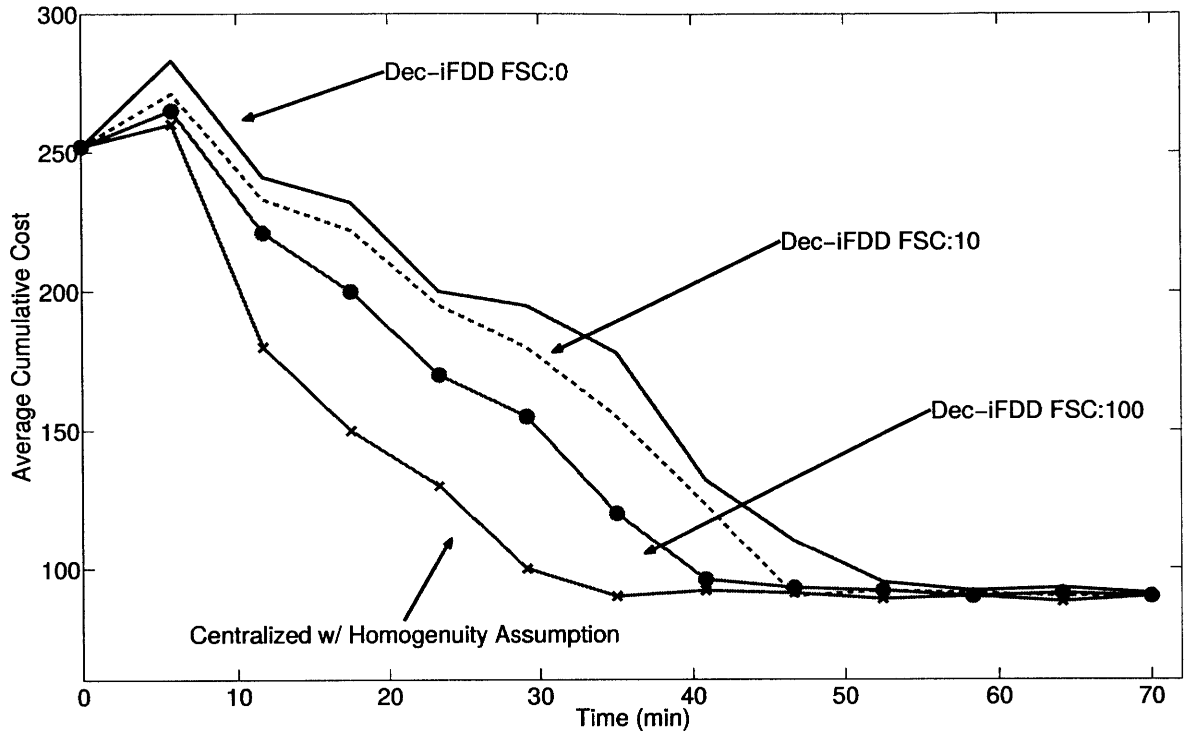


Figure 4-12: Comparison of centralized approach versus decentralized approaches with different feature sharing caps (FSC) for the PST scenario where the model perturbation is 5% from the nominal model. Results are averaged over 100 runs.

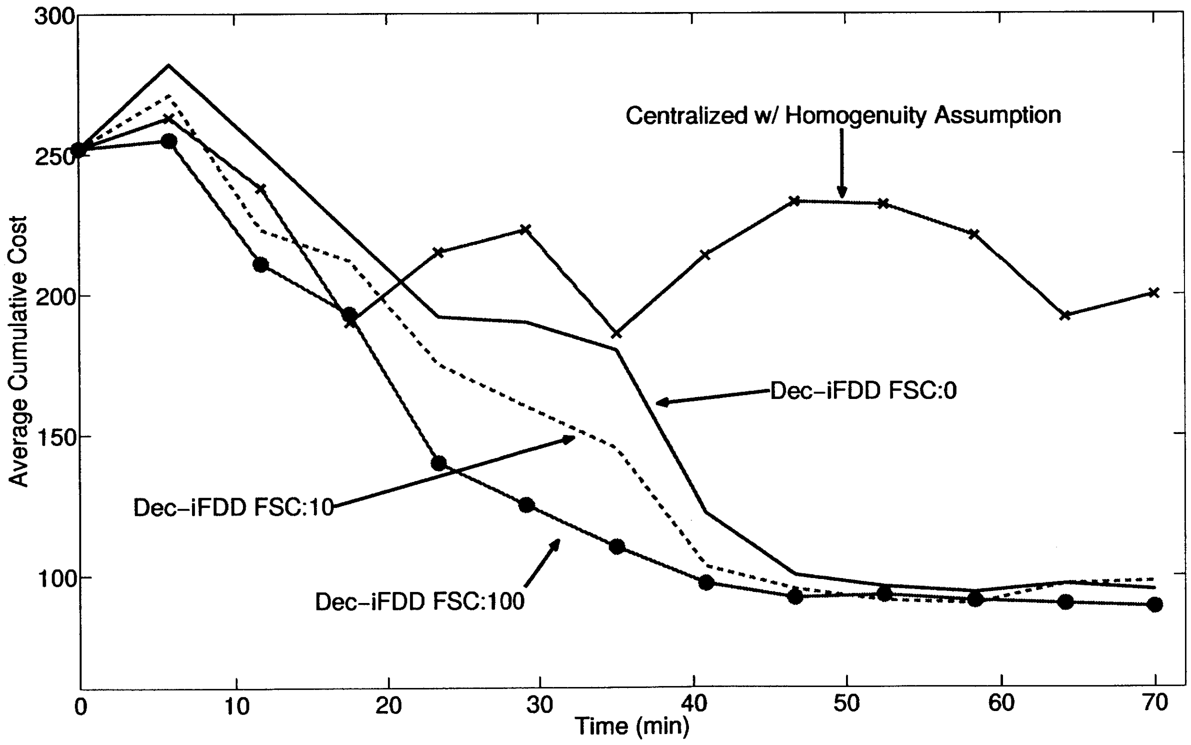


Figure 4-13: Comparison of centralized approach versus decentralized approaches with different feature sharing caps (FSC) for the PST scenario where the model perturbation is 20% from the nominal model. Results are averaged over 100 runs.

Table 4.4: Evaluation of averaged mission metrics for centralized planner with homogeneity assumption and coupled GA-Dec-MMDP Dec-iFDD planner with FSC = 100.

Planner	# Failures	# Base Visits
Centralized Homogeneous	90.2	81.2
GA-Dec-MMDP with Dec-iFDD	55.1	125.4

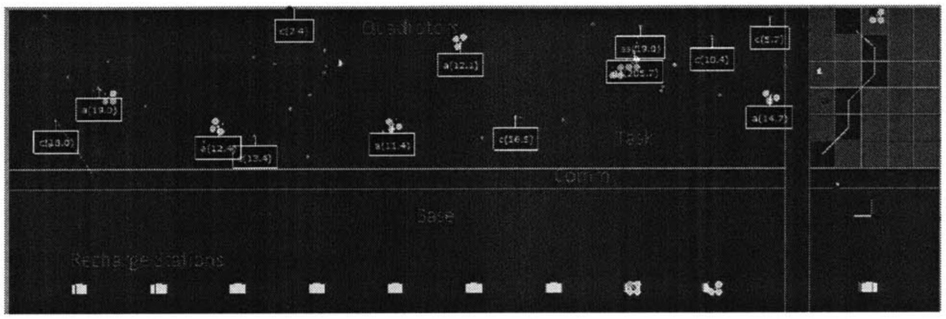


Figure 4-14: Visual representation of the test environment showing Base, Communication and Tasking areas. The real quadrotor is constrained to operate in the RAVEN environment (right part of figure), and the rest of the team (9 quadrotors) operate in the simulated area to the left. The recharge station is shown at the bottom.

Finally, it is possible to compare feature sharing to an alternative approach in which all observations are shared by estimating the average number of shared features, in terms of communication cost. Let κ be the size of a message that can be passed in the network, which corresponds to an single integer. Thus, each initial feature and a component of the state vector corresponds to messages of size κ. The average number of features shared in the simulations were computed to be $\approx 4.2\kappa$. Hence the total load on the network is $4.2\phi_{cap}\kappa$, since he maximum amount of features shared in the network is limited to ϕ_{cap} by Algorithm 5. In the considered scenario, each observed state transition corresponds to 84κ sized messages. Hence if n agents share observations instead of features, they would need to send $84\kappa n$ messages. For the simulation results presented above, number of agents $n = 10$, thus the load on network for passing observations instead of features is around 840κ. Hence, even with a feature sharing cap of 100, sharing features require less load on network in terms of communication cost, compared to sharing observations. In addition, it is seen that size of observations scale linearly with number of agents, while the number of shared features is fixed for a constant $\phi_{cap}f$. Hence the designer can tune ϕ_{cap} solely based on the limits of the communication network, while direct sharing of measurements is limited by the size of the observations and the number of agents in the scenario.

4.3 Closed Loop Multiagent Learning: Collaborative Filtering Decentralized iFDD (CF-Dec-iFDD)

In the previous Sections, we have utilized the iFDD algorithm for solving learning problems in both single agent (Section 4.1) and multiagent domains (Section 4.2). Particularly, in the Dec-iFDD algorithm agents learn their individual models by using the stochastic gradient descent (Eq. 2.9) while iFDD (see Section 2.3.3) updates the set of features ϕ_i at every learning step. The communication (Eq. 2.6) is handled by letting each agent sort their most relevant features $\phi_{i,j}$ based on their weights $\theta_{i,j}$ and send a fraction of these top ranking features to other agents. Note that the number of features that can be shared at each step is limited by f_{cap}. The inherent problem with Dec-iFDD is the fact that communication is handled in a completely open loop manner, i.e., sending agents do not receive any feedback on if the transferred features actually have reduced the model error of the receiving agents. This Section presents a novel multiagent learning algorithm, referred to as Collaborative Filtering Decentralized Incremental Feature Dependency Discovery (CF-Dec-iFDD) which alleviates this issue by allowing agents send feedback to each other on model error reduction after the feature transfer. These feedback signals are maintained in a matrix format by each agent and matrix completion techniques borrowed from Collaborative Filtering research is applied to predict which agents are likely to benefit from other features. Hence, the overall feature sharing scheme becomes closed loop, which results in even faster convergence rate due to elimination of transfer of irrelevant features.

Along with the formal description of the algorithms, this Section also includes the theoretical analysis of closed loop versus open loop feature sharing, as well as the simulation results comparing these two approaches in two different large-scale multiagent planning/learning problems.

4.3.1 The Algorithm

The basic idea behind the new algorithm, Collaborative Filtering Decentralized iFDD (CF-Dec-iFDD), is to allow receiving agents to send feedback to sending agents on the model error reduction rate of the shared features, so that the sending agents can make better predictions on which features should be shared in the next communication step. This idea is realized through the use of matrix completion algorithm described in Section 2.4 (also see Fig. 4-15). At every communication step, agents update a sparse matrix with entries consist

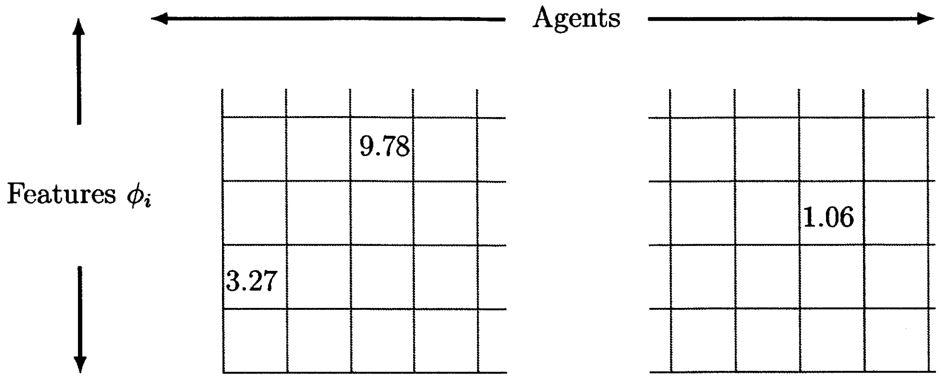


Figure 4-15: Collaborative filtering applied to the model sharing problem. The matrix entries are the weights $\theta_{i,j}$ of each feature from the perspective of the individual agent, the empty entries are features yet to be discovered by each agent. Collaborative filtering techniques generate predictions of these empty entries, allowing each agent to suggest useful features to other agents.

of weights of features of each agent. Then in the communication step, each agent solves the optimization problem in Eq. 2.13 to predict the missing entries in the matrix. This allows agents to send features that are possibly more relevant to the receiving agents, since the shared features are selected based on the feedback history provided by the receiving agents. The psuedocode is given in Algorithm 6.

The algorithm is initialized by setting the same set of initial features ϕ_{init} for each agent. Agent i's rating matrix M^i and the prediction matrix X^i are initialized with zero matrices. Next, the algorithm goes through the following steps at every iteration;

1. (Lines 5-9): If the current step is a communication step, agent i sends and receives features from its neighbors $N(i)$ in the network. For each receiving agent $j \in N(i)$, agent i takes the j-th column of the prediction matrix (which is denoted as X^i_j), and sorts the corresponding features according to their weights. More precisely,

$$\{\phi\}^j_i \leftarrow \underset{\{\phi_{i,l_1},\ldots,\phi_{i,l_{fcap}}\}}{\operatorname{argmax}} \sum_{l=l_1}^{l_{fcap}} |\theta_{j,l}|, \quad \theta_{j,l} \in X^i$$

   where $\{\phi\}^j_i$ denotes the features agent i is sending to agent j. After sending $\{\phi\}^j_i$ to agent j, agent i receives features $\{\phi\}^i_j$ from agent j likewise (Line 8) After that, agent i adds the received features $\{\phi\}^j_i$ to its feature set (Line 9).

2. (Lines 10-12:) After receiving new features $\{\phi\}^j_i$, agent i updates its weights θ_i accord-

Algorithm 6: Collaborative Filtering Decentralized iFDD Model Learning Algorithm (CF-Dec-iFDD)

```
Input: Number of Agents $n_a$, Initial Binary Features $\phi_{init}$, Shared Feature Cap $f_{cap}$
Output: Estimated model $\tilde{T}_i$ for each agent $i = 1, ..., n_a$
1  Initialize $\phi_i = \phi_{init}$, $\theta_i = 0$, $M^i = 0$, $X^i = 0$
2  foreach Step do
3      foreach Agent i do
4          if Step == Communication Step then
5              foreach Agent $j \in N(i)$ do
6                  $\{\phi\}_i^j \leftarrow$ Top $f_{cap}$ features from $X_j^i$
7                  Send $\{\phi\}_i^j$ to agent $j$
8                  $\{\phi\}_i^j \leftarrow$ Listen()
9                  $\phi_i \leftarrow \phi_i \cup \{\phi\}_i^j$
10                 Set weights $\theta_i$ according to $\{\phi\}_i^j$
11                 Send $M_i^i$ to agent $j$, receive $M_j^j$
12             Solve Eq. 2.13 with $M^i$ to update $X^i$
13         else
14             $s_l, t_l \leftarrow$ Observe Data
15             $\theta_i, \phi_i \leftarrow$ Run Eq. 2.9 with iFDD$(s_l, t_l)$
```

ing to the new features,

$$\theta_{i,p} = \sum_q \theta_{i,q}, \qquad \wedge_q \phi_{i,q} = \phi_{i,p}, \qquad \phi_{i,p} \in \{\phi\}_i^j,$$

where $\wedge_q \phi_{i,q} = \phi_{i,p}$ represents that the feature $\phi_{i,p}$ is generated by taking conjunctions of features $\phi_{i,q}$. After the weights are set, the new weights populate the column M_i^i, which are sent to agent j as a feedback on how much the transferred features ϕ_i^j have reduced the estimation error. Similarly, agent j sends its own rating column M_j^j (Line 10).

Finally, after receiving M_j^j from all its neighbors, agent i solves the optimization problem in Eq. 2.13 to update the prediction matrix X^i.

3. After the communication step, agents receive new samples (s_l, t_l) and use the iFDD algorithm (Section 2.3.3) along with stochastic gradient descent (Eq. 2.9) to update both the features ϕ_i and weights θ^i. The algorithm continues to run until convergence or manual termination.

Lines 9-11 are the points where CF-Dec-iFDD departs significantly from its predecessor

Dec-iFDD. In the Dec-iFDD algorithm, agents send and receive features from each other only once in each communication step, whereas in the CF-Dec-iFDD algorithim, an extra two-way communication is established to update matrices M^i through matrix completion. These matrices are the main elements that enable improved quality (in terms of model error reduction rate) in the feature sharing, since they represent an estimation of which features other agents are likely to benefit from. Note that the algorithm makes a delicate trade-off between the communication cost and the model quality. Basically, CF-Dec-iFDD enables learning better models by allowing more communication between agents. Section 4.3.3 represents simulation results on how well the CF-Dec-iFDD captures the matrices M for varying communication bandwidth and the number of samples.

4.3.2 Analysis of Closed Loop Learning

This Section compares the closed loop (CF-Dec-iFDD) and open loop learning (Dec-iFDD) in a theoretical manner. The main aim is to show that the closed loop learning is more likely to accelerate the overall learning process under certain heterogeneity assumptions.

In order to make the analysis cleaner, we will study a simplified 2-agent scenario with a single round of learning and exchanging features across agents. Note that this simplified learning scenario is the basic building block for the both CF-Dec-iFDD and Dec-iFDD algorithms, hence before extending the analysis to more than 2 agents and multiple rounds of learning, it is necessary to analyze this case.

Formally, define the Φ as the pool of all possible features. Since only binary features and their conjunctions are considered in the iFDD framework, it is possible to break down Φ into union of these two sets,

$$\Phi = \Phi_{init} \cup \Phi_{conj},$$

where $\Phi_{init} = [f_1, f_2, \dots, f_{n_f}]$ is a finite set of features and Φ_{conj} is the set of all their conjunctions. In other words, let $\Phi_{conj,2}$ be the set of all 2-conjunctions,

$$\Phi_{conj,2} = [f_i \wedge f_j : f_i, f_j \in \Phi_{init}],$$

hence $\Phi_{conj} = \bigcup_{i=2}^{n_f} \Phi_{conj,i}$.

It is assumed that the Agents 1's and Agent 2's true representations are formed from this pool and both their representations share the same set of initial features Φ_{init} and a subset of the Φ_{conj}. That is,

$$\mathcal{T}^1 = \theta^{1'} \Phi^1, \text{ and } \mathcal{T}^2 = \theta^{2'} \Phi^2, \tag{4.6}$$

where

$$\Phi^i = \Phi_{init} \cup \Phi^i_{conj}, \text{ and } \Phi^i_{conj} \subset \Phi_{conj}, i = 1,2, \tag{4.7}$$

in which $\mathcal{T}^i i = 1,2$ are the true representations of Agents 1 and 2, $\theta^1 \in \mathbf{R}^{n_1}$, $\theta^2 \in \mathbf{R}^{n_2}$ are the weight vectors, $n_1 = |\Phi^1_{conj}|$ and $n_2 = |\Phi^2_{conj}|$ denotes the number of conjunction features in the actual representations.

Let $\tilde{\mathcal{T}}^i, i = 1,2$ represent the current estimate of agents representations. Accordingly, let $\tilde{\Phi}^1 \subset \Phi_1$ and $\tilde{\Phi}^2 \subset \Phi_2$ be the features discovered so far.

We analyze the step where the Agent 1 sends a feature to the Agent 2. The main objective of the analysis is to examine how much error reduction the transferred feature induces on the estimated model of the Agent 2. Assume that the communication is limited; Agent 1 can pass only one feature $f_{1,2} \in \tilde{\Phi}^1_{conj}$ to Agent 2. Note that the transferred feature always comes from the conjunction set, since both agents have the same set of initial features.

We look at three different methods of how Agent 1 can choose $f_{1,2}$:

- **Random Approach:** Agent 1 selects $f_{1,2}$ from the elements of the set Φ^1_{conj} with equal probability. That is,

$$Prob\left(f_{1,2} = f, \quad f \in \Phi^1_{conj}\right) = \frac{1}{n_1} \tag{4.8}$$

- **Open Loop Approach (Dec-iFDD):** Agent 1 selects $f_{1,2}$ as the feature that is most heavily weighted in its representation,

$$f_{1,2} = f_j, \quad j = \underset{j=1,\dots,n_1}{\operatorname{argmax}} |\tilde{\theta}^1_j| \tag{4.9}$$

- **Closed Loop Approach (CF-Dec-iFDD):** Agent 1 maintains an estimate of the Agent 2's weights, which are denoted as $\tilde{\theta}^{1,2}$. Closed Loop feature transfer consists of two separate stages. In stage 1, Agent 1 performs matrix completion (See Section 2.4) to update the estimate $\tilde{\theta}^{1,2}$ and selects $f_{1,2}$ that corresponds to the most heavily weighted feature in the representation,

$$f_{1,2} = f_j, \quad j = \underset{j=1,\dots,n_1}{\operatorname{argmax}} |\tilde{\theta}^{1,2}_j| \tag{4.10}$$

  In the second stage, Agent 2 sends back its weight $\tilde{\theta}^2_{f_{1,2}}$, that corresponds to the transferred feature $f_{1,2}$. Then Agent 1 updates $\tilde{\theta}^{1,2}$ using the matrix completion algorithm (See Section 2.4). In the subsequent iterations, Agent 1 uses the updated representation. Note that this approach requires twice as much communication at every iteration compared to the previous approaches, since the communication is performed both ways.

It is evident that if $f_{1,2} \in \Phi_2$, then the representation of Agent 2 will improve. Hence in order to compare the three approaches above, we need to compute the probability $Prob(f_{1,2} \in \Phi_2)$. The computation of this probability depends on how much the feature sets Φ^1_{conj} and Φ^2_{conj} are similar to each other. We will define two similarity metrics, one based on the number of common features and another one based on the sorting of initial weights.

The first similarity metric η_ϕ simply counts the number of common features in the conjunction set and divides by the cardinality of the smallest set for normalization.

$$\eta_\phi = \frac{|\Phi^1_{conj} \cap \Phi^2_{conj}|}{\min(n_1, n_2)}. \tag{4.11}$$

Hence, if $\eta_\phi = 1$ then two representations have exactly the same set of features($\Phi^1 = \Phi^2$). On the other hand, if $\eta_\phi = 0$ then their conjunction sets have no common features.

To compare the weights between two representations $\mathcal{T}^1$ and $\mathcal{T}^2$, it is sufficient to look at the weights for the initial features θ^i_{init}. This is because iFDD weights the conjunction features proportionally to the weights of the conjunction features. In other words,

$$\theta_{f_1 \wedge f_2} \propto |\theta_{f_1}| + \theta_{f_2}|. \tag{4.12}$$

Let the function $Sort : \mathbb{R}^{n_{init}} \to \{1, 2, \dots, n_{init}\}$ return the indices of the elements of θ in descending order with respect to their absolute values. The second similarity metric η_θ checks

$$\eta_\theta = \frac{\sum_{i=1}^{n_{init}} \mathbb{1}(Sort(\theta^1_{init})_i = Sort(\theta^2_{init})_i)}{n_{init}}, \tag{4.13}$$

where $\mathbb{1}$ is the indicator function that returns 1 if $(Sort(\theta^1_{init})_i = Sort(\theta^2_{init})_i)$ and zero otherwise.

The following Theorem computes the $Prob(f_{1,2} \in \Phi^2)$ for the three approaches mentioned above, depending on the magnitude of η_θ and η_ϕ.

Theorem 4.3.1 *Let two agents true representations be given as in Eq. 4.6 and Eq. 4.7. Let Agent* 1 *pass the feature* $f_{1,2} \in \tilde{\Phi}^1$ *to Agent* 2 *to after* 1 *round of communication. Then the following holds depending on how* $f_{1,2}$ *is selected,*

- ***Random Approach:***

$$Prob(f_{1,2} \in \Phi^2) = \eta_\phi \tag{4.14}$$

- ***Open Loop Approach:***

$$Prob(f_{1,2} \in \Phi^2) = \eta_\theta \tag{4.15}$$

- ***Closed Loop Approach:***

$$Prob(f_{1,2} \in \Phi^2) = 1 - \frac{\eta_\phi}{n_{prev} + 1}, \tag{4.16}$$

where n_{prev} is the number of previous communications prior to this round.

Proof The probability that a random feature from the set Φ^1_{conj} will also be contained in Φ^1_{conj} is equal to the η_ϕ. Hence Eq. 4.14 follows from the definition of η_ϕ, which is given in Eq. 4.11.

From the Eq. 4.15, it is seen that the open loop approach will choose the feature $f_{1,2}$ that is most heavily weighted in Φ^1_{conj}. According to Eq. 4.12, this also corresponds to the feature that is most heavily weighted in the initial representation. Hence the probability that $f_{1,2} \in \Phi^2$ is simply the same as probability that the absolutely maximal weights of two representations are the same; i.e. $\arg\max_i |\theta^1_i| = \arg\max_i |\theta^2_i|$. This is exactly what η_θ (See Eq. 4.13) computes, hence Eq. 4.15 follows.

In the closed loop approach, first consider the case where $n_{prev} = 0$, i.e. the current round is the first round of communication. The only case $f_{1,2} \notin \Phi^2$ could happen is when Agent 2 reports back $\theta^{1,2}_{f_{1,2}} = 0$. This is the only case where the matrix completion algorithm will not return any useful suggestions due to incomplete data. $\theta^{1,2}_{f_{1,2}} = 0$ simply means that the feature $f_{1,2}$ is absent from Agent 2's representation, which can only happen with probability $1 - \eta_\phi$. Hence, Eq. 4.16 follows.

Theorem 4.3.1 hints which approach will be useful depending on the team heterogeneity assumptions. Based on the results of this theorem, we can classify the missions into 3 categories depending on which feature sharing approach is going to be useful.

- **Highly Homogeneous Scenarios:** This is the case where teammates actual representations have a high number of similar features. This case corresponds to the missions where agents are trying to learn the same distribution and/or they have highly similar transition dynamics. Hence $\eta_\phi \approx 1$ between any two teammates. In such scenarios, the random approach (Eq. 4.8) might be the best, since it is computationally very cheap and the probability that a randomly shared feature is going to belong to one of the teammates representation is significantly high. On the other hand, if the initial weights are also homogeneous (i.e. η_θ is close to 1) open loop approach will also work in the expense of performing more computation. Closed loop approach might not be preferable for these scenarios, because it is difficult to justify the extra computation and communication effort required for CF-Dec-iFDD, when the random approach just works as well.

- **Scenarios with Homogeneous Weights and Heterogeneous Features:** In this case $\eta_\theta \gg \eta_\phi$, hence sharing features randomly is not likely to succeed. This is the kind of scenarios where agents have dissimilar features due to their dynamics and operational conditions, but the features that are in common have similar weights. In this case the open loop approach (Dec-iFDD) might be the best option, due to high η_θ. The closed loop approach (CF-Dec-iFDD) also works, however it requires more computational resources and communication bandwidth.

- **Highly Heterogeneous Scenarios:** In this case both $\eta_\theta, \eta_\phi \ll 1$. These are the scenarios where the agents are learning highly dissimilar distributions, hence sharing features randomly or in an open loop manner is not likely to succeed. CF-Dec-iFDD (closed loop approach) is the best option in this case, since it is the only approach that figures out the common features between agents and the probability of success goes to 1.0 as more communications rounds are completed (See Eq. 4.16).

4.3.3 CF-Dec-iFDD Simulation Results

This Section compares the performance of CF-Dec-iFDD and Dec-iFDD algorithms on two large-scale multiagent problems.

Forest Fire Management Domain

This section applies the CF-Dec-iFDD algorithm to learn forest fire spread patterns. The stochastic forest fire model is from Boychuk [98], which is provided in the Appendix A. The domain was described previously in Section 3.5.

Fig. 4-16 shows the results for distributed learning of fire spread dynamics for 5 agents. Each agent learns on a distinct 20×20 grid. The average η_ϕ (See Eq. 4.11) for this scenario is 0.2, which corresponds heterogeneous team as described in Section 4.3.2.Agents first learn independently and then exchange features with each other using the CF-Dec-iFDD and Dec-iFDD[109] algorithms. Results show that allowing more features to be shared at each iteration eventually slows down the learning with Dec-iFDD, since processing more features with SGD takes more iterations. On the other hand, CF-Dec-iFDD has the best convergence rate among compared approaches, even tough it has the same f_{cap} as the best performing Dec-iFDD instantiate. This is due to fact that CF-Dec-iFDD shares features that are more relevant to each other due to utilization of the rating matrix completion performed by agents. Results also show that the distributed learning framework enables fire-spread model learning in large-scale environments, since the fire model in 20×20 grid corresponds to approximately 7.2×10^{16} parameters in the transition model.

Table 4.5: Comparison of the average model estimation error of different feature sharing approaches for instantiates of the forest fire domain with different heterogeneity levels. The resulst are obtained for 10^3 samples and averaged over 30 runs. Feature sharing cap for all algorthims are 200.

	Learning Algorithm		
η_ϕ	Random Sharing	Dec-iFDD	CF-Dec-iFDD
0.9	15.18±6.3%	22.82±5.1%	25.11±5.6%
0.5	25.17±9.2%	14.27±4.6%	10.11±3.4%
0.2	45.27±10.2%	18.24±3.2%	7.56±2.1%

The Table 4.5 represents the model estimation error of random sharing, Dec-iFDD (open-loop sharing) and CF-Dec-iFDD (closed-loop sharing) for different instantiates of the forest fire domain with different heterogeneity levels. These results agree with conclusions of the analysis performed in the Section 4.3.2. For homogeneous scenarios ($\eta_\phi = .9$), random approach yields the best results, since all the shared features are likely to be included in the true representations of the other agents. However, as the heterogeneity among the team increases, CF-Dec-iFDD yields the best model estimation error among the compared approaches.

Fig. 4-18 displays the matrix completion error as a function of the number of collected samples for different feature sharing caps. These results show that with increasing number of samples and the number of shared features, CF-Dec-iFDD estimates the matrices M^i (See Section 4.3.1) with higher precision, which in turn results in better learning performance.

Persistent Search and Track Domain

Fig. 4-17 shows the planning/learning performance of CF-Dec-iFDD for a 10 agent PST mission with uncertain actuator dynamics. GA-Dec-MMDP was used as the planner, and cumulative cost was averaged over 30 runs. The results are similar to the fire spread learning results. Dec-iFDD learning slows down as number of shared features increases, which leads to inferior performance. Whereas CF-Dec-iFDD yields better planning performance compared to open loop sharing with same comm constraints. The Table 4.6 represents the model estimation error of random sharing, Dec-iFDD (open-loop sharing) and CF-Dec-iFDD (closed-loop sharing) for different instantiates of the persistent search and track domain with different heterogeneity levels. Fig. 4-19 displays the matrix completion error as a function of number of collected samples for different feature sharing caps. These results reinforces the idea that the closed loop sharing (CF-Dec-iFDD) is especially useful for scenarios with heterogeneous teams, which is due to the precise estimation of M^i matrices with increasing number of samples.

Table 4.6: Comparison of the average model estimation error of different feature sharing approaches for instantiates of the persistent search and track domain with different heterogeneity levels. The resulst are obtained for 10^3 samples and averaged over 30 runs. Feature sharing cap for all algorthims are 200.

	Learning Algorithm		
η_ϕ	Random Sharing	Dec-iFDD	CF-Dec-iFDD
0.9	10.23±3.2%	15.33±5.1%	19.22±3.2%
0.5	12.43±7.2%	8.62±4.6%	8.22±2.2%
0.2	36.78±9.1%	20.12±3.2%	3.44±1.1%

4.4 Summary

This Chapter provided a family of single and multiagent model learning methods built upon the iFDD algorithm. First, the convergence guarantees of iFDD were proven and the algorithm was shown to be an efficient single agent approximate learning algorithm for moderately sized domains. Dec-iFDD algorithm enabled learning in large-scale multiagent scenarios, and its applicability was demonstrated through simulation results for both robotics and UAV domains. Finally, CF-Dec-iFDD algorithm was introduced, which enables planning for highly heterogeneous teams, in the expense of performing more communication and computation. Both theoretical results and simulation results were provided for CF-Dec-iFDD. Results showed that CF-Dec-iFDD yields better performance than Dec-iFDD for highly heterogeneous scenarios. The next Chapter extends the results to real flight experiments in order to demonstrate the applicability of the developed learning algorithms in real-world scenarios.

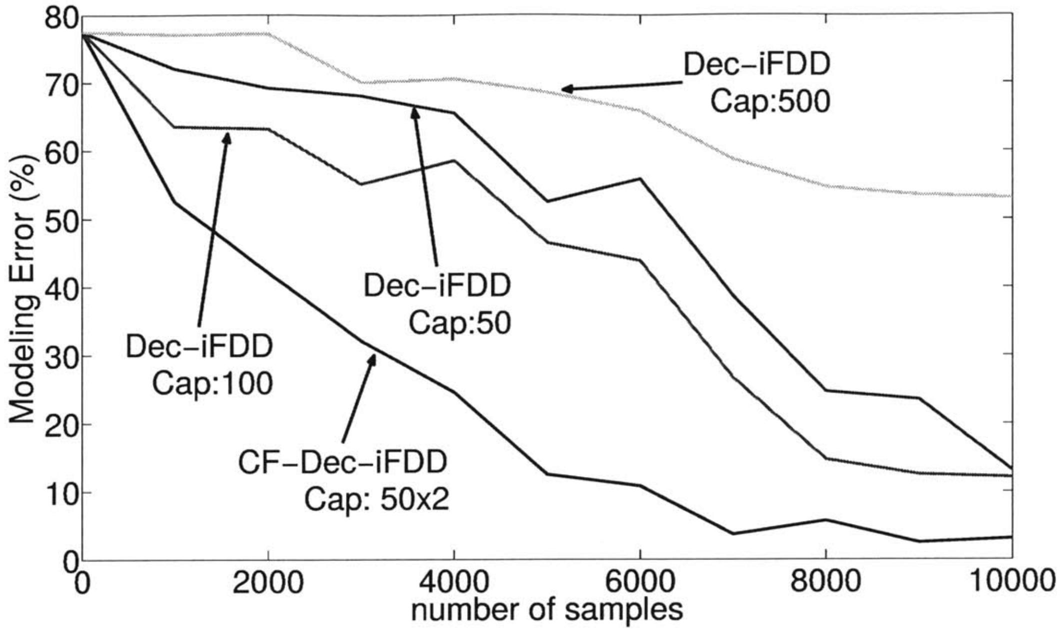


Figure 4-16: Comparison of modeling error of the Dec-iFDD and CF-Dec-iFDD algorithm for a 5 agent fire spread simulations with different feature cap sizes. Cap represents the number of features allowed to be exchanged at each iteration. Since agents communicate twice as much in CF-Dec-iFDD the cap is halved for a fair comparison.

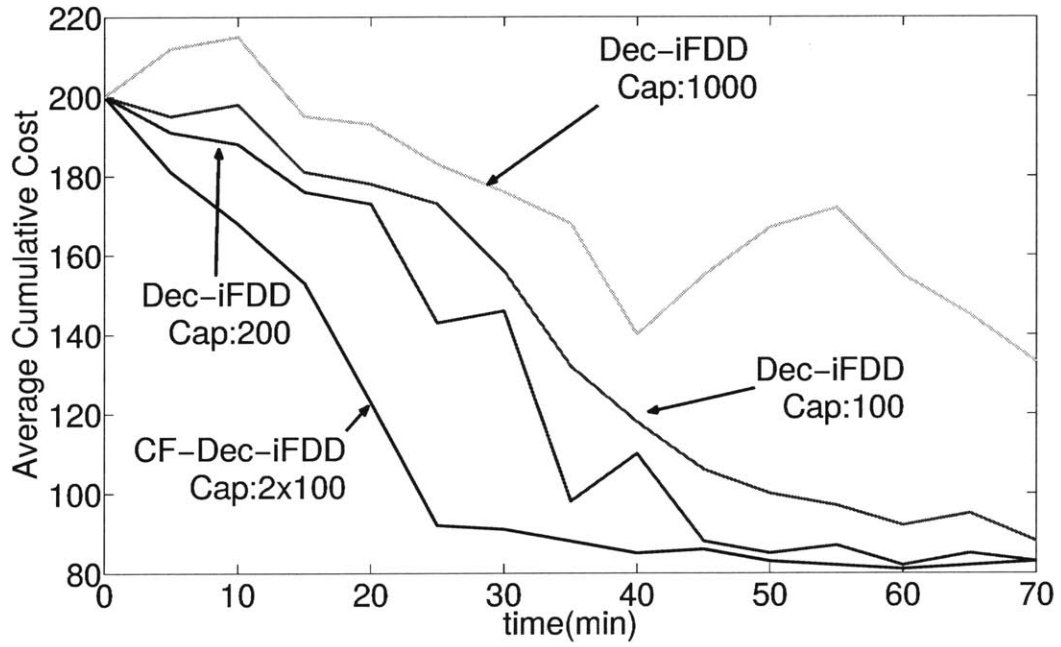


Figure 4-17: Comparison of average cumulative cost of the Dec-iFDD and CF-Dec-iFDD algorithm for a 10 agent PST simulations with different feature cap sizes. Cap represents the number of features allowed to be exchanged at each iteration. Since agents communicate twice as much in CF-Dec-iFDD the cap is halved for a fair comparison.

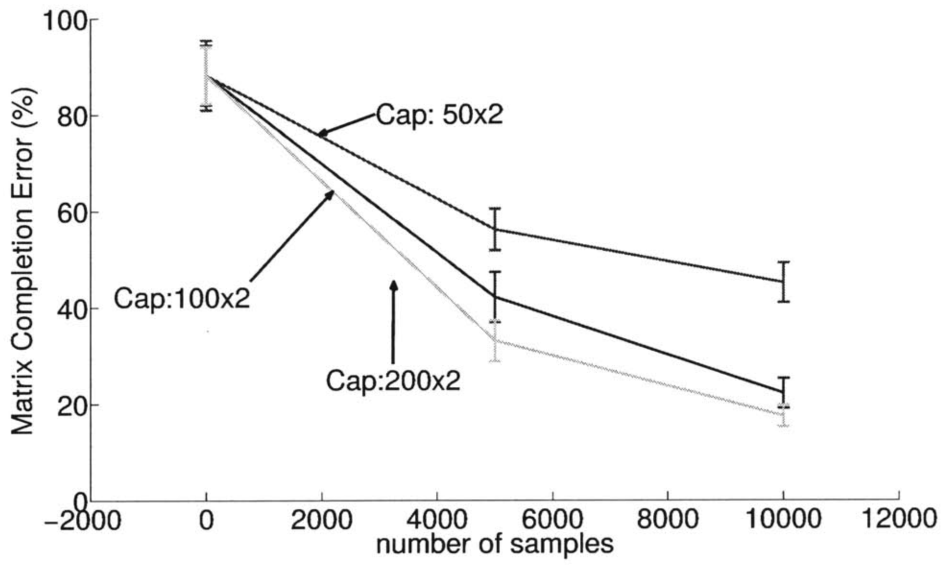


Figure 4-18: The matrix completion error as a function of number of collected samples for different feature sharing caps on the forest fire domain.

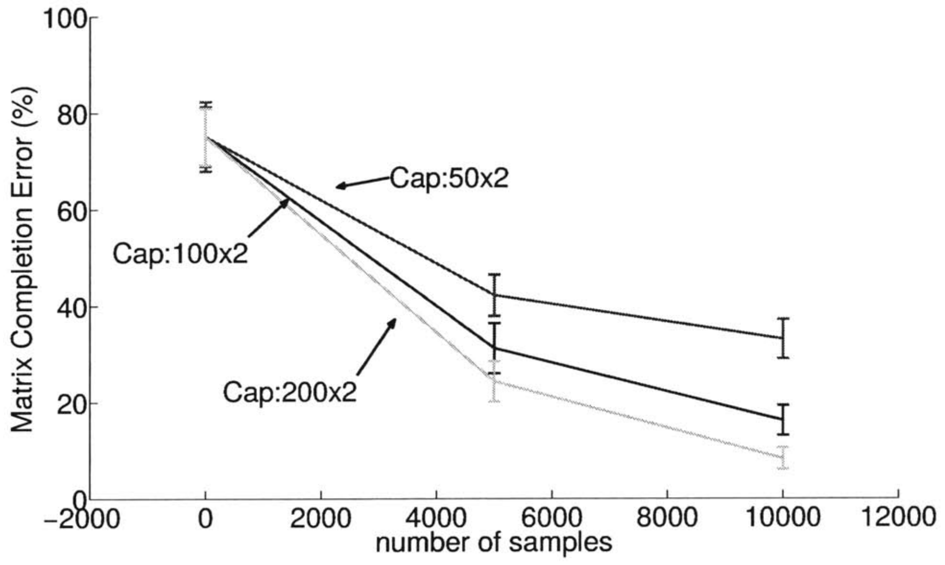


Figure 4-19: The matrix completion error as a function of number of collected samples for different feature sharing caps on the persistent search and track domain.

Chapter 5

Flight Experiments

This Chapter provides the results of the integrated planning/learning indoor flight experiments conducted in MIT's RAVEN facilities. The main objective of these experiments is to demonstrate that the developed learning algorithms are able to work in real-time and tolerate uncertain elements such as wind gusts and sensor noise. The Chapter also gives details on a number of novel experimental devices developed in order to conduct these experiments. These devices include an autonomous battery recharge station for enabling long endurance missions and ceiling mounted laser projectors for displaying the belief space of the agents.

The Chapter is organized as follows. Section 5.1 describes the testbed and its components. Sections 5.2 and 5.3 provides the experimental results for iFDD and Dec-iFDD respectively. Finally, Section 5.4 presents the results for the integrated planning/learning using RCD/CF-Dec-iFDD algorithms.

5.1 Testbed

The flight experiments were conduced in the Aerospace Controls Lab indoor facility RAVEN [110, 111, 112] (see Figure 5-7). RAVEN uses realistic small-scale air and ground vehicles operating indoors with a motion capture system to emulate GPS. This Section gives the details on some of the key components of the testbed.

5.1.1 Quadrotor UAVs

To address the need for inexpensive aerial mobile robots, an in-house quadrotor UAV is developed to be used in flight experiments. The UAV, shown in Fig. 5-1, is built using a carbon-fiber and foam sandwich plate frame (approximately length of 30 cm across) with brushless motors, electronic speed controllers manufactured by AscTec capable of measuring

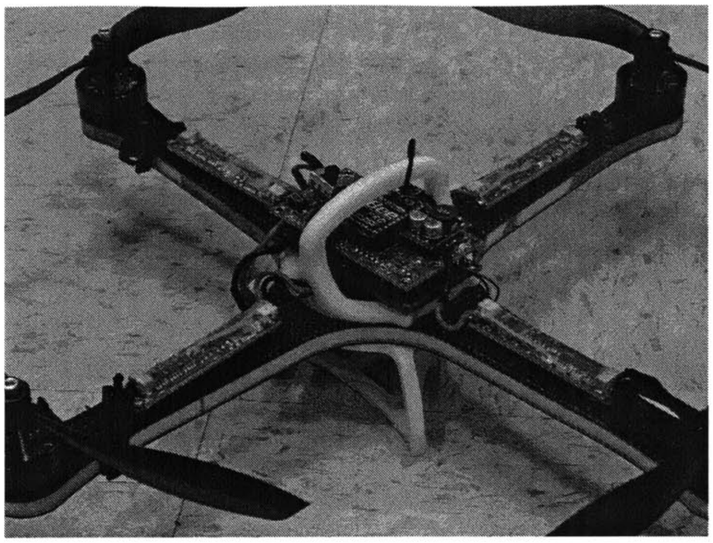


Figure 5-1: Quadrotor built in-house for flight experiments

temperature and current, and an off-the-shelf autopilot board with accelerometers, gyros and a pic-based microcontroller known as UAVDevBoard sold by Sparkfun Inc. The firmware for the autopilot was also developed in-house to close the attitude loop onboard for flight control [113]. An ADC circuit was used on UAVs to acquire the battery residue and and broadcast it via wireless modem.

The quadrotor is also mounted with an onboard camera (SONY 700 TVL FPV Ultra Low Light Mini Camera), a wireless transmitter (Boscam TS350 5.8G 10mW 8 Channel AV Transmitter FPV) and a 5.8GHz Circular Polarized Antenna antenna (see Fig. 5-2). This onboard camera allows us to gather data for image processing and emulate the camera sensing noise.

5.1.2 Autonomous Recharge Station for Enabling Long Endurance Missions

Given their relative safety and cost-effectiveness, the quadrotors described in the previous section are being used extensively in indoor flight facilities [111, 114, 115, 116]. However, the current battery technology only enables flights of 8–15 minutes (depending on vehicle, payload, and wind conditions) for a typical quadrotor [111, 117], which constrains the mission and experimental capabilities of these small-scale UAVs. To enable the capability of sustaining persistent missions, and, in the process, enable research concerned with developing autonomous planning and learning algorithms that depend on data obtained from long-term operations, there is a need to develop a system that can sustain long duration flight. This

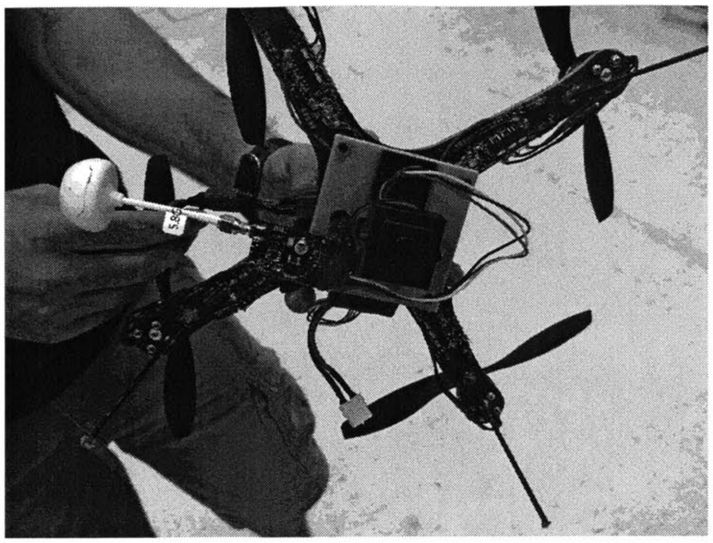


Figure 5-2: The onboard camera, wireless transmitter and antenna mounted on the quadrotor.

Section presents a such system for small-scale UAVs, consisting of an autonomous battery change/recharge station that is equivalent to a refueling station. The device integrates numerous mechanical and electronic components orchestrated through supporting software.

Fig. 5-3 shows the complete platform for the automated battery recharge system. The station has a central stack (behind the circuit board) that houses the motors to turn the two battery drums and supports the top-mounted landing pad. The two drums on both sides of the stack rotate in either direction about a horizontal axis. Each drum consists of four battery bays, each of which can carry and recharge a single battery. The chargers for the batteries are located at the front – one for each drum.

In operation, a UAV modified with a battery receiver and carriage lands on the landing pad and is locked down securely with two arms that carry shore power to the UAV. The drums are rotated to align the appropriate battery bays around the shaft (as shown in Fig. 5-3) using a stepper motor. The battery on the UAV is then swapped out with a freshly charged battery using a linear swapping motion. The old battery is placed into an empty bay on the opposite drum. The charging for that battery then starts until that battery is needed (approximately 1 hour later given the current system). Since all steps are automated, the platform provides the capability to automatically change and charge batteries without requiring intervention of a human operator.

The carriage and receiver design is an integral part of the swapping mechanism. Note that the basic idea behind the entire design was to create a battery swapping process in which one linear motion performs the steps of removing the old battery and replacing it with a new

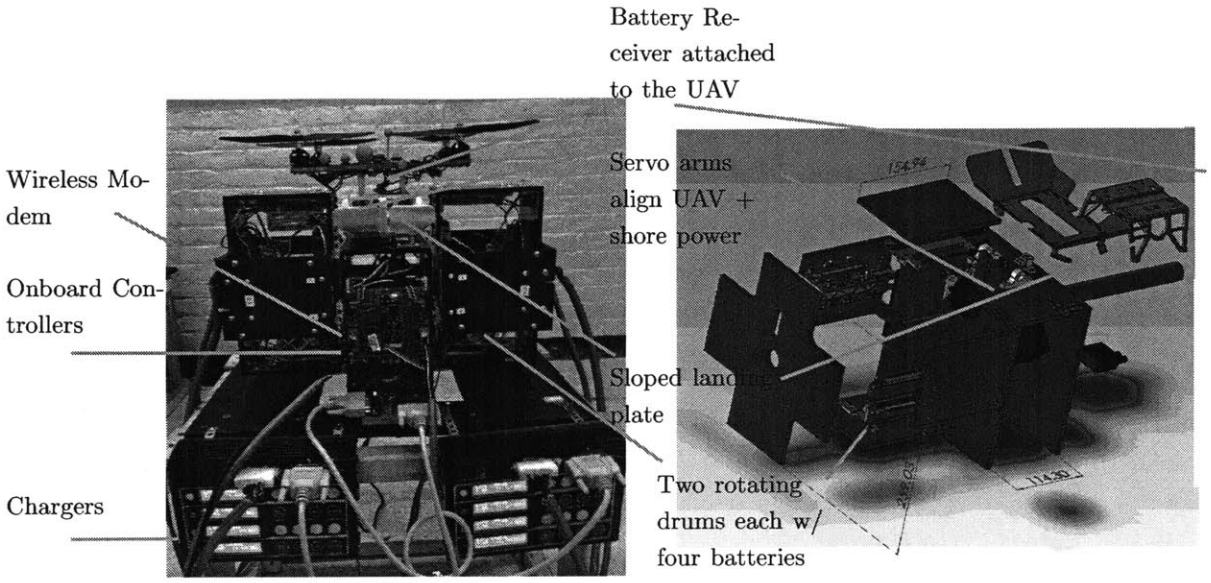
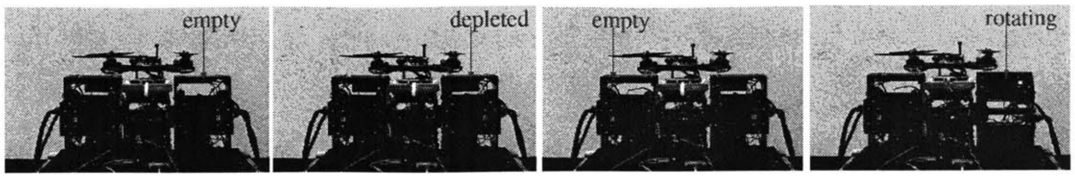

Battery Receiver attached to the UAV
Wireless Modem
Servo arms align UAV + shore power
Onboard Controllers
Sloped landing plate
Chargers
Two rotating drums each w/ four batteries
154.94
114.30

Figure 5-3: The recharge station designed to enable persistent flight of small UAVs. Left: The operational recharge station and its components. Right: Exploded view of the CAD diagram of the recharge station Only Left drum is shown for simplicity, all dimensions are in mm.

one. This is in contrast to other options that might have involved multiple steps, such as removing the old battery into an empty bay, aligning a new bay with fully charged battery, and then inserting the new battery. This may be how a human might replace a battery in a typical device, but it involves multiple dexterous steps that complicate the overall process. The alternative approach taken here was to align three bays (e.g, an empty bay on the right, the vehicle in the middle, and a bay holding a charged battery on the left, see Fig. 5-4). The aligned bays now provide a nearly continuous T-rail support from the far left to the far right on the device, with small gaps in between. Note that the two ends of the T-rail are beveled to ease the transition across the gaps from one rail support to another – it simplifies insertion into the new rail support, and then the T-rail can force the proper alignment as it moves across.

The battery swapping process in displayed step by step on Fig. 5-4. Videos of the recharge station and the experiment presented in Section 5.2 are available at [118].

More details on the design process of the recharge station can be found in [119, 120].



(a) After landing, quadrotor is locked to the landing pad and provided with shore power.

(b) The depleted battery in quadrotor is pushed into the empty spot.

(c) The fresh battery is pulled into the battery carriage under the quadrotor

(d) Locking arms released, the quadrotor is ready to take off. Drums rotating to align next battery

Figure 5-4: Steps in a battery exchange between the recharge station and the quadrotor.

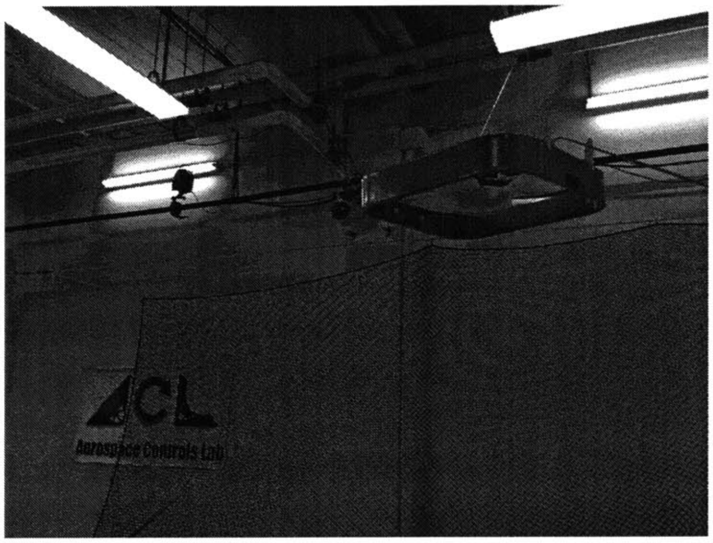


Figure 5-5: A box fan is hang from the ceiling to simulate the effect of localized wind.

5.1.3 Ceiling Mounted Fans for Wind Emulation

To implement the effect of localized wind, a 20-inch diameter box fan was hung from the ceiling in the RAVEN environment, as illustrated in Figure 5-5. On high power setting, the box fan can produce a downward wind flow of approximately 5 m/s over a 0.25 m^2 area. From combined on-board voltage and current reading, the system can reliably distinguish between windy and non-windy region.

5.1.4 Ceiling Mounted Projectors for Belief Space Projection

During execution of planning and learning algorithms, numerous latent variables such as probability distributions over system states, predicted agent trajectories, and transition probabilities are manipulated. Though hardware experiments allow spectators to observe

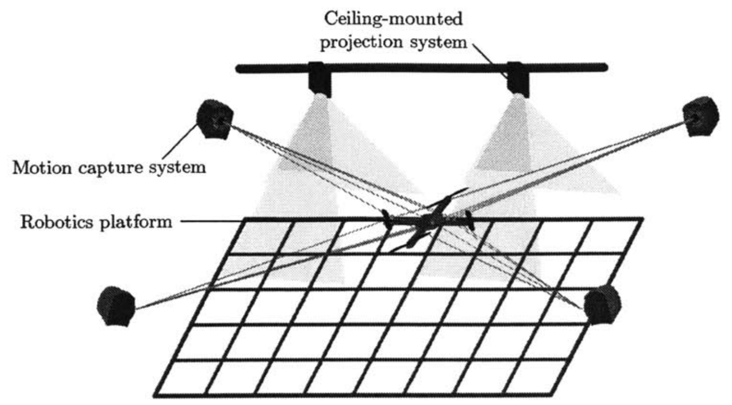


Figure 5-6: Hardware overview for the projection system

performance of such algorithms in the real world, it is difficult to simultaneously convey latent information alongside physical platforms. Apparent performance of algorithms relying on complex background processes can, therefore, suffer due to this complexity not being appropriately conveyed to spectators. In certain scenarios, once experimental data is gathered, latent information can be visualized through simulation software. However, it may be difficult for spectators to synchronously monitor behavior in the simulator and on the physical platform, especially in the case of real-time algorithms. It is much more beneficial to augment the experiment area with real-time visualization of this data, allowing direct perception of the progress of the planning/learning algorithm. The RAVEN is equipped with ceiling mounted projectors (See Fig. 5-6), which allows projection of belief space of planning/learning algorithms as well as animated dynamics of the environment. The benefit of the projection system is two-fold, it enables a more efficient monitoring of the experimental process and it also provides raw data to emulate camera sensing noise on quadrotors.

5.2 iFDD Learning Experiments

This Section presents fight test results for the PST mission in presence state-correlated sensor failure uncertainty with Dec-MMDP planning algorithm [7] integrated with iFDD learning algorithm (Section 4.1). The view of the experiment can be seen in Figs. 5-7 and 5-8. The setting and the state-correlated uncertainty representation is the same as discussed in Section 4.1.5. The autonomous flight test lasted for three hours, during which about 120 total autonomous battery swaps were performed by the the three recharge stations (Section 5.1.2). The parameters of the state correlated sensor failure model, as learned through

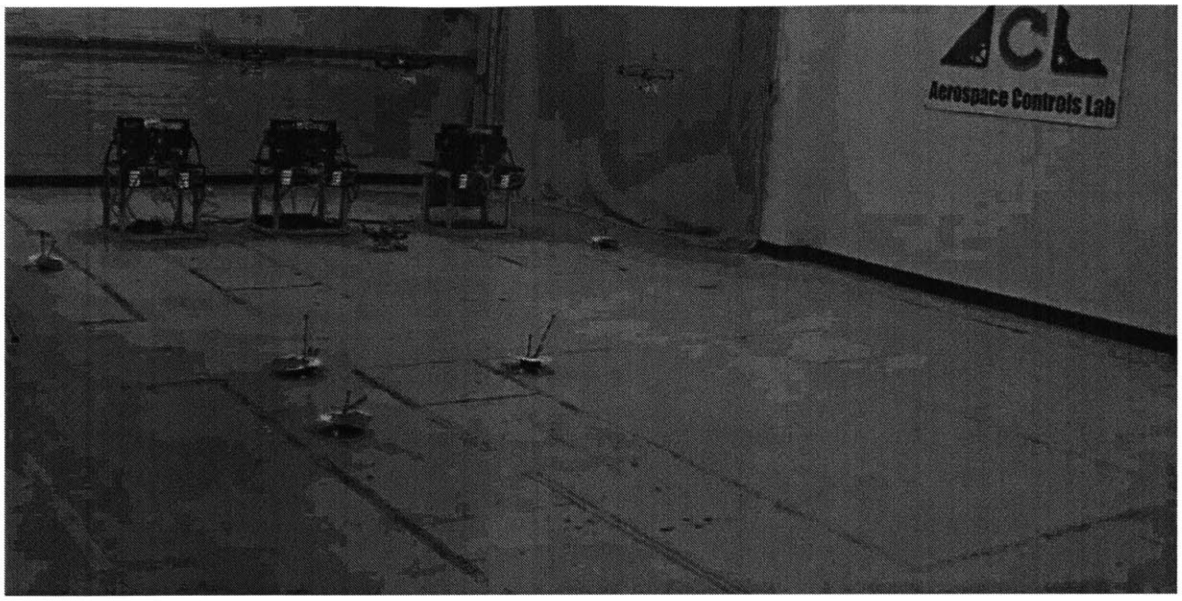


Figure 5-7: Experiment setup showing the four types of agent platforms used in the flight experiments: Team UAV (3) Team UGV (blue), Target UGV (green/red), and a Neutral UGV (white). The three change/recharge stations used to enable persistent flight operations for the UAVs are shown in the background.

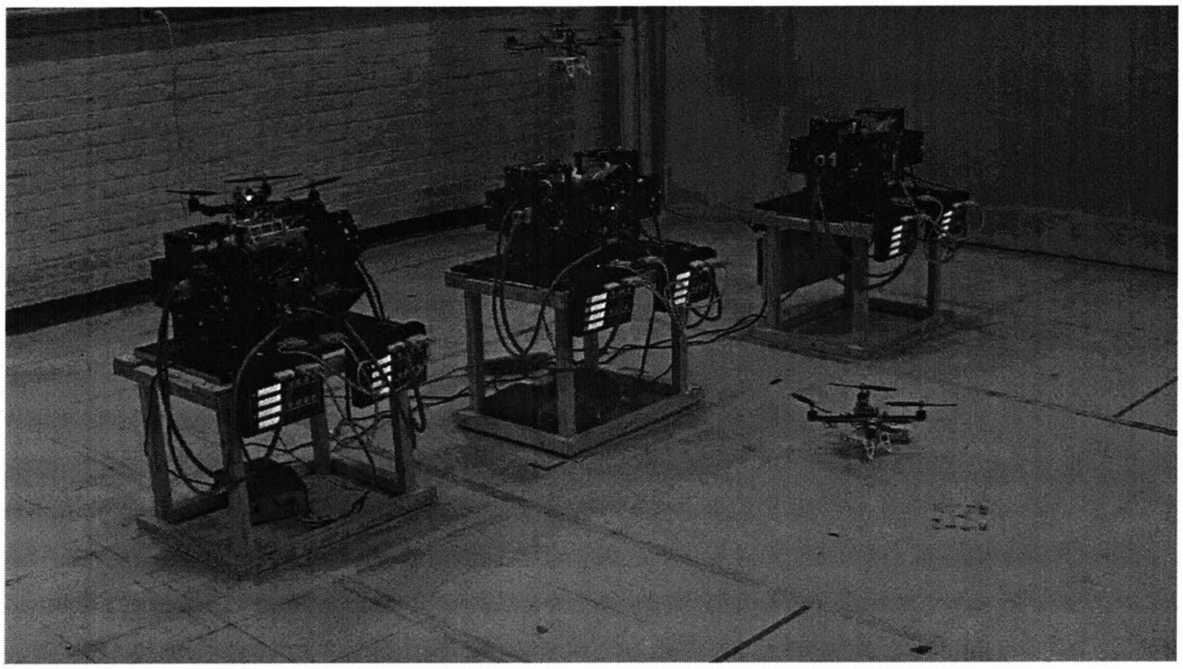


Figure 5-8: The three recharge stations (see Section 5.1.2) shown here were developed at MIT's Aerospace Control Laboratory and are designed to enable multiple-hour UAV missions.

Table 5.1: Probability of sensor failure for UAVs across different locations and fuel levels as estimated by iFDD learning algorithm.

	Fuel Level	
Location	High Fuel Level	Low Fuel Level
Base	0.0	0.0
Communication	0.067	0.132
Surveillance	0.123	0.351

iFDD is displayed in Table 5.1. Comparison of the estimated values to actual values in Table 4.1, shows that iFDD was able to estimate the parameters within $2-3\%$ accuracy on average. After every iFDD update of the uncertain parameters, the Dec-MMDP policy was recomputed using the new estimates. Model of the integrated learning planning framework was initialized 30% sensor failure probability across all states. This amounts to a pessimistic initialization of sensor failure probability. Figure 5-9 displays the performance of the updated policy after each learning cycle in terms of average cumulative cost. It is seen that, due to pessimistic initialization, initial policy has a high cost, because it calls UAVs back to base frequently for sensor repair. Note that, sensor status is a part of the state and UAVs with failed sensors/actuators do not return to base unless they are commanded to do so by the policy. As the iFDD learning algorithm estimates the parameters of the state dependent uncertainty, average accumulated cost decreases and the policy stabilizes around halfway through the mission.

Fig. 5-10 displays the average number of battery swaps every half an hour. The results show that initially UAVs are called back to base frequently, as reflected by high number of battery swaps. This is a result of pessimistic estimate of the probability of sensor failure. As the uncertainty parameters estimates are improved through iFDD learning, the planning algorithm becomes more confident in the UAVs' ability to operate without failures and assigns the UAVs more efficiently between the base and tasking areas. This is visible through relatively lower number of average battery swaps per half hour in the second half of the mission.

5.3 Dec-iFDD Learning Experiments

This Section presents the results of an hour long indoor flight test on a large-scale PST mission with 10 UAV agents, 10 recharge stations [7, 120], and several other ground agents. Of the 10 UAV agents, 9 are simulated (referred to as virtual agents) using physics based simulations, and 1 agent is physically present in the experiment area (referred to as real

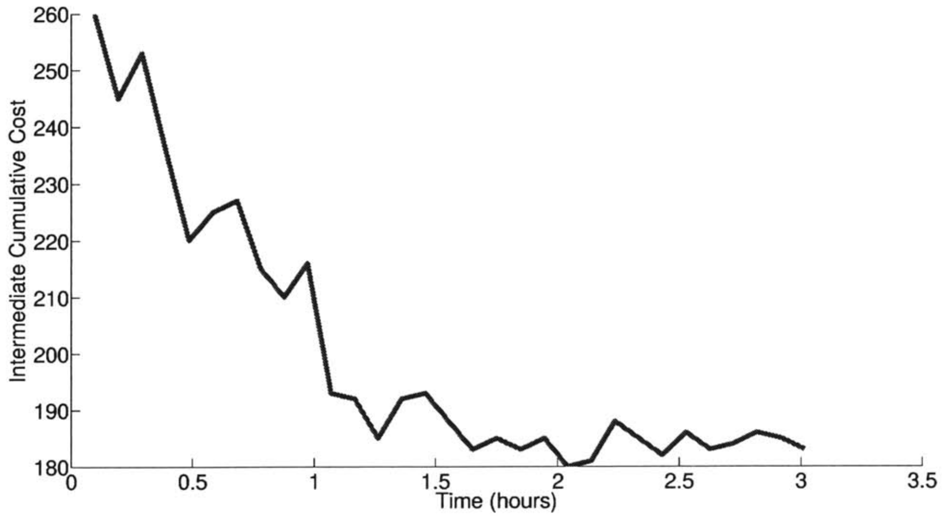


Figure 5-9: Flight test results for Dec-MMDP planner and iFDD learner. Plot shows the decrease of average cumulative cost as a result of learning process

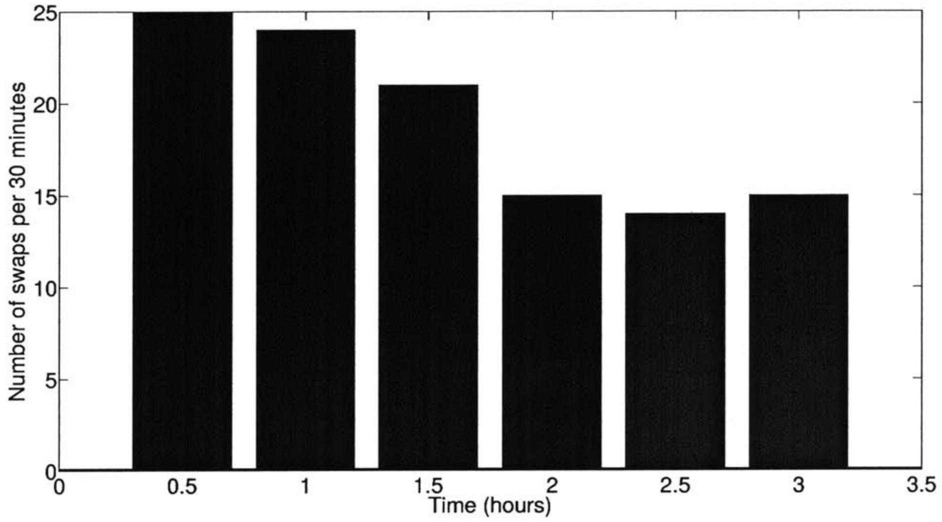


Figure 5-10: Number of battery swaps in recharge stations as a function of time. Decreased number of swaps indicate more efficiency UAV coordination between the base and surveillance area

agent). The GA-Dec-MMDP algorithm was used as the planner while the Dec-iFDD algorithm (Section 4.2) was used as the learner.

5.3.1 Experiment I: Single Fan and State-Independent Fuel Burn Rate Learning

In order to verify the applicability of the decentralized learning and planning approach developed in this thesis, an additional scenario where each UAV has a different state-dependent fuel burning rate was considered in experimental setting. State-dependency of the fuel burning rate was motivated from the possibility of having non-uniform wind in the mission environment and thus each region induces a different fuel burning rate. Table 4.3 shows the probability of incurring nominal fuel burn in the respective states. Else, the vehicle experiences twice the nominal fuel burn.

Note that the existence of a windy environment is not known to the planning-learning algorithm a priori, learning the existence of the wind and its impact on the fuel burning rate is the objective of the learning algorithm. The wind was emulated using ceiling mounted fans (Section 5.1.3).

Results of the experiment averaged over 4 different runs is displayed in Figure 5-11. As the learning algorithm learns a better model for the fuel burning rate, the performance of the policy produced by the planner for allocating UAVs between the base and surveillance locations improves. It can be seen that the planner computes a more efficient policy using the continually improving model. In particular, UAVs in windy areas are returned back to base more frequently to ensure that they do not run out of fuel and UAVs in non-windy areas are allowed to spend more time in the surveillance area to accomplish more tasks.

5.3.2 Experiment II: Two Fans and State-Dependent Fuel Burn Rate Learning

The test environment is depicted in Fig. 4-14. The mission environment is separated such that real and virtual agents do not cross each other's operational spaces. The actuator failure probabilities are set to the values used for the heterogeneous team model described in the Section 4.2.2, and the nominal fuel burn rate probabilities for all the virtual agents are set according to the heterogeneous team model described in the Section 4.2.2. The nominal fuel burn probability of the real agent is influenced by the custom setup of vertical fans implemented in the experiment area (see Figs. 5-12 and 5-13), which is explained later in this section. The GA-Dec-MMDP planner [7] is used to coordinate the 10 agents across

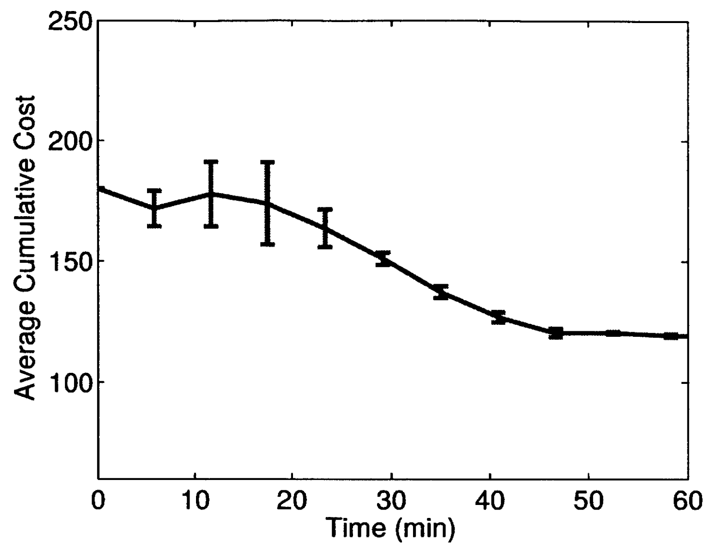


Figure 5-11: Results from a large scale persistent mission experiment with 9 simulated quadrotors and one physical quadrotor. A stochastic model of agent fuel burning rate is learned using decentralized iFDD learning. It can be seen that as the fuel burning model is learned better the cumulative mission cost is reduced.

the base, communication and tasking areas, however the health models are not known to the planner, and must be estimated during the experiment. Objective of the experiment is to show that the Dec-iFDD algorithm succeeds in learning these models and the GA-Dec-MMDP planner can lower the mission cost progressively.

The tasking area for the virtual agents is populated with randomly generated tasks, which are represented by the colored dots in Fig. 4-14. Allocation of these tasks among virtual agents are handled by the CBBA task allocation algorithm [121].

The tasking environment for the real agent is shown in Figs. 5-12 and 5-13. As depicted in these figures, the real agent has two static target observation tasks (task 1 has a higher reward than task 2) and the agent must fly through a region with localized wind disturbances, created using two vertically aligned fans, in order to perform the tasks. However the locations and the magnitude of the localized wind disturbances are unknown to the agent a priori. The wind disturbances generated by the fans significantly impact the battery life of the quadrotor, as can be inferred from Fig. 5-14. The figure shows that flying under the wind (blue line) results in approximately 17% more current drawn from the battery compared to not flying under the wind (red line), thus it can be stated that flying under the wind corresponds to higher battery discharge rate, which results in shorter flight times before returning to base for battery swap. We would like to emphasize that impact of the wind on the battery life of the agent is not known to the planner in the beginning of the mission.

The tasking space of the real agent is separated into several grids, with each grid inducing a different unknown fuel burn rate based on whether the fan above the grid is active or not. Since the wind can appear and disappear during the mission, it is more appropriate to model their effect probabilistically rather than having a binary wind or no wind value for each grid. Hence, the probability of nominal fuel burn rate p_{fuel} is a function of the agent's grid location. Therefore, p_{fuel} can be treated as a state-dependent uncertainty (see Section 4.2) and is learned here using the iFDD algorithm. In the actual experiment, during the period where the real agent is in the tasking area, the agent collects observations of battery usage at each grid it had traveled to. When the agent returns to the base, the iFDD algorithm processes this batch of observations to generate an estimate of $p_{fuel}(grid)$. This process is referred to as one iteration of wind learning. After each wind learning iteration, a dynamic programming algorithm is used to re-plan a trajectory that minimizes the based on the current estimate of $p_{fuel}(grid)$. The actual policy that the agent implements is randomly chosen from the optimal policy obtained from the DP algorithm or from a random policy, with the probability of choosing random actions reducing over time. This is an ϵ-greedy approach (see [28]), with $\epsilon = 0.2$, designed to encourage exploration.

In the beginning of the mission, only the Fan 1 is on and the fan on the the top of task 1 (Fan 2) is turned off. Fig. 5-15 displays a sequence of selected trajectories for the real agent during different stages of the learning process. At the first wind learning iteration, the planner routes the agent towards the task with the higher reward (Fig. 5-15a), during agents 3^{rd} visit to the tasking area The Fan 2 is manually turned on unknown to the agent. It can be seen that the learning algorithm identifies the non-zero probability of experiencing wind for this grid (Fig. 5-15b). After taking more observations of the environment, the planner discovers that doing task 1 corresponds to higher probability of non-nominal fuel burn rates and starts to route the agent towards task 2 (Fig. 5-15c). After 10 wind learning iterations, planner converges to a trajectory that has the least probability of burning non-nominal fuels (Fig. 5-15d). The overall planning performance of the whole team is presented in the average cumulative cost versus time plot on Fig. 5-16. These results are consistent with their simulation counterparts in Section 4.2.2, in the sense that the Dec-iFDD algorithm learns the actuator failure and fuel burn models across the team and the GA-Dec-MMDP planner is then able to provide improved (lower) average cost. These flight experiment and simulation results verify the applicability of the GA-Dec-MMDP planner and the Dec-iFDD learner for large-scale UAV missions with unknown health models and heterogeneous team structure.

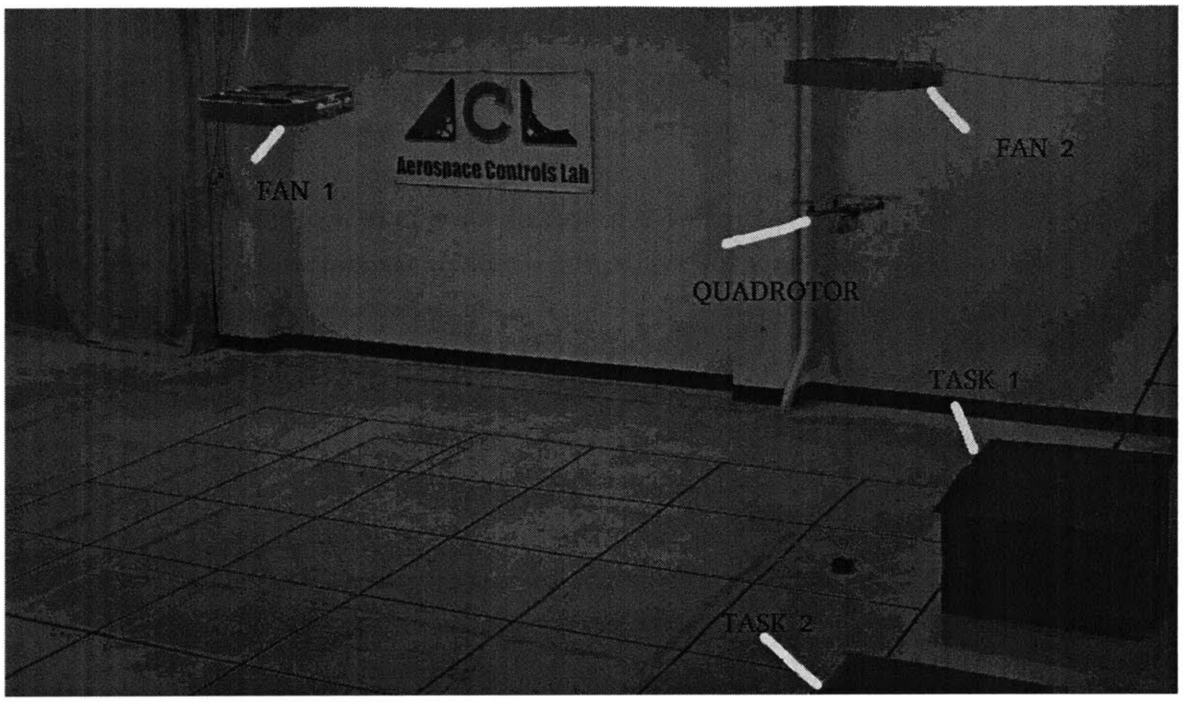


Figure 5-12: View of the Task area of the real agent.

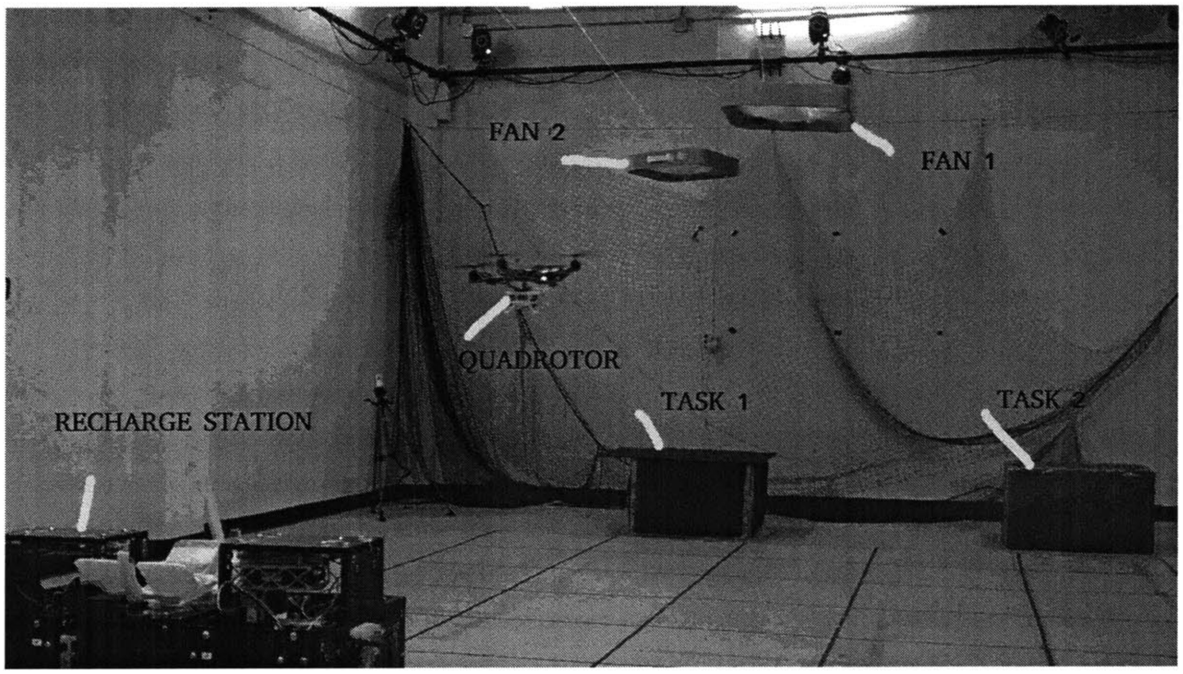


Figure 5-13: View of the Task area of the real agent from Base.

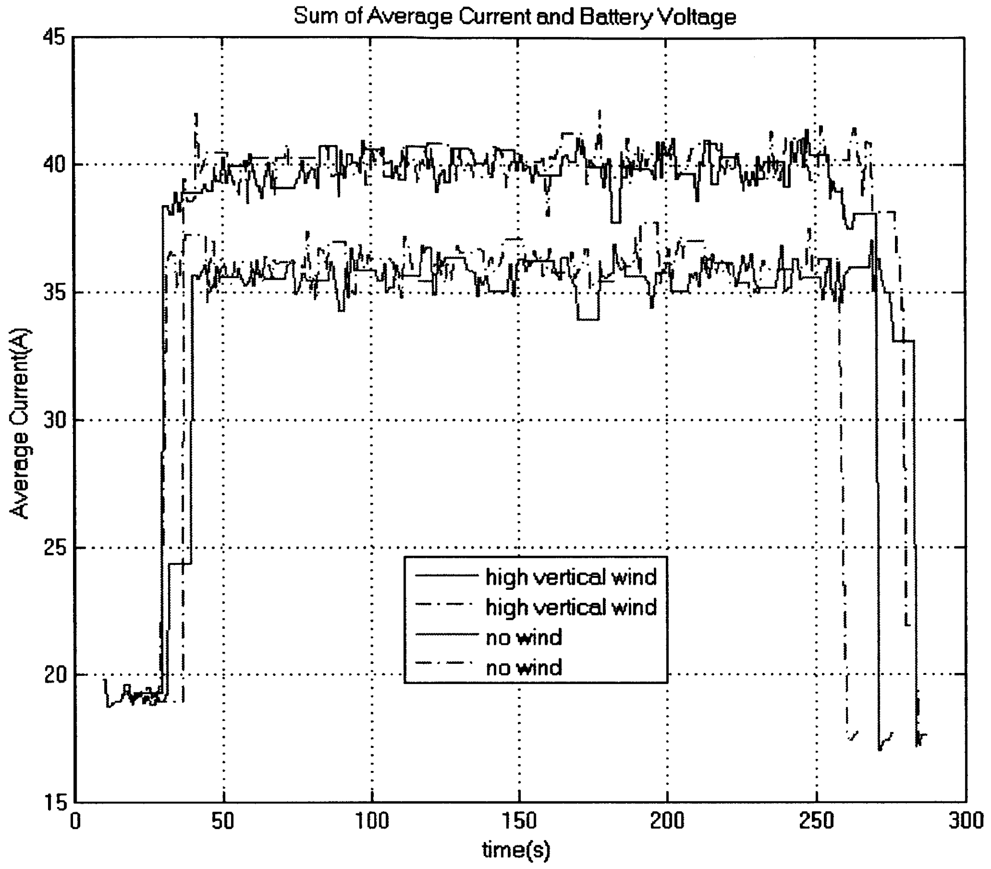
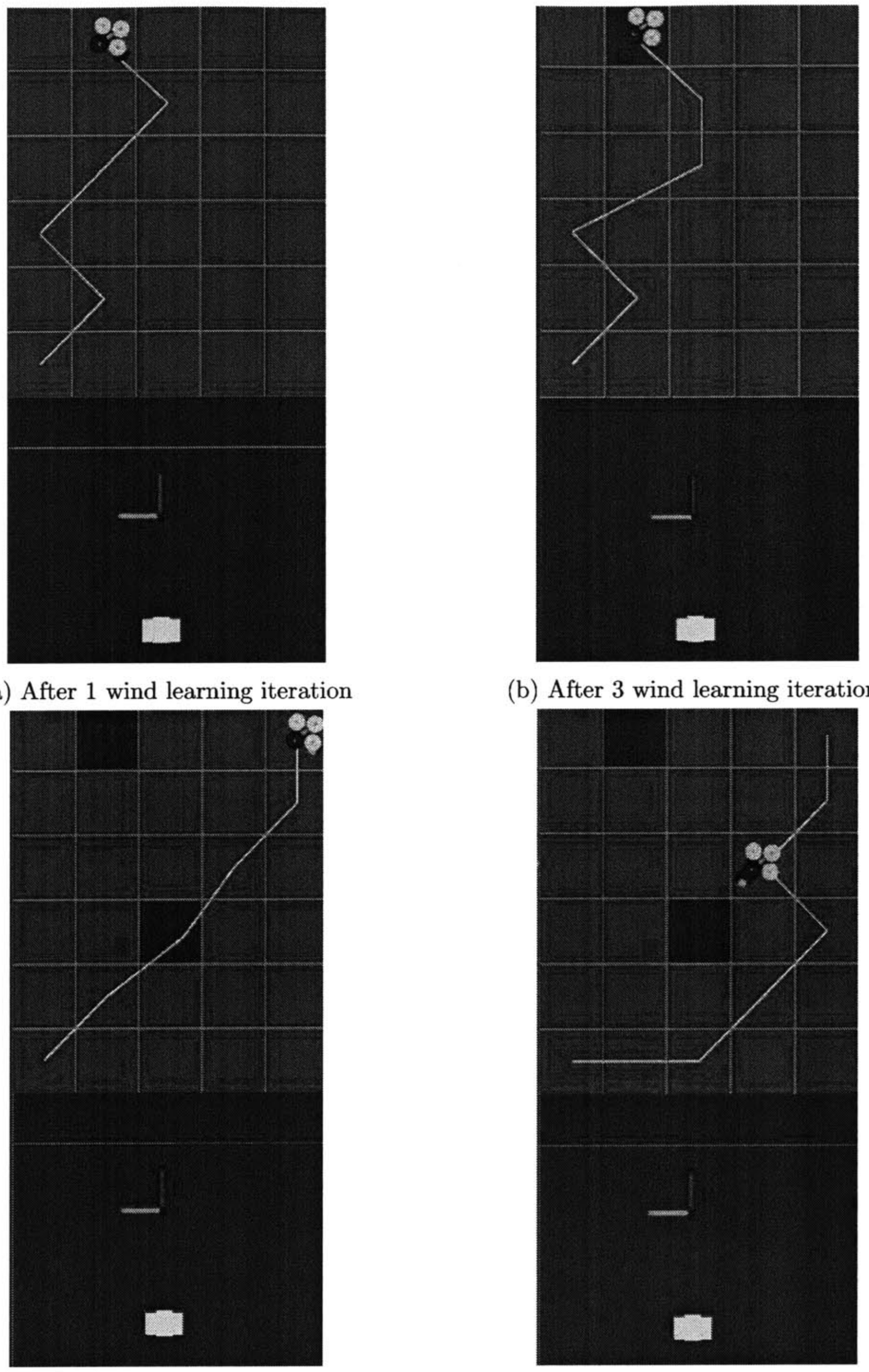


Figure 5-14: Compares the current drawn from the battery while flying under the fan with current drawn while flying away from it.

5.4 CF-Dec-iFDD Learning Experiments

In this Section we use the real sensor data obtained from onboard cameras mounted on quadrotor UAVs to estimate the fire spread model on a forest covered with different types of vegetation. Note that the results in the previous sections used the exact state transition samples to estimate the model. The main objective of this Section is to show that the developed learning algorithm can build useful models using noisy measurements (in the form of images taken by the camera) of state transition samples. The CF-Dec-iFDD algorithm (Section 4.3) is utilized as the multiagent learning algorithm in the experiments and the RCD algorithm (Section 3.3) is utilized as the multiagent planning algorithm.

The forest fire spread model is projected on the ground as seen in Figure 5-18, using the ceiling mounted projectors (Section 5.1.4). The layout of the forest is static and different types of vegetation, trees and rocks are represented with different textures. The textures of


(a) After 1 wind learning iteration

(b) After 3 wind learning iterations

(c) After 7 wind learning iterations

(d) After 10 wind learning iterations

Figure 5-15: Selected trajectories of the real agent during the wind learning process. Purple grids, which are learned by the agent, correspond to grids with non-zero probability of experiencing wind.

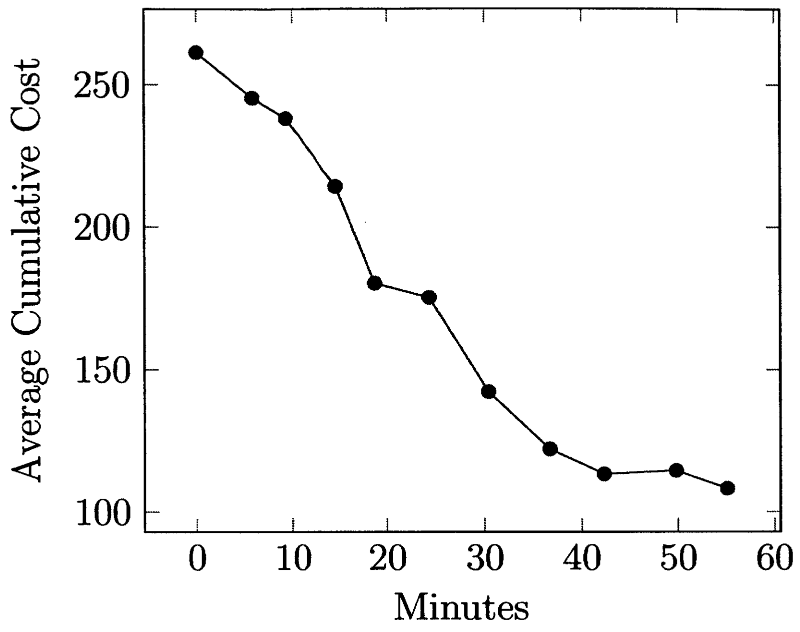


Figure 5-16: Average cost of the mission as the learning and replanning progresses for the large-scale flight experiment

combustible regions (trees, bushes etc.) are mixture of different shades of green and the color of non-combustible regions (rocks) are mainly in brown. The dynamic fire spread animation is projected upon the static image of the forest. The brightness of the fire in each region is varied according to the amount of fuel left in that particular region. The brightness of the fire at each region is propagated using the model provided in the Appendix A.

The map used for the experiments is discretized on a 12×30 grid, which is displayed in Fig. 5-17. At each step of the experiment, the quadrotor flies autonomously following a sweeping pattern across the room to collect images of the forest fire. Fig. 5-19 shows a sample image collected by the onboard camera. The collected images are then transmitted to the image processing computer. There images are stitched together to recover the full picture of the forest, which is then converted to a multidimensional array with hue-saturation-value values. Next, a simple classification algorithm is ran on these values to estimate a-) whether the region of interest is on fire or not, b-) if the regions is on fire, what is the amount of fuel left in that region.

The quadrotor was flown through 3 different fire spread patterns and collected 140 image samples in total. These 3 different runs were treated as samples obtained by different agents and CF-Dec-iFDD algorithm (Alg. 6) was used to generate model of the fire spread.

Table 5.2 provides the comparison of the results obtained form simulations and hardware

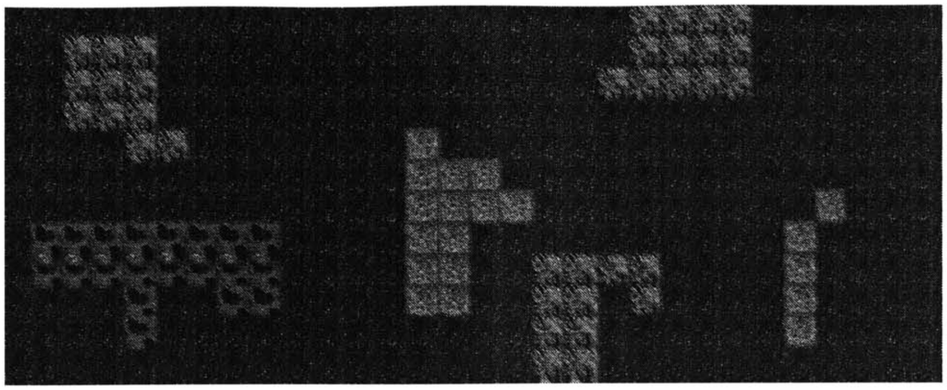


Figure 5-17: The background image representing the forest terrain. Brown regions are non-combustible rocks, dark green is the regular grass, light green is the Borel-spruce type vegetation and the mid-dark green on the lower left is the matted grass.

Table 5.2: Comparison of hardware and simulation results. Lower prediction error corresponds to making more accurate predictions regarding the direction of the fire spread.

	With Noisy State Transitions From Onboard Camera	Using Exact State Transitions
Model Error	32.2%	17.8%
Prediction Error	16.2%	10.9%

tests. The simulation results uses the exact state transitions (i.e. the exact amount of fuel on each cell) whereas the samples obtained from the experiments are prone to noise from the camera and classification errors that come from image processing. The first row of the table 5.2 compares the average model error and we see that using exact state transitions yield a significantly more accurate model as expected. However, the gap between the compared approaches is much smaller in the prediction error, which is displayed on the second row. Here the prediction error refers to the maximum likelihood estimate of which cell is going to catch fire in the next time step. As explained in the introduction, making such predictions is the main motivation for performing model learning. Hence we conclude that the developed multiagent algorithm is able to build models that yield accurate prediction of critical events (such as fire spread) using noisy state transition measurements.

After the model learning process, the learned model at each step was used by the multiagent planning algorithm RCD (Chapter 3) in a 6 quadrotor experiment (3 real and 3 simulated) and the planning performance was evaluated by computing the average cumulative cost, which corresponds to the weighted cumulative number of burning cells per evaluation.

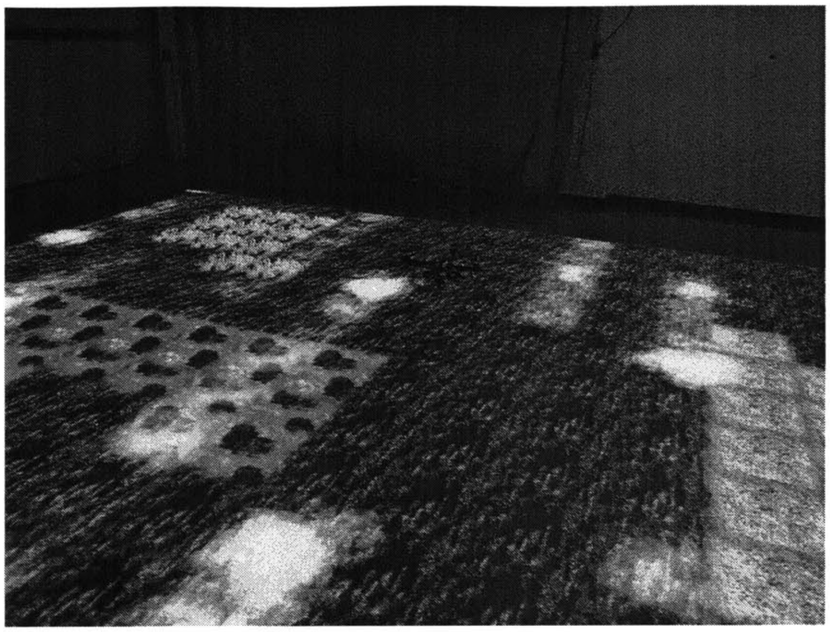


Figure 5-18: RAVEN testbed with projected fire animation on the ground and hovering quadrotor with onboard camera. Different shades of green correspond to different type of vegetation, brown textures correspond to rocky regions and the brightness of the fire represents the amount of fuel left in each cell. The fire animation is propogated by using the fire spread model provided in the Appendix.

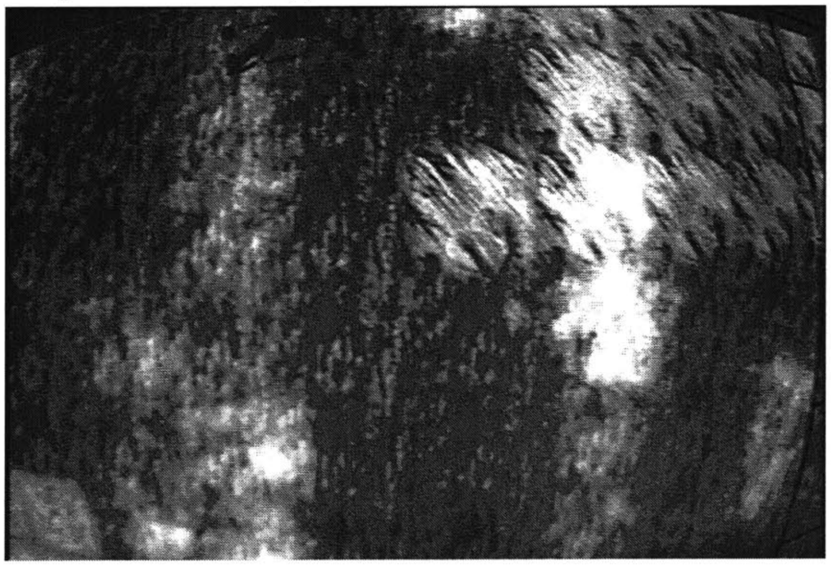


Figure 5-19: Zoomed image of the fire viewed from the onboard camera mounted on quadrotor. The images are transmitted through a wireless connection for image processing. State of the forest fire (i.e. whether the cell is burning or not and the amount of fuel left in each cell) is recovered from these images.

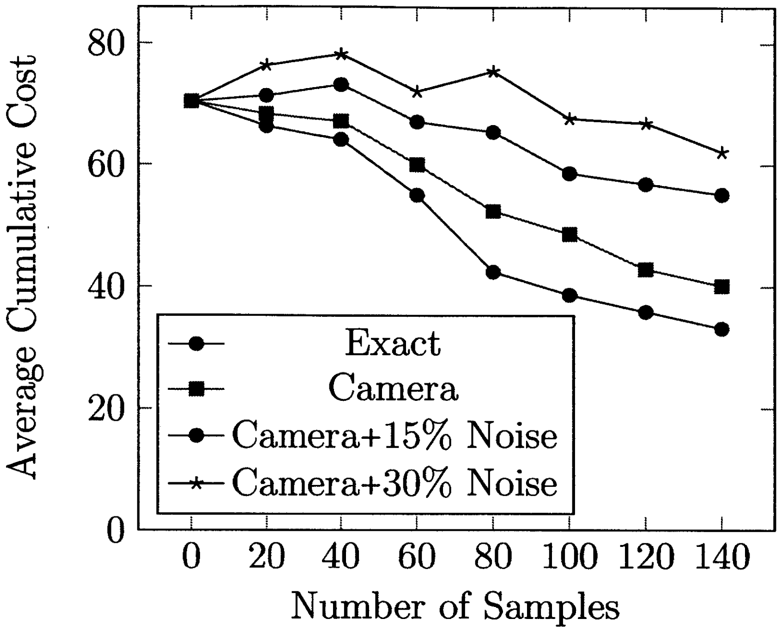


Figure 5-20: Average cost of the fire fighting mission as the learning (CF-Dec-iFDD) and replanning (RCD) progresses. The model was learned uing 4 different schemes with varying noise level. Lower cost corresponds to less number of burning cells.

The results are displayed in Fig. 5-20. Results show that as the number of samples increases the planning performance gets better, since the planning algorithm is able to make more accurate predictions regarding the fire spread. In order to assess the impact of the sensor noise on the planning performance, the planner/learner that uses the noisy camera images were compared against 3 different planners that use different models with varying levels of noise. In the Fig. 5-20, "Exact" refers to the model built from actual state transition samples. "Camera" refers to the model built from noise state transitions obtained by the camera. "Camera+15% Noise" refers to the model built from processing noisy images obtained by adding Gaussian noise with the mean of 15% of HSV values of the original image obtained from the on-board camera. "Camera+30% Noise" refers to the model built from processing noisy images obtained by adding Gaussian noise with the mean of 30% of HSV values of the original image obtained from the on-board camera. The results show that as the intensity of the noise level on the state transition samples increases, the planning performance degrades significantly due to the decreased model quality.

Finally, a 12 quadrotor (3 real, 9 simulated) hardware experiment was conducted for assessing the value of using a model under different fire spread conditions. Two different team configurations were considered. The first configuration, called the "greedy" team, does not utilize any model learning and implements a simple greedy planner to kill the fire. At each step, quadrotors move to the fire cell closest to them and kill the fire at that cell. On the other hand, the other team, called the "planning/learning" team, employs 4 quadrotors (1 in the real team and 3 in the virtual team) with on-board cameras for processing the images that are necessary to learn the fire spread model. These quadrotors are incapable of killing the fire, their only ability is to collect images required for model building. The model learning is achieved by running the CF-Dec-iFDD algorithm on the 4 learning quadrotors. Next, the RCD algorithm is used for planning based on the learned model. Note that the planning/learning team has only 8 quadrotors for killing the fire (2 real, 6 virtual), as opposed to the greedy team which utilizes all 12 quadrotors for firefighting. Two different fire conditions were considered, a scenario with low wind velocity (10 mph), and another scenario with high wind velocity (50 mph). As indicated in Appendix A, high wind velocity increases the probability of fire spreading to the neighboring cells.

Results for the low wind velocity fire configuration is given in Fig. 5-21. It is seen that in this scenario learning the model isn't very useful, the greedy team performs significantly better than the planning/learning team. After 50 samples, the greedy team accumulated $\approx 25\%$ less cost than the planning/learning team. Due to the low wind speed the fire also spreads very slowly, hence having a quadrotor to perform firefighting is more advantageous than allocating it to the model learning. Even tough the planning/learning team possess a better model of the environment dynamics, the greedy team performs better because it is superior in the number of firefighting quadrotors.

Results for the high wind velocity fire configuration is given in Fig. 5-22. It is seen the planning/learning team accumulated $\approx 24\%$ less cost than the greedy planner, despite having less number of firefighting quadrotors. Due to the high wind velocity the fire spreads more rapidly, hence it is useful to learn to model so that the direction of the fire spread can be predicted with high accuracy. This is the main reason why the planning/learning team performs better in the long term.

This result can also be observed by visual inspection of the experiment. Fig. 5-23 shows two screenshots of the experiment of the high wind velocity scenario for the greedy team, which correspond to time steps $k = 35$ and $k = 40$. Quadrotors ignore the fire cell at the top of the map and move to the burning cells that are closest to them. Due to the high wind velocity, the fire at the top of the map spreads to the highly combustible Borel-Spruce type vegetation and then the fire spreads out quickly, as displayed on the bottom image in Fig. 5-23. On the other hand, Fig. 5-24 shows two screenshots of the experiment of the high wind velocity scenario for the planning/learning team for the same time steps. Since the planning/learning team possesses the model of the fire spread dynamics, the planner predicts that the fire at the top of the map is likely to spread quickly to the Borel-Spruce vegetation, and immediately sends quadrotors to kill the fire at that point. As the bottom image of the Fig. 5-16 shows, the fire is prevented from spreading to Borel-Spruce vegetation. Hence, it can be said that having a model for the fire spread prediction could provide very useful for scenarios where the fire grows quickly due to the high wind velocity.

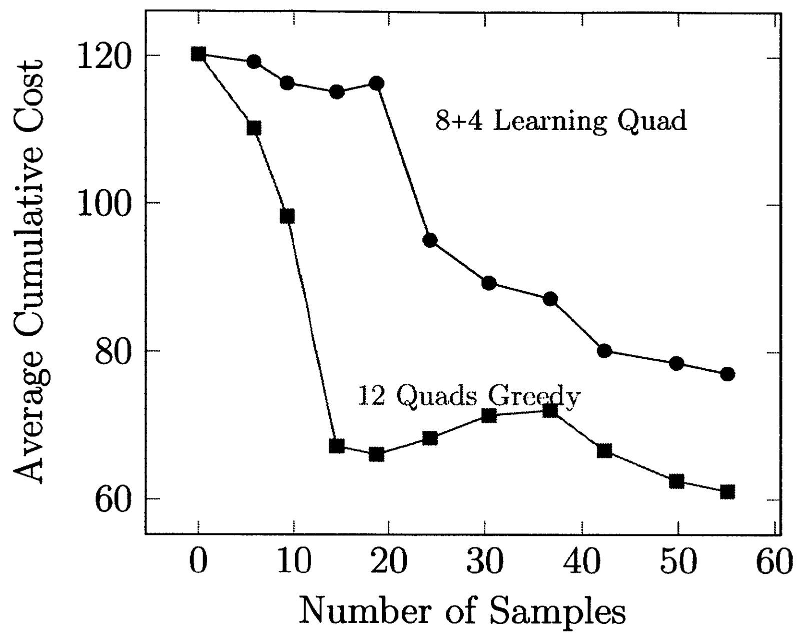


Figure 5-21: Comparison of average cost of the fire fighting mission versus number of samples for the greedy and the planning/learning teams. The wind velocity is 10mph. Lower cost corresponds to less number of burning cells.

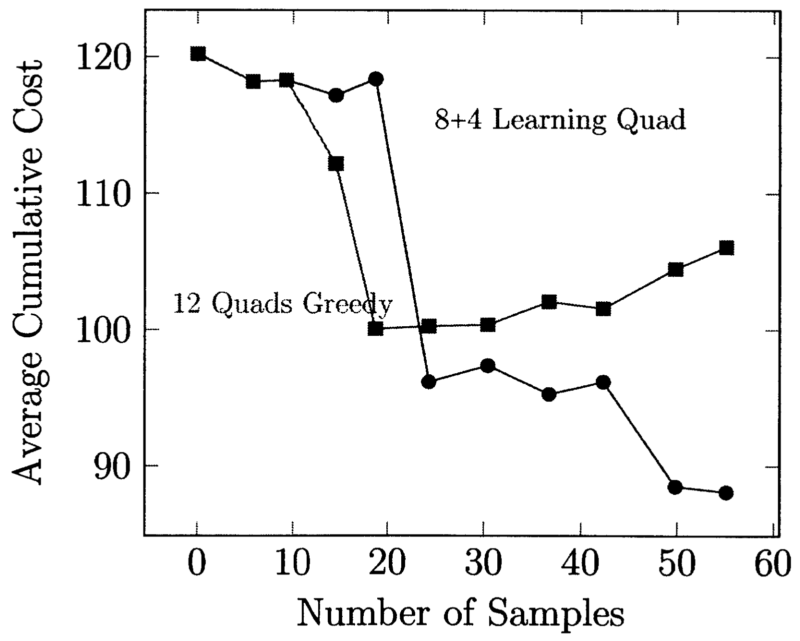
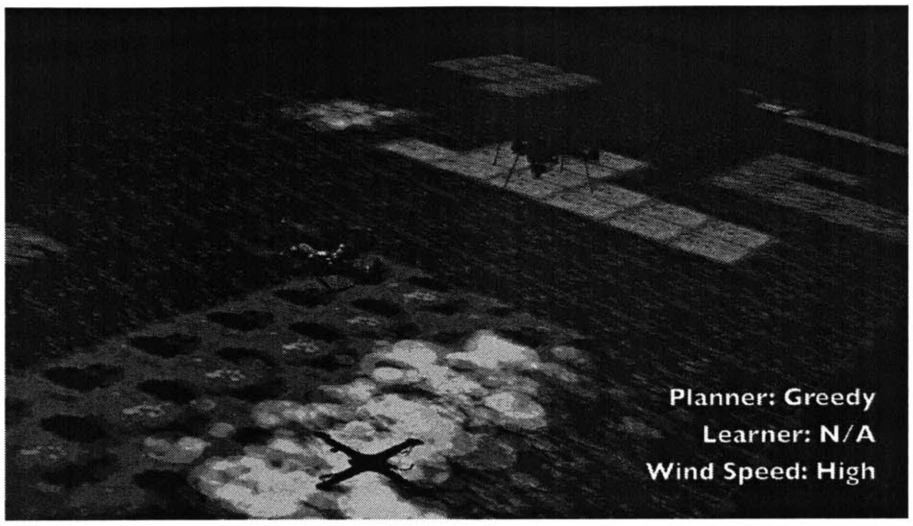


Figure 5-22: Comparison of average cost of the fire fighting mission versus number of samples for the greedy and the planning/learning teams. The wind velocity is 50mph. Lower cost corresponds to less number of burning cells.



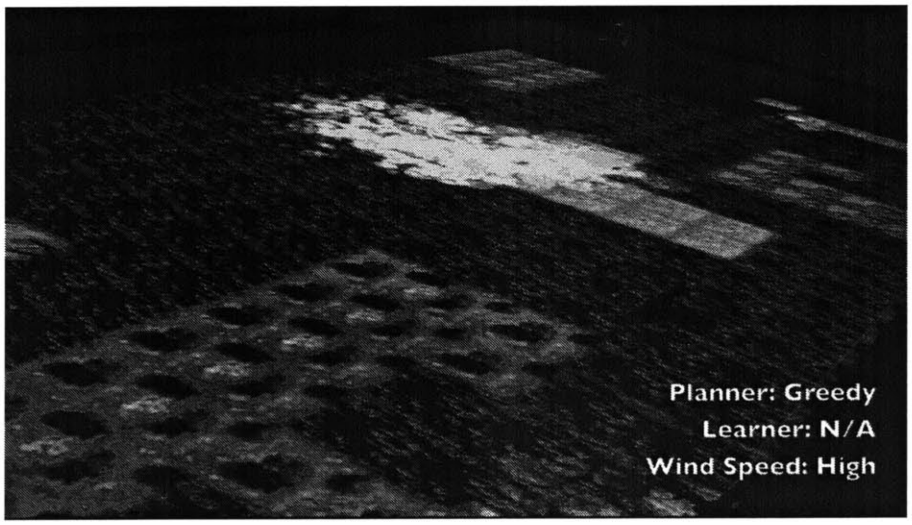
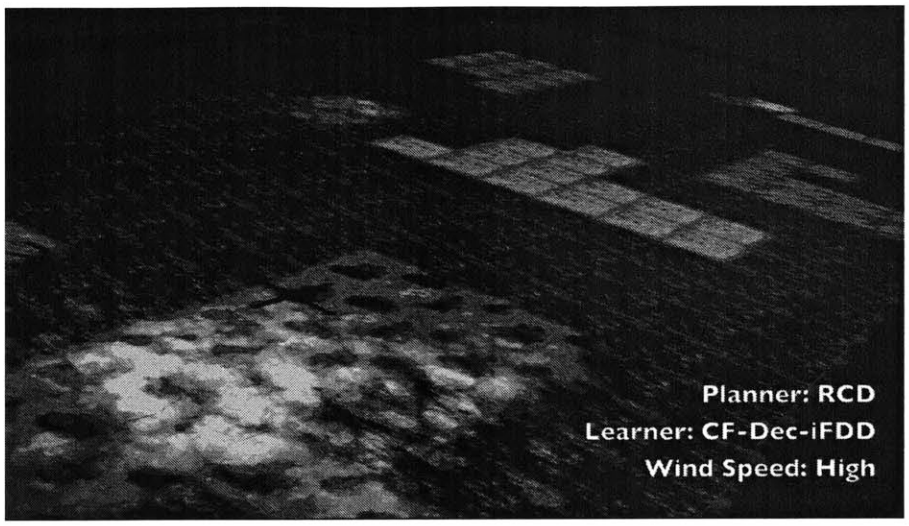


Figure 5-23: Snapshots from the hardware experiment corresponding to the fire configuration with high wind velocity for the greedy team.



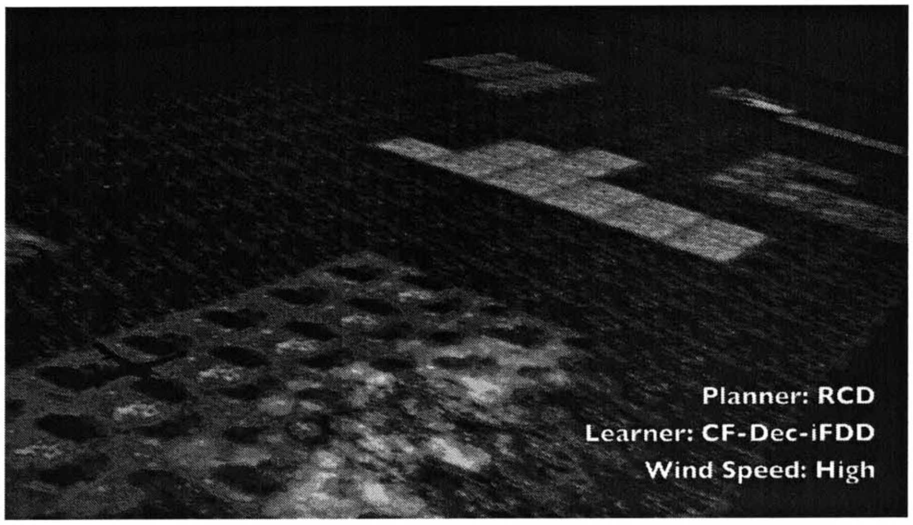


Figure 5-24: Snapshots from the hardware experiment corresponding to the fire configuration with high wind velocity for the planning/learning team.

5.5 Summary

This Chapter provided the details of the experimental results for the multiagent learning algorithms developed in the Chapter 4. The results demonstrated the applicability of these algorithms in the presence of real flight hardware, long duration missions, sensor noise and wind disturbance. In particular, the developed multiagent learning algorithm CF-Dec-iFDD and the multiagent planning algorithm RCD was verified on a large scale firefighting mission that involves ceiling mounted projectors and on-board cameras.

Chapter 6

Conclusions and Future Work

This Chapter provides an overall summary of the work done in this Thesis, its contributions and some suggestions for the future work.

6.1 Conclusions

The main focus of this thesis was to address three particular challenges in the field of planning/learning in multiagent systems. Chapter 1 laid out these challenges along with the problem formulations. The first challenge was the scalability, which is concerned with the issue of algorithms becoming computationally intractable as the number of agents in the problem increases. The second challenge was the heterogeneity, which is concerned with the issue that transferring learned transition models across the agents becomes difficult when the agents have dissimilar transition dynamics. The final challenge was the lack of domain knowledge, which is concerned with the issue that existing algorithms rely on domain expertise to handle parameter tuning for complex applications, whereas such knowledge may not exist in many problems.

The Chapter 3 focused on the development of the RCD algorithm. The main premise of the algorithm was being scalable to large-scale scenarios, while not being strongly dependent on the domain knowledge. This property was achieved through replacement of the manual tuning with an embedded randomized search process. This Chapter presented the development of the algorithm, showed that the algorithms iteration complexity was linear in the number of agents, proved that the algorithm is asymptotically convergent and compared the algorithms planning performance with alternative approaches across multiple multiagent planing domains. Results showed that RCD yields multiagent policies with significantly lower costs.

The Chapter 4 focused on the development of the iFDD, Dec-iFDD and CF-Dec-iFDD learning algorithms. iFDD learning algorithm was developed for single agent model learning applications. The main premise of the iFDD learning is not fixing the model structure beforehand and allowing the model complexity grow based on the observed estimation error. The Chapter presented the algorithm development, proved the asymptotic convergence of iFDD and provided comparative simulation results showing the superiority of the algorithm over alternative approaches. The Dec-iFDD and CF-Dec-iFDD were developed as the multiagent extensions of the iFDD learning algorithm. Both algorithms operate by allowing agents to exchange learned features under limited communication. The simulation results and theoretical analysis highlighted the scenarios where these algorithms are superior to learning independently or randomly sharing features across agents. In particular, it was shown that CF-Dec-iFDD enables faster learning in heterogeneous teams.

Finally, the Chapter 5 presented the hardware implementations of the iFDD, Dec-iFDD and CF-Dec-iFDD algorithms in hardware flight tests. The results demonstrated that the algorithms were able to work in scenarios that involve, real autonomous vehicles, real-time computation constraints, external disturbances and sensor noise. A number of novel experimental devices were also developed to accommodate the experiments, which involves a recharge station for enabling fully autonomous long endurance missions, ceiling mounted fans for emulating wind disturbances and ceiling mounted projectors for animating environmental dynamics and projecting the planner's belief space onto the testbed for visual monitoring.

6.2 Future Work

The following future work is suggested.

6.2.1 Extension to State-Dependent Coordination Graphs

The Coordination Factor Graphs introduced in the Section 3.2 assumes that the structure is fixed throughout the state-space. This might be a limiting assumption in some multiagent planning scenarios. For instance, in a multiagent pursuit evasion scenario, coupling/decoupling agents based on their proximity to the targets might result in better planning performance. Hence instead of assuming a single graph structure, utilizing a family of coordination graphs that switches based on the current state might be more efficient. Such graphical structures were studied previously in [122, 123]. However, these works assume that the state-dependent coordination graphs are hard-coded beforehand. The RCD algorithm can be extended to discover such state-dependent coordination graphs. The main challenge is developing an efficient parametrization of these graphs, and defining the probability distributions for sampling from the space of these graphs.

6.2.2 Analysis of Convergence Rate of RCD

The theoretical results provided in the Section 3.4 proved the asymptotic convergence of the RCD algorithm. However, these results do not provide any bounds/guarantees on the finite sample performance, which is of great practical interest. Methods such as PAC-analysis [124] can be used to investigate the convergence rate of the RCD-TBPI algorithm. The main challenge is analyzing the stochastic dynamics induced by the Bayesian Optimization Algorithm.

6.2.3 Extension to Decentralized Partially Observable MDPs

This Thesis focused on centralized multiagent planning problems and assumed that the full state information was available. However, there exists applications where the full state information may not be available and a centralized planning architecture might not be feasible due to communication constraints. Such scenarios can be modelled as Decentralized Partially Observable MDPs (Dec-POMDPs) [65, 66]. Although the development of planning algorithms for Dec-POMDPs is significantly more difficult, there has been a significant recent effort in this area with promising results [23]. Automating the coordination structure is also a relevant problem for Dec-POMDPs [125, 126, 127], however the existing methods have been focusing on greedy search algorithms so far. The coordination search for Dec-POMPDs can benefit from a randomized search algorithm such as RCD. However the extension is non-trivial, since RCD might need many planning iterations to converge, whereas the most Dec-POMDP solvers are computationally expensive.

6.2.4 Non-binary Features for iFDD

iFDD uses a linear combination of binary features to reduce computational complexity. However learning the health dynamics of agents may demand models with more powerful representational features for scenarios with stronger couplings between the vehicle health and the mission (e.g., with more complex environments and task definitions). Also, in the context of control of manufacturing and maintenance process, modeling aging and breakdown dynamics usually demand complicated continuous models. Future work might involve investigating new model representations for decentralized learning, such as parametric and non-parametric Bayesian representations [19] and generalization of iFDD from binary features to continuous representations, such as radial basis functions. The issues here will be to retain the sparsity (inherent to iFDD) in the representations and mechanics of growing the representation based on the observed data to reduce computational complexity.

6.2.5 Analysis of Convergence Rate of iFDD, Dec-iFDD and CF-Dec-iFDD

The Section 4.1 proved the asymptotic convergence of the iFDD algorithm. The work in [99] gave results on the convergence rate of the iFDD in the context of value function approximation. The same guarantees can be extended to the transition model learning framework of the Section 4.1. A particularly interesting future work is extending the same convergence results to the Dec-iFDD and CF-Dec-iFDD algorithms. This is a challenging task, since the analysis should take into consideration convergence of multiple models and dynamics of the feature sharing process.

6.2.6 Offline Pattern Classification

The algorithms developed in Chapter 4 are designed to handle single run scenarios. These methods can be extended to transfer learned models across different missions. A library of learned models can be built using probabilistic clustering algorithms [128, 129], and in the new scenario the learned models can be checked if they are a member of existing clusters to accelerate the overall learning process.

6.2.7 More Complex Experiments with Emulated Communication Networks

Establishing a reliable information flow between the agents is a complex task in real missions with terrain and lossy communication networks, since many of the packets sent are lost or corrupted. This has become a dominant issue in DoD research, especially with the focus on contested environments in which external factors (intentional or not) limit the ability of a UAV team to communicate between themselves and the base. In order to create the same effect in an indoor environment, a network emulator can be embedded into the testbed to induce communication and data loss on the experiment. These emulators can demonstrate the robustness of the developed algorithms in face of lossy communication networks. Since the probability of achieving a successful communication link strongly depends on the separation between the agents and the location in the environment [130], these models can also serve as an opportunity to demonstrate the state-dependent uncertainty learning in our work [32].

Appendix A

Stochastic Forest Fire Model

The model was obtained from Boychuk [98]. Let the environment be discretized into an $n \times n$ grid. Each cell is labeled with their coordinates $(i,j), i,j = 1,..,n$. The state $s_{(i,j)}$ represents if the cell is burning or not and the amount of fuel contained within the cell. $s_{(i,j)}$ takes values from the set $\{0,\dots,f^{\max}_{(i,j)}, NB\}$, where NB denotes that the cell is currently not burning, $f^{\max}_{(i,j)} \in \mathbb{Z}$ is the maximum amount of fuel the cell can hold and $s_{(i,j)} = x, x \in \{0,...,f^{\max}_{(i,j)}\}$ denotes that the cell is burning and the remaining amount of fuel is x.

Setting different $f^{\max}_{(i,j)}$ among the cells corresponds to having a heterogeneous terrain and vegetation in the environment. For instance, for a cell full of vegetation $f^{\max}_{(i,j)}$ can be set to 10, whereas for a non-combustible cell (such as a cell full of rocks), $f^{\max}_{(i,j)}$ can be set to 0. Let k represent the discrete time and $s_{(i,j)}(k)$ denote the state of the cell at time k.

The cell (i,j)'s neighborhood $\mathcal{N}_{(i,j)}$ is the set of cells that surrounds it,

$$\mathcal{N}_{(i,j)} = \{(i-1,j),(i+1,j),(i,j-1),(i,j+1)\}. \tag{A.1}$$

The probability that the cell (i,j) will transition from NB state to burning state depends on three different factors; the state of its neighboring cells $\mathcal{N}_{(i,j)}$, the wind w and the vegetation type of the cell $\nu_{(i,j)}$. This dependence is given as,

$$Pr\left(s_{(i,j)}(k+1) = f^{\max}_{(i,j)} | s_{(i,j)}(k) = NB\right) = \sum_{(k,l)\in\mathcal{N}_{(i,j)}} f(s_{(k,l)}, w, \nu_{(i,j)}), \tag{A.2}$$

where $f(s_{(k,l)}, w, v_{(i,j)}) = 0$ if $s_{(k,l)} = NB$ or $s_{(k,l)} = 0$. Otherwise,

$$f(s_{(k,l)}, w, v_{(i,j)}) = \frac{\nu^b_{(i,j)}}{1 - \cos(\theta_{(k,l)} - \theta_m)\left(1 - \frac{\nu^b_{(i,j)}}{\nu^b_{(i,j)}}\right)}, \tag{A.3}$$

where $\nu^b_{(i,j)}$ and $\nu^a_{(i,j)}$ are dimensionless parameters that depend on the vegetation type of cell (i,j) and the maximum speed of the wind w, θ_m is the direction of the wind and $\theta_{(k,l)}$ is the angle between the cell (i,j) and (k,l),

$$\theta_{(k,l)} = \begin{cases} \pi & (if)(k,l) = (i,j-1) \\ 0.5\pi & (if)(k,l) = (i+1,j) \\ 1.5\pi & (if)(k,l) = (i-1,j) \\ 0 & (if)(k,l) = (i,j+1) \end{cases}. \tag{A.4}$$

When the cell starts burning, the fuel decreases in a deterministic manner with each time step,

$$Pr\left(s_{(i,j)}(k+1) = x - 1 | s_{(i,j)}(k) = x\right) = 1, \quad x \neq 0, \tag{A.5}$$

and when the amount of fuel in the cell reaches zero, the cell is burnt out and will stay in this state for all future times,

$$Pr\left(s_{(i,j)}(k+1) = 0 | s_{(i,j)}(k) = 0\right) = 1. \tag{A.6}$$

Examining Eqs. A.2 and A.3, shows that the probability the fire will spread to the cell (i,j) strongly depends on the state of its neighbors. If all the neighbors are in NB state or burnt-out, then the cell will not be affected by fire. Eq. A.3 shows that if the cell (i,j) has a burning neighbor in the direction of the wind flow, the probability that the fire will spread to the cell (i,j) increases. Also as seen in Eq. A.2 the probability of fire spread increases linearly with the number of burning neighboring cells. Once the fire spreads to the cell, the transition dynamics become deterministic and cell burns until it runs out of fuel. For the experiments in the Section 3.5 and Section 5.4, the parameters were set as follows:

1. For Borel Spruce type cells, $\nu^b = 30, \nu^m = 66.5, f^{\max} = 10$,
2. For matted grass type cells, $\nu^b = 1.5, \nu^m = 8.5, f^{\max} = 7$.
3. For regular grass type cells, $\nu^b = 1.0, \nu^m = 8.5, f^{\max} = 3$.
4. The wind direction is $\theta_m = \pi/6$ [98], and the wind speed was set as 50 mph.

Bibliography

[1] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[2] Nicholas R Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial intelligence*, 75(2):195–240, 1995.

[3] Claire Tomlin, George J Pappas, and Shankar Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):509–521, 1998.

[4] Jayashankar M Swaminathan, Stephen F Smith, and Norman M Sadeh. Modeling supply chain dynamics: A multiagent approach*. *Decision sciences*, 29(3):607–632, 1998.

[5] Julian De Hoog, Stephen Cameron, and Arnoud Visser. Role-based autonomous multi-robot exploration. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD'09. Computation World:*, pages 482–487. IEEE, 2009.

[6] Andreas Kolling and Stefano Carpin. Pursuit-evasion on trees by robot teams. *Robotics, IEEE Transactions on*, 26(1):32–47, 2010.

[7] N. K. Ure, G. Chowdhary, J. Redding, T. Toksoz, J. P. How, M. Vavrina, and J. Vian. Experimental demonstration of efficient multi-agent learning and planning for persistent missions in uncertain environments. In *Conference on Guidance Navigation and Control*, Minneapolis, MN, August 2012. AIAA.

[8] Hongru Tang, Xiaosong Cao, Aiguo Song, Yan Guo, and Jiatong Bao. Human-robot collaborative teleoperation system for semi-autonomous reconnaissance robot. In *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pages 1934–1939. IEEE, 2009.

[9] Lorena A Bearzotti, Enrique Salomone, and Omar J Chiotti. An autonomous multi-agent approach to supply chain event management. *International Journal of Production Economics*, 135(1):468–478, 2012.

[10] Dave McKenney and Tony White. Distributed and adaptive traffic signal control within a realistic traffic simulation. *Engineering Applications of Artificial Intelligence*, 26(1):574–583, 2013.

[11] Tadashi Koga and Xiaodong Lu. Autonomous decentralized surveillance system and continuous target tracking technology for air traffic control applications. In *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, pages 1–8. IEEE, 2013.

[12] Takaya Miyazawa, Hideaki Furukawa, Kenji Fujikawa, Naoya Wada, and Hiroaki Harai. Development of an autonomous distributed control system for optical packet and circuit integrated networks. *Optical Communications and Networking, IEEE/OSA Journal of*, 4(1):25–37, 2012.

[13] Alex Rogers, Sarvapali D Ramchurn, and Nicholas R Jennings. Delivering the smart grid: Challenges for autonomous agents and multi-agent systems research. In *AAAI*, 2012.

[14] Gilson Yukio Sato, Hilton José Azevedo, and Jean-Paul A Barthès. Agent and multi-agent applications to support distributed communities of practice: a short review. *Autonomous Agents and Multi-Agent Systems*, 25(1):87–129, 2012.

[15] Paulo Leitão, José Barbosa, and Damien Trentesaux. Bio-inspired multi-agent systems for reconfigurable manufacturing systems. *Engineering Applications of Artificial Intelligence*, 25(5):934–944, 2012.

[16] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Machine Learning and Knowledge Discovery in Databases*, pages 656–671. Springer, 2008.

[17] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[18] David H Wolpert, Kagan Tumer, and Jeremy Frank. Using collective intelligence to route internet traffic. *arXiv preprint cs/9905004*, 1999.

[19] Trevor Campbell, Sameera S. Ponda, Girish Chowdhary, and Jonathan P. How. Planning under uncertainty using nonparametric bayesian models. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2012.

[20] Luca F. Bertuccelli. *Robust Decision-Making with Model Uncertainty in Aerospace Systems*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, September 2008.

[21] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[22] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, 1999.

[23] C. Amato, G. Chowdhary, A. Geramifard, and N. K. Ure. Decentralized Control of Partially Observable Markov Decision Processes. In *The 52nd IEEE Conference on Decision and Control (CDC)*, 2013.

[24] Andrew N. Kopeikin. Dynamic Mission Planning for Communication Control in Multiple Unmanned Aircraft Teams. Master's thesis, Massachusetts Institute of Technology, June 2012.

[25] Michael Wooldridge. *An introduction to multiagent systems.* Wiley, 2002.

[26] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 1st edition, 2007.

[27] W.B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality.* Wiley-Interscience, 2007. pp. 225–262.

[28] R. Sutton and A. Barto. *Reinforcement Learning, an Introduction.* MIT Press, Cambridge, MA, 1998.

[29] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press, 2010.

[30] Sridhar Mahadevan, Mauro Maggioni, and Carlos Guestrin. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research (JMLR)*, 8:2007, 2006.

[31] P.M. Djuric and Yunlong Wang. Distributed Bayesian learning in multiagent systems: Improving our understanding of its capabilities and limitations. *Signal Processing Magazine, IEEE*, 29(2):65 –76, march 2012.

[32] Nazim Kemal Ure, Girish Chowdhary, Yu Fan Chen, Jonathan P. How, and John Vian. Health-aware decentralized planning and learning for large-scale multiagent missions. In *Conference on Guidance Navigation and Control*, Washington DC, August 2013. AIAA.

[33] Justin A Boyan and Michael L Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pages 671–671, 1994.

[34] M. Bowling and M. Veloso. Scalable learning in stochastic games, 2002.

[35] Alborz Geramifard. *Practical Reinforcement Learning Using Representation Learning and Safe Exploration for Large Scale Markov Decision Processes.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, February 2012.

[36] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958.

[37] R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60:503–515, 1954.

[38] Richard Ernest Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[39] D. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2005.

[40] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[41] Ronen I. Brafman and Moshe Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 3:213–231, 2002.

[42] J. N. Tsitsiklis and B. V. Roy. An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.

[43] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[44] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.

[45] Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In Lise Getoor and Tobias Scheffer, editors, *International Conference on Machine Learning (ICML)*, pages 881–888. ACM, June 2011.

[46] C. Painter-Wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. *arXiv preprint arXiv:1206.6485*, 2012.

[47] Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2):161–178, 2002.

[48] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence*, pages 106–111. JOHN WILEY & SONS LTD, 1997.

[49] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *J. Artif. Intell. Res. (JAIR)*, 19:399–468, 2003.

[50] Kevin Patrick Murphy. *Dynamic Bayesian networks: representation, inference and learning.* PhD thesis, University of California, 2002.

[51] Daphne Koller and Ronald Parr. Policy iteration for factored mdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 326–334. Morgan Kaufmann Publishers Inc., 2000.

[52] Daniela Pucci de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[53] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, 2002.

[54] Alexander L Strehl, Carlos Diuk, and Michael L Littman. Efficient structure learning in factored-state mdps. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 645. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[55] Thomas Degris, Olivier Sigaud, and Pierre-Henri Wuillemin. Chi-square tests driven method for learning the structure of factored mdps. *arXiv preprint arXiv:1206.6842*, 2012.

[56] Karina Valdivia Delgado, Scott Sanner, and Leliane Nunes De Barros. Efficient solutions to factored mdps with imprecise transition probabilities. *Artificial Intelligence*, 175(9):1498–1527, 2011.

[57] Finale Doshi-Velez, David Wingate, Joshua Tenenbaum, and Nicholas Roy. Infinite dynamic Bayesian networks. 2011.

[58] Jelle R Kok, Pieter Jan't Hoen, Bram Bakker, and Nikos A Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *CIG*, 2005.

[59] Thomas G Dietterich et al. The maxq method for hierarchical reinforcement learning. In *Proceedings of the fifteenth international conference on machine learning*, volume 8. Citeseer, 1998.

[60] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the fourteenth conference on Uncertainty in Artificial Intelligence*, 1998.

[61] Richard S Sutton, Doina Precup, Satinder Singh, et al. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[62] Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *Proceedings of The 31st International Conference on Machine Learning*, pages 127–135, 2014.

[63] Marc Toussaint, Laurent Charlin, and Pascal Poupart. Hierarchical pomdp controller optimization by likelihood maximization. In *UAI*, volume 24, pages 562–570, 2008.

[64] George Konidaris and Andrew G Barto. Efficient skill learning using abstraction selection. In *IJCAI*, volume 9, pages 1107–1112, 2009.

[65] Daniel S Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 32–37. Morgan Kaufmann Publishers Inc., 2000.

[66] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32(1):289–353, 2008.

[67] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[68] Douglas Gale and Shachar Kariv. Bayesian learning in social networks. *Games and Economic Behavior*, 45(2):329–346, 2003.

[69] Cameron S.R. Fraser, Luca F. Bertuccelli, Han-Lim Choi, and Jonathan P. How. A hyperparameter consensus method for agreement under uncertainty. *Automatica*, 48(2):374–380, February 2012.

[70] Jörgen W Weibull. *Evolutionary game theory*. MIT press, 1997.

[71] Vikram Krishnamurthy. Quickest time herding and detection for optimal social learning. *arXiv preprint arXiv:1003.4972*, 2010.

[72] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, page 163, 1994.

[73] José M Vidal and Edmund H Durfee. Agents learning about agents: A framework and analysis. In *Working Notes of the AAAI-97 workshop on Multiagent Learning*, volume 7176, 1997.

[74] Mitchell A Potter. *The design and analysis of a computational model of cooperative coevolution*. PhD thesis, Citeseer, 1997.

[75] Sean Luke et al. Genetic programming produced competitive soccer softbot teams for robocup97. *Genetic Programming*, 1998:214–222, 1998.

[76] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.

[77] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[78] A Evgeniou and Massimiliano Pontil. Multi-task feature learning. In *Advances in*

neural information processing systems: Proceedings of the 2006 conference, volume 19, page 41. The MIT Press, 2007.

[79] Weike Pan, Evan Wei Xiang, and Qiang Yang. Transfer learning in collaborative filtering with uncertain ratings. In *AAAI*, 2012.

[80] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[81] Martin Pelikan. Bayesian optimization algorithm. In *Hierarchical Bayesian Optimization Algorithm*, pages 31–48. Springer, 2005.

[82] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[83] Alborz Geramifard, Thomas J. Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, and Jonathan P. How. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends in Machine Learning*, 6(4):375–451, 2013.

[84] L. Li, M.L. Littman, and T.J. Walsh. Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th international conference on Machine learning*, pages 568–575. ACM New York, NY, USA, 2008.

[85] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

[86] HJ Kushner and DS Clark. Stochastic approximation methods for constrained and unconstrained systems (series: applied mathematical sciences). 1978.

[87] N. Kemal Ure, Alborz Geramifard, Girish Chowdhary, and Jonathan P. How. Adaptive Planning for Markov Decision Processes with Uncertain Transition Models via Incremental Feature Dependency Discovery. In *European Conference on Machine Learning (ECML)*, 2012.

[88] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4, 2009.

[89] Emmanuel J Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[90] Jelle R Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *The Journal of Machine Learning Research*, 7:1789–1828, 2006.

[91] David Barber. *Bayesian reasoning and machine learning.* Cambridge University Press, 2012.

[92] Martin J Wainwright, John D Lafferty, and Pradeep K Ravikumar. High-dimensional graphical model selection using l1-regularized logistic regression. In *Advances in neural*

information processing systems, pages 1465–1472, 2006.

[93] Qingfu Zhang and Heinz Muhlenbein. On the convergence of a class of estimation of distribution algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(2):127–136, 2004.

[94] D.P. Bertsekas and J.N. Tsitsiklis. *Introduction to probability.* Athena Scientific Belmont, Massachusetts, 2008.

[95] Malcolm Ritchie Adams and Victor Guillemin. *Measure theory and probability.* Springer, 1996.

[96] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

[97] N. Kemal Ure, Shayegan Omidshafiei, Thomas Brett Lopez, Ali akbar Aghamohammadi, Jonathan P. How, and john Vian. Heterogeneous Multiagent Learning with Applications to Forest Fire Management. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015 (Submitted).

[98] Den Boychuk, W John Braun, Reg J Kulperger, Zinovi L Krougly, and David A Stanford. A stochastic forest fire growth model. *Environmental and Ecological Statistics*, 16(2):133–151, 2009.

[99] Alborz Geramifard, Thomas J. Walsh, Nicholas Roy, and Jonathan How. Batch iFDD: A Scalable Matching Pursuit Algorithm for Solving MDPs. In *Proceedings of the 29th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Bellevue, Washington, USA, 2013. AUAI Press.

[100] Jelle R Kok, Matthijs TJ Spaan, and Nikos Vlassis. Multi-robot decision making using coordination graphs. In *Proceedings of the 11th International Conference on Advanced Robotics, ICAR*, volume 3, pages 1124–1129, 2003.

[101] B. Michini and J. P. How. Bayesian Nonparametric Inverse Reinforcement Learning. In *European Conference on Machine Learning (ECML)*, Bristol, UK, Sept. 2012.

[102] Harold J. Kushner and G. George Yin. *Convergence of indirect adaptive asynchronous value iteration algorithms.* Springer, 2003.

[103] Paul E. Utgoff and Doina Precup. *Feature extraction, construction, and selection: A data-mining perspective*, chapter Constructive function approximation. 1998.

[104] A. Shapiro and Y. Wardi. Convergence analysis of gradient descent stochastic algorithms. *Journal of Optimization Theory and Applications*, 91(2):439–454, 1996.

[105] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd Edition.* Prentice-Hall, Englewood Cliffs, NJ, 2003.

[106] B. Bethke, J. P. How, and J. Vian. Multi-UAV Persistent Surveillance With Communication Constraints and Health Management. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2009. (AIAA-2009-5654).

[107] J. D. Redding, T. Toksoz, N. Kemal Ure, A. Geramifard, J. P. How, M. Vavrina, and J. Vian. Persistent distributed multi-agent missions with automated battery management. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2011. (AIAA-2011-6480).

[108] Nazim Kemal Ure, Girish Chowdhary, Yu Fan Chen, Mark Cutler, Jonathan P. How, and John Vian. Decentralized learning based planning multiagent missions in presence of actuator failures. In *International Conference on Unmanned Aircraft Systems*, Atlanta GA, August 2013. IEEE.

[109] N. Kemal Ure, Girish Chowdhary, Yu Fan Chen, Jonathan P. How, and John Vian. Distributed learning for planning under uncertainty problems with heterogeneous teams. *Journal of Intelligent and Robotic Systems*, pages 1–16, 2013.

[110] M. Valenti, B. Bethke, G. Fiore, J. P. How, and E. Feron. Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Keystone, CO, August 2006 (AIAA-2006-6200).

[111] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, 28(2):51–64, April 2008.

[112] J. P. How, C. Fraser, K. C. Kulling, L. F. Bertuccelli, O. Toupet, L. Brunet, A. Bachrach, and N. Roy. Increasing autonomy of UAVs. *Robotics and Automation Magazine, IEEE*, 16(2):43–51, June 2009.

[113] Mark Cutler. Design and Control of an Autonomous Variable-Pitch Quadrotor Helicopter. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, August 2012.

[114] N. Michael, J. Fink, and V. Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30(1):73–86, January 2011.

[115] S. Lupashin, A. Schollig, M. Hehn, and R. D'Andrea. The flying machine arena as of 2010. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2970–2971. IEEE, 2011.

[116] G Chowdhary, D. M. Sobers, C. Pravitra, C. Christmann, A. Wu, H. Hashimoto, R. Kalghatgi, C. Ong, and E. Johnson. Self-contained autonomous indoor flight with ranging sensor navigation. *AIAA Journal of Guidance Control and Dynamics*, 35(6):1843–1854, November-December 2012.

[117] S. S. Ponda, L. B. Johnson, A. N. Kopeikin, H. Choi, and J. P. How. Distributed

planning strategies to ensure network connectivity for dynamic heterogeneous teams. *IEEE Journal on Selected Areas in Communications*, 30(5):861 –869, June 2012.

[118] T. Toksoz and N. K. Ure and G. Chowdhary and A. Geramifard and J. P. How. 3 hours persistent search and track mission with autonomous recharge station. Available at http://acl.mit.edu/projects/recharge.htm, 2012.

[119] N. K. Ure, G. Chowdhary, T. Toksoz, J. P. How, M. Vavrina, and J. Vian. Automated battery management system for enabling multi-agent persistent missions. *IEEE Transactions of Mechatronics*, 2012 (submitted).

[120] Tuna Toksoz. Design and Implementation of an Automated Battery Management Platform. Master's thesis, Massachusetts Institute of Technology, August 2012.

[121] H.-L. Choi, L. Brunet, and J. P. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, August 2009.

[122] Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-specific multiagent coordination and planning with factored mdps. In *AAAI/IAAI*, pages 253–259, 2002.

[123] Jelle R Kok and Nikos Vlassis. Sparse cooperative q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 61. ACM, 2004.

[124] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research (JMLR)*, 10:2413–2444, 2009.

[125] M.T.J. Spaan, F.A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in dec-pomdps via incremental expansion. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2027–2032. AAAI Press, 2011.

[126] Pradeep Varakantham, Jun-young Kwak, Matthew E Taylor, Janusz Marecki, Paul Scerri, and Milind Tambe. Exploiting coordination locales in distributed pomdps via social model shaping. In *ICAPS*, 2009.

[127] Stefan J Witwicki and Edmund H Durfee. Towards a unifying characterization for quantifying weak coupling in dec-pomdps. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 29–36. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[128] Miao Liu, Girish Chowdhary, Jonathan P. How, and Lawrence Carin. Transfer learning for reinforcement learning with dependent dirichlet process and gaussian process. In *Neural Information and Processing Systems*, Lake Tahoe, NV, December 2012. workshop on Bayesian Nonparametric Models For Reliable Planning And Decision-Making

Under Uncertainty.

[129] Trevor Campbell, Miao Liu, Brian Kulis, Jonathan P. How, and Lawrence Carin. Dynamic clustering via asymptotics of the dependent dirichlet process. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[130] Andrew N. Kopeikin, Sameera S. Ponda, and Jonathan P. How. *Handbook of Unmanned Aerial Vehicles*, chapter Control of Communication Networks for Teams of UAVs. Springer, 2012.