Studies in Computational Intelligence 603

Samuel Barrett

Making Friends on the Fly: Advances in Ad Hoc Teamwork

Springer

Studies in Computational Intelligence

Volume 603

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

About this Series

The series "Studies in Computational Intelligence" (SCI) publishes new developments and advances in the various areas of computational intelligence—quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life sciences, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems, and hybrid intelligent systems. Of particular value to both the contributors and the readership are the short publication timeframe and the worldwide distribution, which enable both wide and rapid dissemination of research output.

More information about this series at http://www.springer.com/series/7092

Samuel Barrett

Making Friends on the Fly: Advances in Ad Hoc Teamwork

Springer

Samuel Barrett
Kiva Systems
North Reading, MA
USA

ISSN 1860-949X ISSN 1860-9503 (electronic)
Studies in Computational Intelligence
ISBN 978-3-319-18068-7 ISBN 978-3-319-18069-4 (eBook)
DOI 10.1007/978-3-319-18069-4

Library of Congress Control Number: 2015938747

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

In loving memory of my mother,
Ellen Lee Rubinstein Barrett
I will always remember you

Acknowledgments

Despite what the title page implies, writing the book was far from a solitary task. I have many people to thank for making this work possible. First and foremost, I want to thank my Ph.D. advisor Peter Stone, whose advice has guided me through the difficulties of research and graduate school. His guidance has helped me to explore exciting ideas and avoid going down too many wrong paths while also helping me to grow as a researcher. Without his ongoing help, my research would have taken a substantially longer time, if it were completed at all. I also want to thank my committee: Ray Mooney, Risto Miikkulainen, Kristen Grauman, and Sarit Kraus.

I have collaborated with a number of brilliant collaborators. I especially thank my coauthors on the papers that introduced the work described in this book: Noa Agmon, Noam Hazon, Sarit Kraus, and Avi Rosenfeld. I have had many useful and informative conversations with them that have guided my research.

In addition, I have to thank many of my fellow members of the Learning Agents Research Group (LARG), with whom I have had many conversations, traveled, played board games, gone rock climbing, played racquetball, and ran many miles. They have helped me when I was stuck, listened when I needed to complain, and given me support when I needed it. They include Noa Agmon, Tsz-Chiu Au, Doran Chakraborty, Katie Genter, Matthew Hausknecht, Todd Hester, Tobias Jung, Shivaram Kalyanakrishnan, Piyush Khandelwal, Brad Knox, Juhyun Lee, Matteo Leonetti, Elad Liebman, Patrick MacAlpine, Jacob Menashe, Sanmit Narvekar, Michael Quinlan, Adam Setapen, and Daniel Urieli. I especially thank my fellow SPL RoboCup team members from over the years for their tireless efforts and aid in dealing with complexities of robotic competitions. In addition, I would like to thank the other computer science graduate students that I have spent countless hours talking with, numbering too many to be named here.

I also want to thank Sarit Kraus for allowing me to visit her lab at Bar-Ilan University to continue my research in the fall of 2012. All of the people I met there made great efforts to help me adapt to living and working in Israel. I especially want to thank Noam Peled for helping me find a place to live, his advice, and for showing me around. I want to thank Moshe Bitan for his friendship and interesting

conversations. Also, I want to thank Noa Agmon and Noam Hazon for their many productive conversations. In addition, Noa Agmon lending me a bicycle made my life much easier.

I would like to thank my undergraduate research advisor, Jeffrey V. Nickerson, for his help in teaching me to be a researcher. In addition, my advisor at an NSF REU, Xiaojun Qi, helped me to become the researcher I am today.

I would also like to thank the UTCS staff members that have helped me over the years. I would especially like to thank Stacy Miller for her help countless times, especially with figuring out travel rules to attend conferences. In addition, I would like to thank Lydia Griffith for helping me navigate the bureaucracy of graduate school. Furthermore, I appreciate the staff members' help with the condor, maintaining machines, and building the robot soccer field.

I want to thank my family for their continued support. My mother's constant encouragement for me to explore ideas that interested me has been invaluable throughout my life. I want to thank my father for his continuing advice and enthusiasm. In addition, the support of my siblings and their spouses has helped me become who I am. I would like to thank my flatmate, Wenke Li, for putting up with me for longer than he should have. Finally, I would like to thank my girlfriend, Maya Liu, for her support and patience while I worked on this book.

Contents

List of Tables

List of Figures

Abstract

Given the continuing improvements in design and manufacturing processes in addition to improvements in artificial intelligence, robots are being deployed in an increasing variety of environments for longer periods of time. As the number of robots grows, it is expected that they will encounter and interact with other robots. Additionally, the number of companies and research laboratories producing these robots is increasing, leading to the situation where these robots may not share a common communication or coordination protocol. While standards for coordination and communication may be created, we expect that any standards will lag behind the state-of-the-art protocols and robots will need to additionally reason intelligently about their teammates with limited information. This problem motivates the area of *ad hoc teamwork* in which an agent may potentially cooperate with a variety of teammates in order to achieve a shared goal. We argue that agents that effectively reason about ad hoc teamwork need to exhibit three capabilities: (1) robustness to teammate variety, (2) robustness to diverse tasks, and (3) fast adaptation. This book focuses on addressing all three of these challenges. In particular, this book introduces algorithms for quickly adapting to unknown teammates that enable agents to react to new teammates without extensive observations.

The majority of existing multiagent algorithms focus on scenarios where all agents share coordination and communication protocols. While previous research on ad hoc teamwork considers some of these three challenges, this book introduces a new algorithm, PLASTIC, that is the first to address all three challenges in a single algorithm. PLASTIC adapts quickly to unknown teammates by reusing knowledge it learns about previous teammates and exploiting any expert knowledge available. Given this knowledge, PLASTIC selects which previous teammates are most similar to the current ones online and uses this information to adapt to their behaviors. This book introduces two instantiations of PLASTIC. The first is a model-based approach, PLASTIC–Model, that builds models of previous teammates' behaviors and plans online to determine the best course of action. The second uses a policy-based approach, PLASTIC–Policy, in which it learns policies for cooperating with past teammates and selects from among these policies online. Furthermore, we introduce a new transfer learning algorithm, TwoStageTransfer,

that allows transferring knowledge from many past teammates while considering how similar each teammate is to the current ones.

We theoretically analyze the computational tractability of PLASTIC–Model in a number of scenarios with unknown teammates. Additionally, we empirically evaluate PLASTIC in three domains that cover a spread of possible settings. Our evaluations show that PLASTIC can learn to communicate with unknown teammates using a limited set of messages, coordinate with *externally-created* teammates that do not reason about ad hoc teams, and act intelligently in domains with continuous states and actions. Furthermore, these evaluations show that TwoStageTransfer outperforms existing transfer learning algorithms and enables PLASTIC to adapt even better to new teammates. We also identify three dimensions that we argue best describe ad hoc teamwork scenarios. We hypothesize that these dimensions are useful for analyzing similarities among domains and determining which can be tackled by similar algorithms in addition to identifying avenues for future research. The work presented in this book represents an important step towards enabling agents to adapt to unknown teammates in the real world. PLASTIC significantly broadens the robustness of robots to their teammates and allows them to quickly adapt to new teammates by reusing previously learned knowledge.

Chapter 1
Introduction

Robots are becoming cheaper and more durable as manufacturing processes improve, and they are becoming useful for an increasing number of tasks as artificial intelligence improves. These improvements are leading to robots being deployed in more environments for longer periods of time, and we believe that this trend will only grow. As this robotic proliferation continues, it is expected that robots will encounter and interact with a growing variety of other types of robots. In many cases, these interacting robots may share a set of common goals, in which case it will be desirable for them to cooperate with each other. However, existing research into getting multiagent teams to accomplish shared goals assumes that all of these robots share common communication and coordination protocols, an assumption that becomes increasingly unlikely given the rapidly growing number of companies and laboratories designing robots. It is possible that some standards for coordination or communication will be created for all robots to support. We believe that given the speed of changes in artificial intelligence and robotics, if these standards are created, they will lag behind the state-of-the-art protocols and agents will need to intelligently reason about their teammates in addition to the information that these protocols provide. Therefore, in order to effectively cooperate with these new teammates, it is desirable for these robots to observe and adapt to their teammates to accomplish their shared goals.

For example, after a disaster, it is helpful to use robots to search the site and rescue survivors. However, the robots may come from a variety of sources and may not be designed to cooperate with each other, such as in the response to the 2011 Tohoku earthquake and tsunami [1–4]. If these robots are not pre-programmed to cooperate, they may not share information about which areas have been searched; or worse, they may unintentionally impede their teammates' efforts to rescue survivors. Therefore, it is desirable for robots to be designed to observe their teammates and adapt to them, forming a cohesive team that quickly searches the area and rescues the survivors.

This idea epitomizes the spirit of *ad hoc teamwork*. In ad hoc teamwork settings, agents encounter a variety of teammates and try to cooperate in order to accomplish a shared goal. In ad hoc teamwork research, researchers focus on designing a single agent or subset of agents that can cooperate with a variety of teammates. The desire

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_1

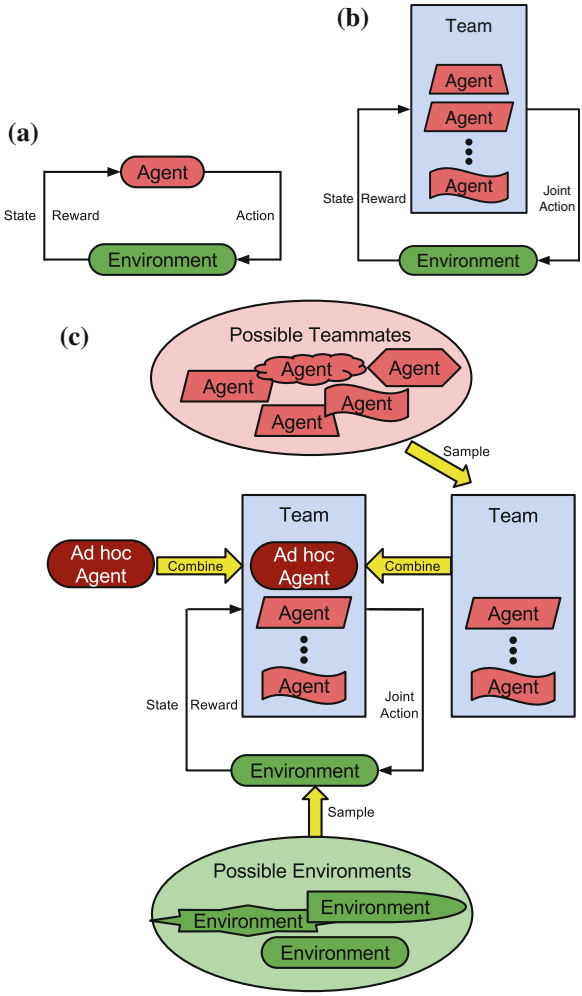

Fig. 1.1 Foci of agent based research. **a** A view of a single agent interacting with its environment used by many reinforcement learning algorithms. **b** A standard view of a unified team interacting with the environment. **c** The ad hoc teamwork setting in which an agent cooperates with an ad hoc team of agents to accomplish shared goals on a given environment where the teammates and the environment are each drawn from diverse sets at the beginning of an episode

is for agents designed for ad hoc teamwork to quickly learn about these teammates and determine how they should act on this new team to achieve their shared goals. Agents that reason about ad hoc teamwork will be robust to changes in teammates in addition to changes in the environment.

In this book, the word "agent" refers to an entity that repeatedly senses its environment and takes actions that affect this environment, shown visually in Fig. 1.1a. As a shorthand, the terms ad hoc team agent and ad hoc agent are used in this book to refer to an agent that reasons about ad hoc teamwork. The environment includes the dynamics of the world the agent interacts with, as well as defining the observations received by the agent. We treat the other agents in the domain as *teammates* because they share a set of common goals; they are fully cooperative in the terminology of game theory.

Previous research has investigated multiagent teams and introduced a number of algorithms for coordinating teams [5–8]. This view of multiagent teams is shown in Fig. 1.1b. However, these approaches require that all agents in the team have shared coordination and communication protocols, so these approaches do not apply in all scenarios. Another approach is to treat other agents as part of the environment. This approach permits the agent to apply single agent learning algorithms to adapt to these teammates, as shown in Fig. 1.1a. However, this view may prevent the team from completing tasks that require explicit coordination, where performing only part of a coordinated action results in a poor performance for the team. In addition, explicitly reasoning about the dynamics of these other agents allows the ad hoc agent to factor the domain, significantly reducing the complexity of the environment.

Rather than adopting either of these approaches, this book focuses on creating a single agent or small subset of agents that cooperate with teammates coming from a variety of sources without directly altering the behavior of these teammates, as shown in Fig. 1.1c. This approach explicitly models that there are other intelligent agents operating in the domain. In addition, rather than focusing on a single task, these agents may face a variety of tasks.

This chapter presents the motivation and objectives for this book as well as an overview of the entire book. Given the preceding motivation for this book, we present the central research question in Sect. 1.1. Next, we present an overview of the algorithm, PLASTIC, that we propose for solving this problem in Sect. 1.2. In Sect. 1.3, we explain the six contributions of this book. Finally, Sect. 1.4 previews the remaining chapters of this book and gives a roadmap to reading this book.

1.1 Research Question

In order to be responsive to different teammates and environments, a fully general ad hoc agent needs two general classes of capabilities: (1) the ability to learn about the environment and calculate the actions necessary to achieve its goal, and (2) the ability to reason about teamwork and learn about its teammates. Previous work in reinforcement learning has largely focused on how an agent should learn about the dynamics of the environment, e.g. [9, 10]. Therefore, this book leverages such past research about (1) and expand this work in the new direction of (2), reasoning about the team and social knowledge required for effective teamwork. To this end, this book identifies three classes of capabilities needed by ad hoc team agents:

1. **Robustness to teammate variety**. Ad hoc team agents should be able to cooperate with a wide variety of teammates following unknown behaviors. To this end, ad hoc agents should learn about their teammates and adapt to their behaviors.
2. **Robustness to diverse tasks**. Ad hoc team agents should be able to cooperate with the teammates to accomplish an array of diverse tasks. An algorithm for ad hoc teamwork that is only applicable to a single task is insufficient for a fully general ad hoc team agent. To be robust, an ad hoc team agent should be able to

adapt to new tasks, while additionally deciding when to take actions to explore its teammates' behaviors.

3. **Fast adaptation**. Ad hoc team agents should be adaptive without lengthy observations of the current teammates or tasks; they should be able to quickly adapt to these new problems and perform effectively with few interactions. In real world settings, robots are unable to interact with their teammates in the given environment for extended periods of time before attempting their task. Therefore, ad hoc agents must be sample efficient, learning to cooperate in few interactions with their teammates. Ideally, ad hoc agents should also exploit any information they have learned about previous teammates and environments to speed up the learning process. This means that ad hoc agents may not achieve optimal performance; instead they may need to compromise by quickly converging to well-performing but suboptimal behaviors.

With these three capabilities in mind, the key question that this book addresses is:

How can an agent cooperate with teammates of uncertain types on a variety of tasks?

Fully answering this question is a long-term challenge beyond the scope of a single book. However, this book moves towards answering this question. Specifically, we introduce an algorithm, PLASTIC, that satisfies the three desired capabilities above. PLASTIC allows ad hoc agents to quickly adapt to a variety of unknown teammates, and our analysis shows that PLASTIC is effective on 3 very different domains, including a complex simulation of robot soccer.

1.2 Algorithm Overview

The primary algorithmic contribution of this book is the Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC) algorithm to address the research question posed in the previous section. Rather than trying to "fit in" with its team by copying their behavior, PLASTIC employs learning algorithms that allow an ad hoc agent to improve over its teammates' potentially suboptimal behaviors. However, learning about each teammate as if it is completely new severely limits the speed that the ad hoc agent can adapt to its teammates. Therefore, PLASTIC speeds up this learning by reusing information it has learned about previous teammates. In addition, PLASTIC allows developers to provide expert knowledge about potential teammates in order to speed up adaptation. PLASTIC then tries to see which of the past teammates and expert-provided information best represents the new teammates that it encounters. This approach allows PLASTIC to quickly adapt to unknown teammates in a variety of domains. The full PLASTIC algorithm is introduced in full detail in Sect. 5.1.

This book presents two instantiations of the general PLASTIC algorithm: PLASTIC–Model and PLASTIC–Policy. When learning about previous teammates, PLASTIC–Model represents its knowledge as models that map current world states to its teammates' actions, where one model is learned for each past teammate. Expert knowledge is similarly represented by models of potential teammates' behaviors. When it encounters new teammates, PLASTIC–Model maintains its beliefs over which model best represents these new teammates. Using its belief distribution over these models, PLASTIC–Model can plan the best actions to cooperate with its teammates. In this book, we use decision trees to learn models of past teammates [11], update the probabilities of models using the polynomial weights algorithm [12], and plan using Upper Confidence bounds for Trees (UCT) [13]. We present the full PLASTIC–Model algorithm in Sect. 5.2.

When the ad hoc agent has a limited number observations of its new teammates, it can also build new models of their behaviors while reusing information about past teammates. Specifically, we introduce a new transfer learning algorithm, TwoStageTransfer, that allows PLASTIC–Model to reuse information about many previous teammates to create a model of the new teammates, exploiting the fact that some teammates are more similar to the new teammates than others. TwoStageTransfer is completely specified in Sect. 5.2.4.

In complex domains, planning algorithms may run into problems due to inaccuracies in their models or due to limitations in computational power. Therefore, directly learning policies for cooperating with various teammates may be more effective. PLASTIC–Policy employs this approach, where knowledge about past teammates is represented as the policies that are used to cooperate with these teammates. Similar to PLASTIC–Model, one policy is learned for each past teammate. In PLASTIC–Policy, experts can provide policies for cooperating with other teammates if they are available. Then, PLASTIC–Policy combines these policies, determines which policies enable it to best cooperate with the current teammates, and selects the actions specified by the most likely policy. In this book, we use fitted Q iteration (FQI) [14] to learn policies for past teammates. To update the probabilities of different teammate types, we build a nearest neighbor model of the teammates' behaviors using the samples collected for FQI and update the probabilities of different teammate types using the polynomial weights algorithm [12]. We present the complete PLASTIC–Policy algorithm in Sect. 5.3.

1.3 Contributions

This book provides the following six major contributions to the field:

1. *PLASTIC*: As described above, this book introduces the general PLASTIC algorithm and its two instantiations: PLASTIC–Model and PLASTIC–Policy. These algorithms are the first generally applicable algorithms that permit ad hoc team agents to quickly adapt to a variety of unknown teammates in many domains.

These algorithms combine knowledge learned from past teammates with any available expert knowledge to adapt to new teammates much more quickly than existing approaches. PLASTIC is presented in full detail in Chap. 5.

2. *Theoretical Analysis*: This book proves that PLASTIC–Model is computationally tractable in a number of scenarios in the bandit domain. We show that the problem can be modeled as a POMDP and bound the complexity of learning the ε-policy in several versions of the domain that vary in the prior knowledge available to PLASTIC–Model. This theoretical analysis is presented in Chap. 6.
3. *Reasoning about Communication*: The majority of prior work in ad hoc teamwork considers cases where communication is not available. However, as the connectivity of devices grows, so does the chances of ad hoc team agents being able to communicate some information with their teammates, though they may not know how these teammates will interpret these messages. Therefore, this book addresses this gap by considering a scenario in which communication is available, namely the bandit domain described in Sect. 3.2.1. Theoretical analysis of the bandit domain with communication are presented in Chap. 6, and empirical analysis of it are given in Sect. 7.1.
4. *TwoStageTransfer*: This book introduces a new transfer learning algorithm, TwoStageTransfer. TwoStageTransfer can efficiently incorporate knowledge coming from many sources to the target setting, exploiting the knowledge that some sources are more similar to the target than others. TwoStageTransfer is a general transfer learning algorithm, but we only apply it to ad hoc teamwork settings in this book. We empirically evaluate TwoStageTransfer for quickly learning models of new teammates given observations of past teammates. TwoStageTransfer is presented in full detail in Sect. 5.2.4, and empirical results of it are presented in Sect. 7.2.7.
5. *Empirical Evaluation*: This book empirically evaluates PLASTIC in 3 different domains, with varying amounts of knowledge about its teammates. This evaluation considers scenarios not covered in the theoretical analysis. Specifically, we focus on *externally-created* teammates, which are created by other developers without considering ad hoc teamwork. Notably, this evaluation also includes a complex ad hoc teamwork problem in the form of the half field offense task in the 2D simulated soccer domain. All empirical results are presented in Chap. 7.
6. *Taxonomy of Ad Hoc Teamwork*: This book identifies three dimensions that we believe are the most informative for analyzing ad hoc teamwork domains and teammates. We believe that domains with similar values along these dimensions will be tractable for similar algorithms, while domains with significantly different values are more likely to be solved using different approaches. These dimensions are presented in full detail in Sect. 2.3. In that section, we also discuss how these dimensions describe the three domains used in this book. In addition, we use these dimensions to analyze the related ad hoc team research in Sect. 4.4.4.

These contributions are described in detail throughout the remainder of this book.

1.4 Book Overview

The remainder of this book is organized as follows.

Chapter 2: This chapter motivates and describes the problem studied in this book, namely the problem of ad hoc teamwork. Then, the chapter presents the framework we use to evaluate ad hoc teamwork agents. Finally, the chapter specifies three dimensions of ad hoc teamwork problems that are useful for analyzing the similarity of problems as well as which algorithms apply to the problems. These dimensions serve as the basis for Contribution 6.

Chapter 3: In this chapter, we present the background information required to understand the remainder of the book. Specifically, we present an overview of Markov Decision Processes (MDPs), Partially Observable Markov Decision Processes (POMDPs), and the algorithms used to solve them in addition to an overview of transfer learning. Then, the chapter describes the 3 domains used to evaluate PLASTIC: the multi-armed bandit domain, the pursuit domain, and the half field offense task in the simulated 2D robot soccer domain. In addition, the chapter discusses the teammates used in these three domains, including the *externally-created* teammates which were created by other developers. The bandit domain is used to investigate communication (Contribution 3) and in the theoretical analysis for Contribution 2. All three domains are used in the empirical evaluation (Contribution 5).

Chapter 4: This chapter situates PLASTIC in the literature. We begin by discussing past work on multiagent coordination, where developers can control the entire team instead of a single agent on the team. Then, we look at research on opponent modeling, where agents learn to adapt to their opponents instead of teammates. We follow this by an overview of some of the most relevant research in each domain used in our evaluations. Finally, we discuss the current state of the art research into ad hoc teamwork and describe how these papers relate to the work presented in this book. We use the three dimensions of ad hoc teamwork to analyze the related literature as part of Contribution 6.

Chapter 5: In this chapter, we specify the general PLASTIC algorithm and the two instantiations of it used in this book: PLASTIC–Model and PLASTIC–Policy. PLASTIC enables an ad hoc agent to quickly adapt to unknown teammates on several domains. To accomplish this task, PLASTIC learns about previous teammates and combines this knowledge with any expert knowledge the developer can encode about potential teammates. When encountering new teammates, PLASTIC determines which of the past teammates are most similar to the current ones and reuses the information learned about them. Different learning and action selection algorithms can be used depending on the domains, but the general architecture remains the same. PLASTIC–Model adopts a model-based approach that uses planning to discover effective actions, while PLASTIC–Policy uses a policy-based approach and selects between these policies. This chapter also specifies our new transfer learning algorithm, TwoStageTransfer, that can aid PLASTIC in adapting to new teammates by combining information

coming from many past teammates, some of which are more similar to the new teammates than others. This chapter presents PLASTIC for Contribution 1 in addition to TwoStageTransfer for Contribution 4.

Chapter 6: This chapter presents a theoretical analysis of PLASTIC–Model. Specifically, it analyzes the complexity of applying PLASTIC–Model to the bandit domain. We model the bandit problem as a POMDP and investigate the computational tractability of calculating the ε-optimal behavior in the resulting POMDP. We consider several variations of the bandit domain, varying in the knowledge the ad hoc agent has about its teammates and its environment. The theoretical analysis in this chapter is Contribution 2 as well as part of Contribution 3.

Chapter 7: We present the empirical analyses of PLASTIC in this chapter. These analyses cover the 3 domains described in Chap. 3: the bandit, pursuit, and half field offense domains. Our tests evaluate both PLASTIC–Model and PLASTIC–Policy on a range of tasks, varying the prior knowledge of the ad hoc agent as well as the teammates it encounters. The results of these tests show the effectiveness of PLASTIC for enabling ad hoc team agents to quickly adapt to a variety of unknown teammates. This chapter presents the results that form Contribution 5. In addition, this chapter looks at the empirical usefulness of communication in the bandit domain as part of Contribution 3.

Chapter 8: This chapter summarizes the contributions of this book. In addition, it identifies areas for future research into ad hoc teamwork.

Appendix A: This appendix presents the full behaviors of the hand-coded predators used in the pursuit domain described in Sect. 3.2.2. The results using these predators as teammates are presented in Sect. 7.2.

While this book is written to be read from start to finish, some sections can be omitted if only interested in a specific aspect of the work. Specifically, it is possible to isolate the theoretical analyses from the empirical ones and to understand the related work without reading most of the book. To understand the theoretical analyses in Chap. 6, it is useful to understand the PLASTIC algorithm presented in Chap. 5 as well as the problem definition in Chap. 2 as well as the discussion of POMDPs in Sect. 3.1.5 and the bandit problem described in Sect. 3.2.1. The empirical analyses in Chap. 7 depend on the reader understanding the PLASTIC algorithm from Chap. 5 in addition to the problem definition (Chap. 2) and the background (Chap. 3). The discussion of related work in Chap. 4 is largely self-contained, but it does help to understand the problem definition and dimensions that describe ad hoc teamwork presented in Chap. 2.

References

1. Huang, Ya-Wen, Y. Sasaki, Y. Harakawa, E.F. Fukushima, and S. Hirose. 2011. Operation of underwater rescue Robot anchor diver III during the 2011 Tohoku earthquake and tsunami. In *OCEANS 2011*, Sept 2011, 1–6.
2. Murphy, R.R., K.L. Dreger, S. Newsome, J. Rodocker, E. Steimle, T. Kimura, K. Makabe, F. Matsuno, S. Tadokoro, and K. Kon. 2011. Use of remotely operated marine vehicles at Minamisanriku and Rikuzentakata Japan for disaster recovery. In *2011 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, Nov 2011, 19–25.
3. Nagatani, K., S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada. 2011. Redesign of rescue mobile Robot Quince. In *2011 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, Nov 2011, 13–18.
4. Richardson, D.K. 2011. Robots to the rescue? *Engineering Technology* 6(4): 52–54.
5. Decker, Keith S., and Victor R. Lesser. 1995. Designing a family of coordination algorithms. In *International conference on multi-agent systems (ICMAS)*, June 1995, 73–80.
6. Grosz, B., and S. Kraus. 1996. Collaborative plans for complex group actions. *Artificial Intelligence (AIJ)* 86: 269–368.
7. Stone, Peter, and M. Veloso. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8(3): 345–383.
8. Tambe, Milind. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)* 7: 81–124.
9. Kalyanakrishnan, Shivaram, and Peter Stone. 2011. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning (MLJ)* 84(1–2): 205–247.
10. Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement learning: An introduction*. Cambridge: MIT Press.
11. Quinlan, John Ross. 1993. *C4.5: Programs for machine learning*, vol. 1. Morgan Kaufmann.
12. Blum, A., and Y. Mansour. 2007. *Algorithmic game theory*, chapter Learning, regret minimization, and equilibria. Cambridge University Press.
13. Kocsis, Levente, and Csaba Szepesvari. 2006. Bandit based Monte-Carlo planning. In *Prooceedings of the seventeenth European conference on machine learning (ECML)*.
14. Ernst, Damien, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)* 503–556.

Chapter 2
Problem Description

This chapter describes the problem setting investigated in this book. Rather than considering unified teams of agents designed by a single designer, we consider controlling a single agent on a newly created team. The following sections describe the problem in more depth as well as the evaluation framework used to measure performance on this problem. In addition, this chapter investigates an approach for describing the different dimensions of ad hoc team problems.

2.1 Ad Hoc Teamwork

Robots are becoming cheaper and more durable and are therefore being deployed in more environments for longer periods of time. As robots continue to proliferate in this way, many of them will encounter and interact with a variety of other kinds of robots. In many cases, these interacting robots will share a set of common goals, in which case it will be desirable for them to cooperate with each other. In order to effectively perform in new environments and with changing teammates, they should observe their teammates and adapt to achieve their shared goals. For example, after a disaster, it is helpful to use robots to search the site and rescue survivors. However, the robots may come from a variety of sources and may not be designed to cooperate with each other, such as in the response to the 2011 Tohoku earthquake and tsunami [2–5]. These robots were used to investigate the Fukushima Daiichi Nuclear Power Station, clear a fishing port, and find victims trapped underwater. These robots were remotely controlled and therefore derived any cooperation from their human operators. Robots that operate autonomously will have to be designed for cooperation as they will not have human operators providing cooperation. If these autonomous robots are not pre-programmed to cooperate, they may not share information about which areas

This chapter contains material from the publication: [1].

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_2

have been searched; or worse, they may unintentionally impede their teammates' efforts to rescue survivors. Therefore, in the future, it is desirable for robots to be designed to observe their teammates and adapt to them, forming a cohesive team that quickly searches the area and rescues the survivors.

This idea epitomizes the spirit of *ad hoc teamwork*. In ad hoc teamwork settings, agents encounter a variety of teammates and try to accomplish a shared goal. Ideally, agents designed for ad hoc teamwork try to quickly learn about their teammates and figure out how they should try to fit into the team. Agents that reason about ad hoc teamwork will be robust to changes in teammates and the environment. They must be adaptive and resourceful, learning how to accomplish the team's goals.

In this book, the word "agent" refers to an entity that repeatedly senses its environment and takes actions that affect this environment. Robots are examples of agents, as are software agents that bid for advertisements. As a shorthand, the terms ad hoc team agent and ad hoc agent are used in this book to refer to an agent that reasons about ad hoc teamwork. The environment includes the dynamics of the world the agent interacts with, as well as defining the observations received by the agent. In addition, the ad hoc agent will have to interact with teammate agents that are attempting to accomplish the same goals as the ad hoc agent. This book considers ad hoc agents that explicitly reason about the behaviors of their teammates separately from the environment because this factoring significantly reduces the complexity of the learning problem. Previous work has largely assumed that all agents in the domain will act as a unified team and are designed to work with their specific teammates [6–9]. On the other hand, this book will focus on creating a single agent that cooperates with teammates coming from a variety of sources without directly altering the behavior of these teammates. This agent will need to adapt to these different teammates and learn to cooperate with them on the fly.

The differences of this book from prior work are presented visually in Fig. 2.1. One existing area of research into how agents should behave is reinforcement learning (RL). Generally, RL problems revolve around a single agent learning by interacting with its environment. In RL problems, agents receive sparse feedback about the quality of sequences of actions. Generally, RL algorithms model other agents as part of the environment and try to learn the best policy for the single agent given this environment. In addition, RL algorithms usually learn from scratch in each new environment, ignoring information coming from previous environments. However, there is a growing body of work on applying transfer learning to RL to allow agents to reuse prior experiences [10]. Figure 2.1a shows the standard RL view of an agent interacting with its environment. Figure 2.1b represents a common multiagent view of a unified team interacting with the environment where the agents model their teammates as being separate from the environment. In this case, the team is designed before being deployed to cooperate with these specific agents to interact with a fixed environment. However, these agents rely on knowing their teammates and usually require an explicit communication and/or coordination protocol to be shared among the whole team [11, 12]. On the other hand, this book will focus on ad hoc teams drawn from a set of possible teammates, where the team tackles a variety of possible environments as shown in Fig. 2.1c. In this case, the teammates are not programmed

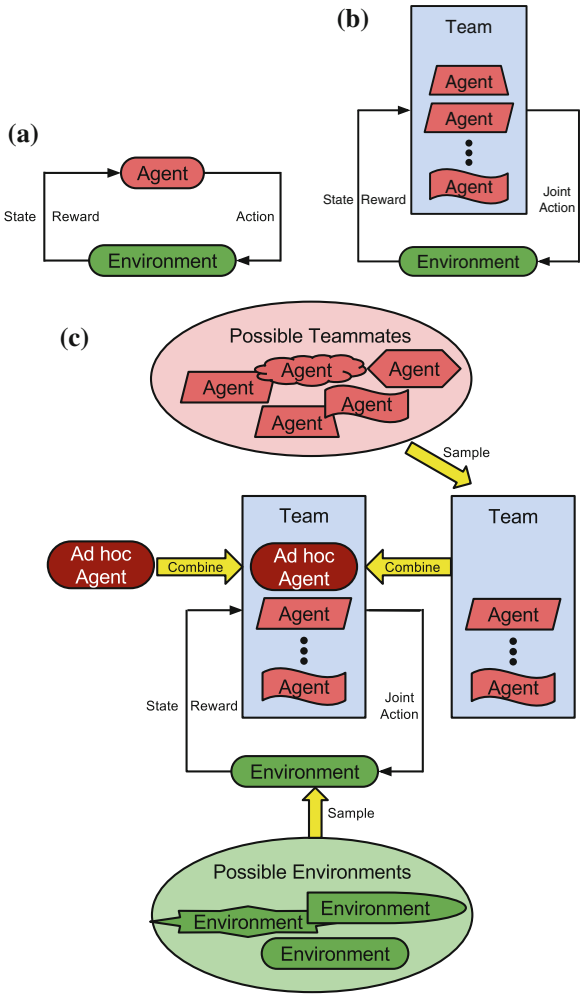

Fig. 2.1 Foci of agent based research. **a** A view of a single agent interacting with its environment used by many reinforcement learning algorithms. **b** A standard view of a unified team interacting with the environment. **c** The ad hoc teamwork setting in which an agent cooperates with an ad hoc team of agents to accomplish shared goals in a given environment where the teammates and the environment are each drawn from diverse sets at the beginning of an episode

to cooperate with this specific ad hoc agent, and they must be treated as fixed and given. Instead, this research focuses on enabling the ad hoc agent to cooperate with a variety of teammates in a range of possible environments.

In order to be responsive to different teammates and environments, a fully general ad hoc agent needs two general classes of capabilities: (1) the ability to learn how to act in an environment to maximize reward, and (2) the ability to reason about teamwork and learn about its teammates. Previous work in reinforcement learning has largely focused on how an agent should learn about the dynamics of the environment [13, 14]. Therefore, this book will leverage such past research about (1) and expand this work in the new direction of (2), reasoning about the team and social knowledge required for effective teamwork.

Ad hoc teamwork problems can be encountered in a variety of real world scenarios. As described in the example above, in search and rescue scenarios, robots from different developers need to cooperate quickly. Furthermore, as more robots enter society, their interactions will increase. In the near future, personal assistant robots may need to interact with other service robots to accomplish their tasks. In addition, the introduction of autonomous cars opens up an interesting area for ad hoc teamwork: cooperating with human drivers. Cars on the road have the shared goal of reaching their destinations quickly and safely, and they need to cooperate with the other cars in order to accomplish these goals. These agents have very limited observations of the other cars, and therefore must adapt quickly.

Another area where ad hoc teamwork comes into play is when robots need to accomplish tasks in workplace settings with human teammates. These settings include manufacturing jobs, where new robots are now able to work more closely with humans, and using robots in warehouses for moving products. The robots are likely to interact with a variety of humans, and therefore need to adapt quickly to these new teammates. While the robots and humans share a common goal, communication between them is limited; humans cannot quickly and fully specify their intentions to the level used in existing multiagent coordination algorithms. Therefore, it is desirable for the robots to reason about ad hoc teamwork.

Another interesting application of ad hoc teamwork is in the area of games. Game-playing agents interact with humans and need to adapt to them with only limited observations. These interactions are incredibly complex, and existing approaches rely heavily on heuristic approaches with only limited adaptations [15–17]. Reasoning about ad hoc teamwork would allow virtual agents in video games to adapt to their human teammates.

2.2 Evaluation Framework

In an ad hoc team, agents need to be able to cooperate with a variety of previously unseen teammates. Rather than developing protocols for coordinating an entire team, ad hoc team research focuses on developing agents that cooperate with teammates in the absence of such explicit protocols. Therefore, we consider a single agent cooperating with teammates that may or may not adapt to its behavior. In this scenario, we can only develop algorithms for the ad hoc team agent, without having any direct control over the other teammates.

However, directly measuring teamwork is difficult. In many cases, the only easily measurable aspect is the overall performance of the team, which makes it difficult to assign credit to each agent. By placing an agent on a variety of teams and measuring those teams' performances, we can estimate how good the agent is at teamwork.

Therefore, we introduce an algorithm that evaluates an ad hoc team agent while considering the teammates and domains it may encounter. This framework is specified in Algorithm 2.1 and visually presented in Fig. 2.2. According to this framework, the performance of the ad hoc team agent a depends on the distribution of problem

Algorithm 2.1 Ad hoc agent evaluation

1: **function** Evaluate:
inputs:
a ▷ the ad hoc agent
A ▷ the set of possible teammate agents
D ▷ the set of possible domains
outputs:
$\frac{r}{n}$ ▷ the average performance (reward)
params:
s_{min} ▷ the minimal acceptable performance of a team
n ▷ the number of iterations

2: Initialize: $r = 0$
3: **for** $i = 1$ to n **do**
4: Sample a task d from D
5: Randomly draw a subset of agents B, from A such that $E[s(B, d)] \geq s_{min}$
6: Randomly select one agent $b \in B$
7: Create the new team $C = \{a\} \cup B \backslash \{b\}$
8: $r = r + s(C, d)$
9: **return** $\frac{r}{n}$

10: If Evaluate(a_0, A, D) > Evaluate(a_1, A, D) and the difference is significant, we can conclude that a_0 is a better ad hoc team agent than a_1 in domain d over the set of possible teammates A.

domains D and the distribution of possible teammates A that it will cooperate with. For the team B cooperating to execute the task d, $s(B, d)$ is a scalar score representing their effectiveness, where higher scores indicate better performance. The algorithm takes a sampling approach to average the agent's performance across a range of

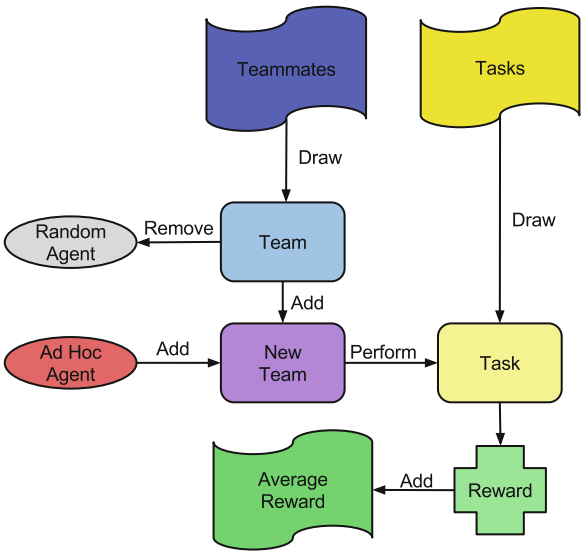

Fig. 2.2 A visual representation of the evaluation algorithm given in Algorithm 2.1

possible tasks and teammates to capture the idea that a good ad hoc team player ought to be robust to a wide variety of teamwork scenarios. We use s_{min} as a minimum acceptable reward for the team to be evaluated, because the ad hoc team agent may be unable to accomplish a task if its teammates are too ineffective, regardless of its own abilities. It is mainly used to reduce the number of samples required to evaluate the ad hoc agents and reduces the noise in the comparisons. Metrics other than the sum of the rewards can be used depending on the domain, such as the worst-case performance.

2.3 Dimensions of Ad Hoc Team Problems

Section 2.2 specified the framework for evaluating ad hoc team agents, but this evaluation depends on the specific domain and teammates that the ad hoc agent may encounter. This section identifies three dimensions of ad hoc teamwork settings that can be used to describe these domains and teammates. We hypothesize that domains with similar values along these dimensions can be tackled by similar algorithms, while domains with very different values will need different algorithms for good performance. For this book, we use these dimensions as a way as classifying problems, but a promising area for future work is to apply these dimensions to predict which algorithms will be effective on different problems.

There are many possible ways that ad hoc team domains can vary, such as the size of the task's state space and the stochasticity of the domain. But, for differentiating among the algorithms in the existing literature, we find the following three to be the most informative.

1. **Team Knowledge**: Does the ad hoc agent know what its teammates' actions will be for a given state, before interacting with them?
2. **Environment Knowledge**: Does the ad hoc agent know the transition and reward distribution given a joint action and state before interacting with the environment?
3. **Reactivity of teammates**: How much does the ad hoc agent's actions affect those of its teammates?

These dimensions affect the difficulty of planning in the domain in addition to how much an ad hoc agent needs to explore the environment and its teammates. When an ad hoc agent has good knowledge, it can plan without considering exploration, but when it has incomplete knowledge, it must reason about the cost and benefits of exploration. The exploration-exploitation problem has been studied previously, but adding in the need to explore the teammates' behaviors and the ability to affect them considerably alters this tradeoff. Sections 2.3.1–2.3.3 provide further details about each of these dimensions, how they are measured, and why they are important for ad hoc teamwork.

To better illustrate the dimensions, we introduce a simple domain to evaluate across each of the dimensions. The domain is described here, and it will be revisited in the discussion of each dimension (Sections 2.3.1–2.3.3).

MatchActions: *This domain is a typical coordination game with two agents, each of which has two actions. If they select the same action, both receive a reward of r_i, where r_i is randomly selected from* $\{0.5, 0.75, 1.0\}$ *for* $i \in 1, 2$, *but fixed for the episode. On the other hand, if both of the agents select different actions, they receive a reward of* 0. *In addition, both agents can observe their teammates' previous actions. The ad hoc agent knows that its teammate is following one of two behaviors*:

- ***FirstAction***: *the teammate always chooses the first action*
- ***BestResponse***: *the teammate chooses the same action as the ad hoc agent did previously*

Therefore, the state can be represented as the previous action taken by the ad hoc agent, called s_0 *if the ad hoc agent chose the first action, and* s_1 *otherwise.*

2.3.1 Team Knowledge

The ad hoc agent's knowledge about its teammates' behaviors gives insight into the difficulty of planning in the domain. The agent's knowledge can range from knowing the complete behaviors of its teammates to knowing nothing about them. Settings with partial information are especially relevant, because in many real world problems, the exact behavior of a teammate may not be known, but some reasonable guidelines of their behaviors exist. For example, when playing soccer, one can usually assume that a teammate will not intentionally pass to the other team or shoot at the wrong goal. If the behaviors are completely known, the agent can reason fully about the team's actions, while if the behaviors are unknown, the agent must learn about them and adapt to find a good behavior.

To estimate the ad hoc agent's knowledge about its teammates' behaviors, we compare the actions the ad hoc agent expects them to take and the ground truth of what actions they take. Specifically, we compare the expected distribution of teammate actions to the true distribution that the teammates follow. To compute the difference between the distributions, we use the Jensen-Shannon divergence measure, which was chosen because it is a smoothed, symmetric variant of the popular Kullback-Leibler divergence measure. Specifically, we denote the Jensen-Shannon divergence by JS where

$$\mathrm{JS}(P, Q) = \frac{1}{2}(\mathrm{KL}(P, M) + \mathrm{KL}(Q, M))$$

and $M = \frac{1}{2}(P + Q)$. The Kullback-Leibler divergence is given by

$$\mathrm{KL}(P, Q) = \sum_i P(i) \log \frac{P(i)}{Qi}$$

When the ad hoc agent has no information about a teammate's action, we assume that it uses the uniform distribution to represent its actions. Therefore, we define the knowledge measure as

$$K(T, P) = \begin{cases} 1 & \text{if } \mathrm{JS}(T, P) = 0 \\ 1 - \dfrac{\mathrm{JS}(T, P)}{\mathrm{JS}(T, U)} & \text{if } \mathrm{JS}(T, P) < \mathrm{JS}(T, U) \\ -\dfrac{\mathrm{JS}(P, U)}{\mathrm{JS}(U, \mathrm{Point})} & \text{otherwise} \end{cases} \tag{2.1}$$

where T is the true distribution, P is the predicted distribution, U is the uniform distribution, and Point is a distribution with all weight on one point (e.g. $[1, 0, 0, \ldots]$). By this definition, $K(T, T) = 1$, so the knowledge is complete if the ad hoc agent knows the true distribution. $K(T, U) = 0$, representing when the ad hoc agent has no knowledge and relies on the uniform distribution. Finally, if the predicted distribution is less accurate than the uniform distribution, then $K(T, P)$ is negative, with a minimum value of -1. This measure captures the range $[0, 1]$ smoothly, but can still be used for the range $[-1, 0]$.[1] However, we generally expect the prediction to be a higher entropy distribution than the true distribution as the ad hoc agent ought to correctly model its uncertainty in its teammates' behaviors rather than being confident and wrong, which keeps the measure in the range $[0, 1]$.

We define the ad hoc agent's knowledge about its teammates' behaviors as the average over the teammates and world states, specifically

$$\mathrm{TeamK} = \frac{\sum_{s=1}^{n} \sum_{t=1}^{k} K(\mathrm{TrueAction}_t(s), \mathrm{PredAction}_t(s))}{nk}$$

where $1 \leq s \leq n$ is the state, $1 \leq t \leq k$ specifies a teammate, $\mathrm{TrueAction}_t(s)$ is the ground truth action distribution for teammate t for state s, and $\mathrm{PredAction}_t(s)$ is the action distribution that the ad hoc agent predicts teammate t to select for state s. Thus, if the ad hoc agent has better information about its teammates' behaviors, the distance between the distributions will be smaller and TeamK will be higher.

Let us now calculate the TeamK for the MatchActions domain. The ad hoc agent has uniform beliefs over its teammate following either the FirstAction or BestResponse behaviors. However, the teammate is actually following the BestResponse behavior. With these beliefs, in s_0, the ad hoc agent expects that its teammate will always chose a_0, so $\mathrm{PredAction}_{s_0} = [1, 0]$. In s_1, the ad hoc agent thinks that the teammate will choose a_0 with probability 0.5 and a_1 with probability 0.5, while it actually chooses a_1 with probability 1. Thus,

[1] One slight anomaly of this measure is that when T is the uniform distribution (e.g. [0.5, 0.5]), K is either 1 when P is exactly correct at [0.5, 0.5] or negative. For all other values of T, K smoothly spans the range $[-1, 1]$.

$$\text{TeamK} = \frac{K([1,0],[1,0]) + K([0,1],[\frac{1}{2},\frac{1}{2}])}{2} = \frac{0+1}{2} = 0.5$$

This value indicates that the ad hoc agent is somewhat knowledgeable about its teammate's actions as it predicts its teammate's actions half the time better than random guessing.

2.3.2 Environmental Knowledge

Another informative dimension is how much knowledge the ad hoc agent has about the effects of a joint action given a state, for example the transition and reward functions. If the ad hoc agent has complete knowledge about the environment, it can plan about what actions it should select more simply than if it must also consider unknown effects of actions. However, if it has incomplete knowledge, it must explore its actions and face the standard problem of balancing exploring the environment versus exploiting its current knowledge.

Similarly to teammate knowledge, we formally define the ad hoc agent's knowledge about the environment's transitions as

$$\text{TransK} = \frac{1}{nm}\sum_{s=1}^{n}\sum_{j=1}^{m} K(\text{TrueTrans}(s,j), \text{PredTrans}(s,j))$$

where $1 \leq s \leq n$ is the state, $1 \leq j \leq m$ is a joint action, K is taken from Eq. (2.1), TrueTrans(s, j) is the ground truth transition distribution from state s given joint action j, and PredTrans is the ad hoc agent's expected transition distribution. If the agent has no information about the transitions, we assume that PredTrans(s, j) is the uniform distribution. Intuitively, if the ad hoc agent knows more about the transition function, then the distance between TrueTrans and PredTrans will be smaller and as a result TransK will be higher. We define the agent's knowledge about the environmental rewards similarly

$$\text{RewardK} = \frac{1}{nm}\sum_{s=1}^{n}\sum_{j=1}^{m} K(\text{TrueReward}(s,j), \text{PredReward}(s,j))$$

We define the environmental knowledge as a 2-dimensional value given by EnvK = (TransK, RewardK).

Revisiting the MatchActions domain, the ad hoc agent knows the true transition function, as it only depends on the ad hoc agent's previous action, so TransK = 1. However, it only knows that the payoff for each action is uniformly drawn from $\{0.5, 0.75, 1.0\}$ and the reward is 0 if the agents' actions do not match. There are 8 possible cases to count over, coming from 2 states, 2 actions for the ad hoc agent, and 2 for its teammate, but the cases fall into 2 sets based on whether the actions

match, each set covering 4 cases. In addition, it does not matter which value each matched action actually takes, so we can simplify the calculation. If the agents take the different actions, the reward is correctly known to be 0. Note that there are four reward values possible: $\{0, 0.5, 0.75, 1.0\}$. Therefore, the knowledge in this case is $K([1,0,0,0],[1,0,0,0]) = 1$. On the other hand, if they take the same actions, the ad hoc agent is unsure which of the three rewards $\{0.5, 0.75, 1.0\}$ it will receive, so the knowledge in this case is $K([0,1,0,0],[0,\frac{1}{3},\frac{1}{3},\frac{1}{3}]) = 0.164$. This leads to

$$\begin{aligned}\text{RewardK} &= \frac{4 * K([1,0,0,0],[1,0,0,0]) + 4 * K([0,1,0,0],[0,\frac{1}{3},\frac{1}{3},\frac{1}{3}])}{8} \\ &= \frac{4 * 1 + 4 * 0.164}{8} \\ &= 0.582\end{aligned}$$

Thus, $\text{EnvK} = (1, 0.582)$. As the agent observes these payoffs, it can refine its knowledge, but we are evaluating these properties prior to the ad hoc agent interacting with its environment.

2.3.3 Teammate Reactivity

The optimal behavior for the ad hoc agent also depends on how much its teammates react to its actions. If its teammates' actions do not depend on the ad hoc agent at all, the ad hoc agent can simply choose its actions to maximize the team reward, as if it were a single agent problem. Considering the actions of its teammates separately from that of the environment may still help computation by factoring the domain. However, if the teammates' actions depend strongly on the ad hoc agent's actions, the ad hoc agent's reasoning should consider what its teammates' reactions will be. If the ad hoc agent is modeling its teammates and its teammates are modeling the ad hoc agent, the problem can become recursive, as is directly addressed by Vidal and Durfee's Recursive Modeling Method [18].

A formal measure of the teammate reactivity needs to capture how different the teammates' actions will be when the ad hoc agent chooses different actions. We measure the distance between the resulting distributions of the teammate joint actions, using the pairwise Jensen-Shannon divergence measures. However, it is desirable for the distance to be 1 when the distributions have no overlap, so we use a normalizing constant of $\log 2$. Thus, we define the *reactivity* of a domain in state s as

$$\text{Reactivity}(s) = \frac{1}{m(m-1)\log 2} \sum_{a=1}^{m} \sum_{a'=1}^{m} \text{JS}(T(s,a), T(s,a'))$$

where JS is the Jensen-Shannon divergence measure, $1 \leq a, a' \leq m$ is the actions available to the ad hoc agent, and $T(s,a)$ is the distribution of the teammates' joint

actions given the state s and the ad hoc agent's action, a. We use $m - 1$ in the denominator because we exclude the case where $a = a'$; in the numerator, the JS measure will be 0 in this case. For the overall reactivity of the domain, we average over the states, resulting in Reactivity $= \frac{1}{n} \sum_{s=1}^{n}$ Reactivity(s). It is possible to consider how an action affects the teammates' actions further in the future, but we restrict our focus to one step reactivity for this book. Note that all of the sums in this formulation can be converted to integrals for continuous states or actions. This formulation is similar to the empowerment measure used by Jung et al. [19], but we consider the ad hoc agent's ability to change the actions of its teammates rather than the environment state.

Let us once again explore this dimension in the context of the MatchActions domain. Although the ad hoc agent is unsure of its teammate's behavior, the teammate is truly playing the BestResponse behavior. Thus, its actions are entirely dependent on the ad hoc agent's actions, so Reactivity $= 1$. If instead the teammate played BestResponse with probability $\frac{9}{10}$ and FirstAction with probability $\frac{1}{10}$, then we would get

$$\text{Reactivity} = \frac{\text{JS}([1,0],[\frac{1}{10},\frac{9}{10}]) + \text{JS}([\frac{1}{10},\frac{9}{10}],[1,0])}{2\log 2} = 0.758$$

Therefore, we can conclude that the agent would still be very reactive, though not as reactive as the BestResponse agent.

2.3.4 Applying the Dimensions

In theory, calculating the dimensions over every possible state is a promising approach. However, as the size of the state space grows, this approach rapidly becomes computationally ineffective. Therefore, it is desirable to approximate the values along each dimension. Specifically, we approximate these values by randomly sampling states and teammates and summing over these samples to calculate approximate values for each of the dimensions. In addition, in continuous state spaces, the summations in the dimension definitions become integrals in the continuous case, but we continue to sample states in these scenarios. Furthermore, the distributions become continuous, but the JS measure can operate over continuous distributions. Specifically, we approximate the JS measure using Monte Carlo sampling in this book. The domains used in this book are described in Sect. 3.2, where we discuss the values of each domain along these dimensions.

2.4 Chapter Summary

This chapter introduces the type of situations this book focuses on: *ad hoc team* problems. In ad hoc teams, agents must adapt to new and unknown teammates without prior coordination, possibly without any explicit communication channels. In addition, this chapter introduces the evaluation framework used to evaluate ad hoc agents. This evaluation framework relies on sampling teams and tasks and then replacing an agent on the team with the ad hoc agent. The resulting team performs the task and receives a reward based on its performance, which is combined with results with other teams and tasks. Finally, this chapter describes 3 dimensions for categorizing ad hoc team problems that indicate which approaches are expected to be effective. These dimensions are: (1) team knowledge, (2) environment knowledge, and (3) team reactivity. This chapter provides the framework for how the rest of the book investigates ad hoc teamwork scenarios. The next chapter will provide an introduction to the algorithms that this book builds upon as well as a description of the domains used to evaluate the proposed algorithm.

References

1. Barrett, Samuel, and Peter Stone. 2012. An analysis framework for ad hoc teamwork tasks. In *Proceedings of the eleventh international conference on autonomous agents and multiagent systems (AAMAS)*, June 2012.
2. Huang, Ya-Wen, Y. Sasaki, Y. Harakawa, E.F. Fukushima, and S. Hirose. 2011. Operation of underwater rescue Robot anchor diver III during the 2011 Tohoku earthquake and tsunami. In *OCEANS 2011*, Sept 2011, 1–6.
3. Murphy, R.R., K.L. Dreger, S. Newsome, J. Rodocker, E. Steimle, T. Kimura, K. Makabe, F. Matsuno, S. Tadokoro, and K. Kon. 2011. Use of remotely operated marine vehicles at Minamisanriku and Rikuzentakata Japan for disaster recovery. In *2011 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, Nov 2011, 19–25.
4. Nagatani, K., S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada. 2011. Redesign of rescue mobile Robot Quince. In *2011 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, Nov 2011, 13–18.
5. Richardson, D.K. 2011. Robots to the rescue? *Engineering Technology* 6(4): 52–54.
6. Decker, Keith S., and Victor R. Lesser. Designing a family of coordination algorithms. In *International conference on multi-agent systems (ICMAS)*, June 1995, 73–80.
7. Grosz, B., and S. Kraus. 1996. Collaborative plans for complex group actions. *Artificial Intelligence (AIJ)* 86: 269–368.
8. Stone, Peter, and Manuela Veloso. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8(3): 345–383.
9. Tambe, Milind. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)* 7: 81–124.
10. Taylor, Matthew E., and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research (JMLR)* 10(1): 1633–1685.
11. Lauer, Martin, and Martin Riedmiller. 2000. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the seventeenth international conference on machine learning (ICML)*. Morgan Kaufmann, 535–542.

12. Xuan, Ping, Victor Lesser, and Shlomo Zilberstein. 2001. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the fifth international conference on autonomous agents (AGENTS)*.
13. Kalyanakrishnan, Shivaram, and Peter Stone. 2011. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning (MLJ)* 84(1–2): 205–247.
14. Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement learning: An introduction*. Cambridge: MIT Press.
15. Millington, Ian, and John Funge. 2009. *Artificial intelligence for games*. 2nd ed. San Francisco: Morgan Kaufmann Publishers Inc.
16. Smed, Jouni, and Harri Hakonen. 2006. *Algorithms and networking for computer games*. Wiley.
17. Buckland, Mat, and Mark Collins. 2002. *AI techniques for game programming*. Premier Press.
18. Vidal, Jose M., and Edmund H. Durfee. 1995. Recursive agent modeling using limited rationality. In *International conference on multi-agent systems (ICMAS)*.
19. Jung, T., D. Polani, and P. Stone. 2010. Empowerment for continuous agent-environment systems. Technical Report AI-10-03, The University of Texas at Austin Computer Science Department.

Chapter 3
Background

While the previous chapter describes the general problem investigated in this book, this chapter describes the mathematical model used to analyze this problem. In addition, this chapter presents the existing algorithms that our approach builds upon. Then, the chapter grounds the general ad hoc teamwork problem in a number of domains that the remainder of the book uses to evaluate the proposed approach. Using the dimensions described in Sect. 2.3, we can analyze these domains as well as the teammates that the ad hoc agent may encounter. Informally, we find that similar algorithms are effective on problems with similar values, but we do not use these values for further algorithm design or selection in this book.

3.1 Background

This section defines the models, Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) we use as the mathematical frameworks for modeling ad hoc team problems. Then, this section presents several existing algorithms used to learn how to act in MDPs. This section concludes with a discussion of transfer learning algorithms, which can be used to efficiently learn models of the teammates' behaviors.

3.1.1 Markov Decision Process

In order to plan and learn in various ad hoc teamwork scenarios, it is helpful to pick a way to model the problem. Agents that need to cooperate in ad hoc teams need to handle sequential decision making problems; therefore, we choose to model these problems as a Markov Decision Process. The Markov Decision Process (MDP) is a standard formalism in reinforcement learning [1], generally used to describe an

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_3

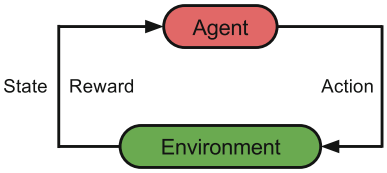

Fig. 3.1 A single agent interacting with the environment in reinforcement learning

agent interacting with its environment. A summary of this interaction is given in Fig. 3.1. An MDP is 4-tuple (S, A, P, R), where S is a set of states, A is a set of actions, $P(s'|s, a)$ is the probability of transitioning from state s to s' when after taking action a, and $R(s, a)$ is a scalar reward given to the agent for taking action a in state s. In the pursuit domain, $s \in S$ corresponds to the current positions of every agent and $a \in A$ is the action that the ad hoc agent chooses (i.e. left, right, up, down, or staying still). In this framework, a policy π is a mapping from states to actions, which defines an agent's behavior for every state. The agent's goal is to find the policy that maximizes its long term expected rewards. The long term expected value from taking action a from state s is the value of the state-action and is denoted $Q(s, a)$. For every state-action pair, $Q^*(s, a)$ represents the maximum long term reward that can be obtained from (s, a) and is defined by the Bellman equation

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a') \tag{3.1}$$

where $0 < \gamma < 1$ is the discount factor representing how much more immediate rewards are worth compared to delayed rewards. The optimal policy π^* can then be derived by choosing the action a that maximizes $Q^*(s, a)$ for every $s \in S$. The goal of most reinforcement learning algorithms is to find this optimal policy.

3.1.2 Value Iteration

Once we model a problem as an MDP, it becomes clear what the agent's objective is: to maximize long term expected reward. In our setting this translates into the ad hoc agent optimally cooperating with its teammates to accomplish their shared goals. One way to calculate the optimal policy is by using Value Iteration (VI) [1]. VI requires a complete model of the environment, specifically the full transition and reward functions. Given these models, VI can calculate the optimal value function $Q^*(s, a)$ and therefore the optimal policy π^*. Value iteration relies on dynamic programming to solve for the optimal state-action values for all state-action pairs. VI initializes the state-action values arbitrarily, and then improves these estimates using an update version of the Bellman optimality equation given in Eq. 3.1. These updates are repeated iteratively until convergence, and the final calculated state-action values are guaranteed to be optimal.

While VI provably converges to the optimal policy, this convergence may take a substantial amount of time. VI has difficulties in scaling to large domains as it requires visiting each state-action over many iterations. In ad hoc teamwork scenarios, this problem is especially costly as the number of agents greatly increases the state space. In many problems, the team's state space becomes exponential in the size of the domain, with a power proportional to the number of agents, $(\#\text{positions})^{(\#\text{agents})}$. Given the symmetries of a specific problem, it is sometimes possible to reduce the number of possible states, but the scaling is still poor. For example, in initial tests into ad hoc teamwork in the pursuit domain (described in Sect. 3.2.2), VI on a 5×5 world took approximately 12 h on the University of Texas Mastodon computing cluster. In a 5×5 world, there are $25^5 \approx 1e7$ states to consider ignoring symmetries, given that there are 5 agents moving around the 25 world positions. Scaling up to a larger problem of a 20×20 world, there are $400^5 \approx 1e13$ states of the entire team. Thus, there are more than a million times more states than the 5×5 world, leading to this problem to be computationally infeasible. Due to the exponential blowup of the size of the state space, many ad hoc teamwork problems are not suitable for VI, even if the teammates' behaviors are fully known and the problem can be described as an MDP.

3.1.3 Monte Carlo Tree Search

Value Iteration is one approach for solving MDPs, and it would allow an ad hoc agent to optimally cooperate with its teammates if were it to have a complete model of its teammates and the environment. However, VI is often infeasible to run in a reasonable time and requires a complex model. Rather than calculating the exact optimal value of every state-action, it is much more computationally tractable to instead learn an approximately optimal value for relevant state-actions. When the state space is large and only small sections of it are relevant to the agent, it can be advantageous to use a sample-based approach to approximating the values of actions, such as Monte Carlo Tree Search (MCTS). Specifically, the MCTS algorithm called Upper Confidence bounds for Trees (UCT) [2] is used as a starting point for creating the primary planning algorithm used in this book.

MCTS does not require a complete model of the environment. Rather than knowing the full probability distribution of next states and rewards resulting from the transition and reward functions, MCTS only needs a model that allows sampling these next states and rewards. Furthermore, rather than treating all of the state-actions as equally likely, UCT focuses on only calculating the values for relevant state-actions. UCT does so by performing a number of playouts at each step, starting at the current state and sampling actions and the environment until the end of the episode. It then uses these playouts to estimate the values of the sampled state-action pairs. Also, it maintains a count of its visits to various state actions, and estimates the upper confidence bound of the values to balance exploration and exploitation. When selecting actions, UCT greedily chooses the action with the highest upper confidence bound.

UCT has been shown to be effective on domains with a high branching factor, such as Go [3] and large POMDPs [4], so it should be able to handle the branching factor caused by the number of agents.

In this book, UCT is modified to use eligibility traces and remove the depth index to help speed up learning. The pseudocode of the algorithm can be seen in Algorithm 3.1, with s being the current state. Similar modifications were made by Silver et al., with good success in Go [5]. In addition, work by Hester and Stone [6] show good results in a number of other reinforcement learning domains.

Algorithm 3.1 The modified version of UCT used in this book

function UCTSelect:
  inputs:
    s ▷ the current state
  outputs:
    a ▷ action selected by UCT
  params:
    γ ▷ discount factor, parameter of the MDP
    NumPlayouts ▷ number of Monte Carlo playouts to perform
    c ▷ weight given to the confidence bound
    λ ▷ eligibility trace parameter - affects amount of backup
    simulateAction(s, a) ▷ an environment model that samples next states

  for $i = 1$ to NumPlayouts **do**
    Search(s)
  return $a = \text{argmax}_a \; Q(s, a)$

function Search(s):
  $a = \text{bestAction}(s)$
  while s is not terminal **do**
    $(s', r) = \text{simulateAction}(s, a)$
    $a' = \text{bestAction}(s')$
    $e(s, a) = 1$
  ▷ Update the Q-values
    $\delta = r + \gamma Q(s', a') - Q(s, a)$
    for all s^*, a^* **do**
      $Q(s^*, a^*) = Q(s^*, a^*) + e(s^*, a^*) * \frac{\delta}{\text{visits}(s^*, a^*)}$
      $e(s^*, a^*) = \lambda e(s^*, a^*)$
    $s = s'; a = a';$

function bestAction(s):
  return $\text{argmax}_a \; Q(s, a) + c\sqrt{\frac{\ln \text{visits}(s)}{\text{visits}(s, a)}}$

3.1.4 Fitted Q Iteration

While UCT is effective for quickly computing an approximately optimal policy in an MDP, it does require a model of the MDP that permits sampling from the transition and reward functions. This model can either be given or learned given enough data. VI requires a stronger model; a model that gives the full probability distribution of next states and rewards for the transition and reward functions. However, other approaches attempt to directly learn the values of state-actions without a model of the transition function, and these approaches are called *model-free*. Model-free approaches do not require building a model of the domain which can be more tractable in hard to model domains. In addition, model-free algorithms are often computationally simpler. In complex domains, it may be difficult for ad hoc agents to compute the model of their environment and teammates, so it may be useful for the ad hoc agent to employ a model-free learning method to find a good policy for cooperating with its teammates.

In this work, our agent uses the Fitted Q Iteration (FQI) algorithm introduced by Ernst et al. [7]. Similar to Value Iteration (VI), FQI iteratively backs up rewards to improve its estimates of the values of states. Rather than looking at every state and every possible outcome from each state, FQI uses samples of these states and outcomes to approximate the values of state-action pairs. This approximation allows FQI to find solutions for complex, continuous domains. Alternative policy learning algorithms can be used, such as Q-learning [8] or policy search [9].

To collect samples of the domain, the agent first performs a number of exploratory actions. From each action, the agent stores the tuple $\langle s, a, r, s' \rangle$, where s is the original state, a is the action, r is the reward, and s' is the resulting state. An advantage of the FQI algorithm is that this data can be collected in parallel from a number of tests. At each iteration, the agent updates the following equation for each tuple

$$Q(s, a) = r + \gamma * \max_{a'} Q(s', a')$$

where $Q(s, a)$ is initialized to 0. Q is an estimate of the optimal value function, Q^*, and this estimate is iteratively improved by looping over the stored samples. To handle continuous state spaces, Q is not stored exactly in a table; instead, its value is approximated using function approximation. In this paper, the continuous state features are converted into a set of binary features using CMAC tile-coding [10, 11], and the estimate of $Q(s, a)$ is given by

$$\hat{Q}(s, a) = \sum_i w_i f_i$$

where f_i is the ith binary feature and w_i is the weight given to the feature with updates split uniformly between the active features. This approach uses a set of overlapping tilings to cover the space with binary activations [1]. The advantages of tile coding include simple computation, binary output, and good control over the generalization of the model.

3.1.5 Partially Observable Markov Decision Process

Section 3.1.1 introduced the Markov Decision Process (MDP), and Sects. 3.1.2–3.1.4 introduce methods for solving MDPs. However, not all problems can be modeled as an MDP. Therefore, we also use an extended version of the MDP known as the Partially Observable Markov Decision Process (POMDP) in our analysis. In this model, the agent cannot directly observe its true state s. Instead, it receives imperfect observations of the underlying state, $\Omega(s) = o \in O$, where O is the set of possible observations. The underlying states and transitions remain unchanged from the original MDP, as does the agent's goal of maximizing the reward. However, the agent's task is harder because it must reason about the true state.

The difficulty of solving a POMDP is bounded by the size of the δ-covering of its belief space. A belief state is the probability distribution over states that the agent may be in. The belief space is a combination of what the agent can directly observe about the world and its beliefs about the hidden state of the world. For a metric space A, a set B is a *δ-covering* if $\forall a \in A \ \exists b \in B$ such that $|a - b| < \delta$. Intuitively, a δ-covering can be thought of as a set of multi-dimensional balls with radius δ filling the space. The *covering number* is the size of the smallest δ-covering. From Theorem 5 in [12], it is known that a policy that performs within ϵ of the optimal policy for a POMDP can be found in polynomial time in terms of the size of a given δ-cover set B where $\delta = \text{poly}(\epsilon)$. This theorem shows this result for the infinite horizon, discounted rewards case, chosen because the discount factor bounds the expected total reward. However, these results extend to the finite horizon setting of the bandit problem used in Chap. 6 given that the expected total reward is bounded by the number of rounds and agents.

3.1.6 Partially Observable Monte Carlo Planning

As in an MDP, in a POMDP, the agent's goal is to maximize its long term expected reward. This task is made more difficult by the partial observability of the domain. However, there are existing algorithms for planning effective policies for these problems, and Partially Observable Monte Carlo Planning (POMCP) is one such algorithm [4]. POMCP is a Monte Carlo Tree Search (MCTS) algorithm that is an extension of the Upper Confidence bounds for Trees (UCT) algorithm, discussed in Sect. 3.1.3. While POMCP is guaranteed to find an optimal policy for a POMDP given infinite time, it loses those guarantees in situations with limited computation. Despite this lack of guarantees when computation is limited, POMCP has been shown to be effective on a number of large POMDPs, scaling far beyond existing approximate solvers.

Similar to UCT, POMCP relies on performing a number of simulations from the current state until reaching the end of the problem. In the simulations, the agent selects its actions using upper confidence bounds on its current estimates of the

available actions. However, rather than being given the true world state, POMCP has to reason about the underlying world state given the observations the agent receives. To handle this uncertainty, POMCP starts from the set of possible initial world states and performs simulations to create a tree, where each node represents the possible observations received and edges are the actions. At each node, POMCP stores counts of which underlying world states are reached that produce these observations. POMCP uses these counts to estimate the probability of a sequence of observations having come from a sequence of states. This estimate allows POMCP to quickly approximate the Bayesian update of beliefs while simultaneously calculating an estimate of the values of each action.

3.1.7 Transfer Learning

While Sects. 3.1.2 and 3.1.3 introduce methods for how the ad hoc agent can compute a policy for cooperating with its teammates given a model of its teammates, it does not specify where these models come from. An approach that we employ in this book is to learn models of past teammates, treating it as a supervised learning problem. When the ad hoc agent has a limited amount of experiences with its current teammates in addition to extensive experiences with past experiences, it may be able to learn models specific to the current teammates. Unfortunately, the limited experiences with the current teammates makes learning a new model from scratch infeasible. However, it may be able to reuse information it has learned about past teammates in addition to what it knows of its current teammates to learn a new model of its teammates. This idea leads us to transfer learning. In Transfer Learning (TL), the goal is to reuse information learned on a *source* data set to improve results on a *target* data set. For TL, only the performance on the target data matters; the source data is only used for training. Following this terminology, for ad hoc teamwork settings, we consider the current teammates to be the *target* set, and the previously observed teammates are the *source* set. In this section, we discuss three state of the art transfer learning algorithms: TrAdaBoost [13], TwoStageTrAdaBoost [14], and TrBagg [15].

TrAdaBoost [13] is a boosting-based algorithm, in which the source and target data sets are lumped together and then a model is learned via boosting. As in many boosting algorithms, data points are weighted in TrAdaBoost, so that each data point can have a different amount of influence in the learned classifiers. As in standard boosting, errors made on the target data set are handled by increasing the weights of points on which mistakes are made. On the other hand, errors on points from the source data set are treated differently; specifically, misclassified points from the source data set have their weights decreased. The intuition is to identify source data points that are distributed similarly to the target data and decrease the effect of irrelevant points.

TwoStageTrAdaBoost [14] was designed in response to problems of TrAdaBoost overfitting the training data. This problem is especially noticeable when the source data set is much larger than the target data set, causing TrAdaBoost to respond

too much to irrelevant source data points before dropping their weights sufficiently. Therefore, TwoStageTrAdaBoost first searches over a set of possible weightings of the source data points, and determines which weighting is best using cross-validation. At each weighting, it fixes the weight of the source data and performs boosting over the combined source and target data sets, only changing the weights of the target data points. These models learned for each weighting are tested via cross-validation before selecting the best weighting on which to train the final model.

While the other transfer learning algorithms described here focus on using boosting, bagging approaches have also shown promise, specifically in the form of TrBagg [15]. The TrBagg algorithm combines the source and target data sets into a single data set. From this combined set, TrBagg samples a number of smaller data sets. Then, a model is learned on each data set, and these models then undergo a filtering phase, using cross validation to determine which models are most helpful. In the filtering phase, models are sorted in order of their performance on the target data, and subsets of increasing size are tested against a fallback model, that prevents negative transfer learning. The final model outputs the median of the models that pass the filtering phase. This method has the advantage of being conceptually simple and very easy to parallelize.

There has also been some research into transferring knowledge from multiple sources. Yao and Doretto [16] introduced two transfer learning algorithms for handling multiple sources. The first, MultiSourceTrAdaBoost, uses an instance-transfer approach, reusing data from the source tasks for training the target classifier. Alternatively, TaskTrAdaBoost employs a parameter-transfer approach where it is assumed that the target data shares some parameters with some of the source data sets. Another look at transfer with multiple source sets is the work of Huang et al. [17]. They propose the SharedBoost algorithm to select the best features for prediction from a small number of source data sets for text classification. Zhuang et al. [18] investigate using autoencoders to determine a feature mapping that allows them to train multiple classifiers from the source domain and apply them effectively on the target domain. Similarly, Fang et al. [19] introduce an algorithm that determines a shared subspace among the labels for multiple sources, where each sample is given multiple labels. Then, this subspace is used to transfer knowledge to the target domain. Another approach was developed by Ge et al. [20]. The authors introduce an online algorithm that transfers knowledge from multiple sources in a principled way, achieving a no-regret guarantee compared to the offline algorithm. These algorithms provide a promising step towards effectively handling multiple sources.

3.2 Domain Descriptions

This section describes the domains that we use to evaluate approaches for cooperating in ad hoc teams. Specifically, we use three domains, which we describe in the order of simplest to most complex. These domains provide a spectrum of different tasks and interactions between agents that allow us to effectively evaluate our approach.

In addition, we also describe the teammates that are used as possible teammates in each of the domains. We use a number of hand-coded teammates to provide a spread of possible teammates that our agent may encounter. However, in ad hoc teamwork research, it is important to also use *externally-created* teammates to evaluate the various ad hoc team agents. Externally-created teammates are created by developers other than the authors and represent real agents that are created for the domain when developers do not plan for ad hoc teamwork scenarios. It is important to use externally-created teammates because their development is not biased to make them more capable of cooperating with ad hoc team agents. Therefore, they are useful for analyzing how ad hoc agents can cooperate with teammates that are not designed to work with them, which is the general case that ad hoc agents would encounter in real world scenarios.

We also describe how these domains fit into the three dimensions described in Sect. 2.3: teammate knowledge, environment knowledge, and teammate reactivity. The exact values of these dimensions vary in the different tests performed in this book. Therefore, tables of the values are specified for various settings at the end of each domain description.

3.2.1 Multi-armed Bandit Problem

To investigate ad hoc teamwork in a simple domain, we use a problem that is commonly used to model decision-making tasks, namely the multi-armed bandit problem. The name of "multi-armed bandits" is derived from the informal name of "one-armed bandit" given to slot machines. In the standard multi-armed bandit setting, an agent has to decide between a number of arms to pull, each corresponding to a slot machine with a different underlying payoff distribution. The agent's goal is to maximize its payoff by learning which arm has the highest payoff mean and repeatedly pulling this arm. However, exploring the different arms has a cost because the other arms may pay out less than the best known arm. Therefore, the central problem for the multi-armed bandit setting is to balance exploration and exploitation.

The multi-armed bandit setting is a fundamental problem in single agent reinforcement learning [1], and a bandit setting without communication has been used to study ad hoc teamwork in the past [21]. This book introduces a multiagent, multi-armed bandit problem that allows limited communication. In this problem, there are several agents that pull the arms simultaneously. After pulling an arm, each agent can broadcast a set of messages to share knowledge with its teammates, and each of these messages has a cost to send. This setting is chosen here to serve as a minimal decision-making domain that exhibits the necessary properties for investigating communication with unknown teammates.

The multi-armed bandit setting is a useful abstraction for many decision-making scenarios. For example, consider a scenario in which a number of robots are deployed to transport supplies following a disaster. These robots must repeatedly carry supplies along one of a few possible routes which vary in their speed and safety. In this setting,

selecting a route corresponds to pulling an arm. It is desirable for these robots to share their knowledge about the routes, but this communication takes time and is limited to whatever messages their teammates understand. A robot that is adept at reasoning about ad hoc teamwork should adapt to its teammates' usage of these routes and help the team select the best routes.

We formally define the bandit problem in this book as the tuple $G = (\mathbb{A}, \mathbb{C}, \mathbb{P}, R)$ where $\mathbb{A}$ is a set of two arms $\{arm_0, arm_1\}$ with Bernoulli payoff distributions, returning either 0 or 1, $\mathbb{C} = \{(c_i, cost(c_i))\}$ is a finite set of possible communications and their costs, $\mathbb{P}$ denotes the players in the problem with $|\mathbb{P}| = n + 1$ with n of the agents being a pre-designed team, and R is the number of rounds. Each round in the problem involves two phases: (1) a communication phase followed by (2) an action phase. In both phases, all agents act simultaneously. In the communication phase, each agent can broadcast a message of each type to its teammates:

- **obs**—Send the agent's last selected arm and payoff
- **mean**$_i$—Send the agent's observed mean and number of pulls for arm_i
- **suggest**$_i$—Suggest that the teammates pull arm_i

These message types are understood by all of the agents. In the action phase, each agent chooses an arm and receives a payoff. The team's goal is to maximize the sum of payoffs minus the communication costs. We use arm_* to denote the arm with the highest payoff. Note that the results in this book can be generalized to any number of fixed arms, other discrete distributions, and other message types.

As a concrete example, consider a case with four agents playing for three rounds with two Bernoulli arms with success probabilities of 0.75 and 0.25 respectively. Their actions and payoffs are shown in Table 3.1. The team state shows the combined knowledge of the team as given by the 4-tuple: $(\frac{s_0}{p_0}, \frac{s_1}{p_1}, r, sugg)$, where p_i and s_i are the number of pulls and successes for arm_i, r is the number of rounds remaining, and $sugg$ is the last suggestion. The rows specify the messages sent, arms pulled, and observed payoffs of each agent on the team. Remember that each agent can send a number of messages during each communication phase and pull one arm during the action phase. The messages are used to keep the team's knowledge of all arms synchronized and encourage the teammates to pull the arm believed to be best. At the end of the game, the team's reward is $7 - 5 \cdot cost(obs) - cost(suggestion)$ given that the team observes 7 successful pulls, sends 5 *obs* messages, and a single *suggestion* message.

3.2.1.1 Teammate Overview

Given that the teammates form an existing team, we assume that they are tightly coordinated. Specifically, this means that the team's behavior can be described as a function of the team's total number of pulls and successes of each arm. The teammates can pool this knowledge using the message types provided in the domain. The team's actions also rely on the ad hoc agent's pulls and successes that it has communicated, combining all of the team's pulls and successes as well as the ad hoc

Table 3.1 Example of play in the bandit domain

	Round 0			Round 1			Round 2		
	Comm	Action	Payoff	Comm	Action	Payoff	Comm	Action	Payoff
Team state	$(\frac{0}{0}, \frac{0}{0}, 2, \text{none})$	$(\frac{0}{0}, \frac{0}{0}, 2, \text{none})$		$(\frac{2}{2}, \frac{1}{2}, 1, \text{none})$	$(\frac{2}{2}, \frac{1}{2}, 1, sugg_0)$		$(\frac{4}{5}, \frac{1}{3}, 0, sugg_0)$	$(\frac{4}{5}, \frac{1}{3}, 0, sugg_0)$	
$agent_0$	$\{\emptyset\}$	arm_0	1	$\{obs, sugg_0\}$	arm_0	0	$\{\emptyset\}$	arm_0	1
$agent_1$	$\{\emptyset\}$	arm_1	0	$\{\emptyset\}$	arm_0	1	$\{obs\}$	arm_0	1
$agent_2$	$\{\emptyset\}$	arm_0	1	$\{obs\}$	arm_0	1	$\{obs\}$	arm_0	0
$agent_3$	$\{\emptyset\}$	arm_1	1	$\{obs\}$	arm_1	0	$\{\emptyset\}$	arm_1	0

agent's into a single estimate of the quality of each arm. While the assumption that all of the knowledge is shared via communication may not always hold, it may hold in many scenarios. This assumption simplifies the problem as representing each agent's knowledge of the numbers of pulls and successes can lead to an exponential blowup of the state space, while maintaining the team's knowledge is merely polynomial, as shown in Chap. 6.

Each teammate's behavior consists of an action function and a communication function. These functions specify the probability of the agent selecting arms or sending messages. We denote the action function of each teammate by the function *act*, where the result of *act* is a probability distribution over the agent pulling each arm. Let a_i^r be the action chosen by agent i in round r, p_k be the number of times that arm_k was pulled by the team, s_k be the number of times that arm_k returned a value of 1, $R-r$ be the number of rounds remaining, and *sugg* be the ad hoc agent's last suggestion. Then, $act(p_0, s_0, p_1, s_1, R-r, sugg, i) = \Pr(a_i^r = arm_j)$. The teammates' communication function is denoted *comm*, where the result of *comm* is the probability of sending each possible message. In particular, $comm(p_0, s_0, p_1, s_1, R-r, sugg, i) = \Pr(c_j \in C_i^r)$ where C_i^r is the set of messages broadcast by agent i in round r and $c_j \in \mathbb{C}$.

Going back to the example in Table 3.1, $agent_0$ is the ad hoc agent. The team state row summarizes the variables described for *act* and *comm* at the beginning of the specified phase. In this scenario, for the remaining agents,

$$comm(c) = \begin{cases} 1 & \text{if } c = obs \text{ and the last payoff was 1} \\ 0 & \text{otherwise} \end{cases}$$

Let us now describe the *act* function. When there have been no pulls yet, the even numbered agents select arm_0, and the odd numbered ones select arm_1. In later rounds, $agent_2$ continues exploring the arms, and the other agents act greedily. To examine one agent's decision, in round 1, $agent_1$'s action function is given by $act(2, 2, 2, 1, 1, sugg_0, 1) = arm_0$ because arm_0's observed mean is higher than that of arm_1.

3.2.1.2 Hand-Coded Teammates

We consider two parameterized types of hand-coded teammates that select arms using: (1) the ε-greedy algorithm and (2) confidence bounds in the form of UCB1 [22]. The value ε controls the exploration of the agent by specifying the chance of taking a random action. In our setting, the teammates' value of c can vary, unlike in the original UCB1, so we call the algorithm UCB(c). The value of c similarly controls exploration, but by specifying how much weight is given to the confidence bounds. The ε-greedy algorithm specifies that the teammates' action function is:

$$v_i = \frac{s_i}{p_i}$$
$$act(arm_i) = \begin{cases} 1-\varepsilon & \text{if } v_i = \text{argmax}_j\, v_j \\ \varepsilon & \text{otherwise} \end{cases} \tag{3.2}$$

The teammates using UCB(c) select actions using:

$$v_i = \frac{s_i}{p_i} + c\sqrt{\frac{\ln(p_0+p_1)}{p_i}}$$
$$act(arm_i) = \begin{cases} 1 & \text{if } v_i = \text{argmax}_j\, v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

In some of our theoretical analysis, we consider teammates where $\varepsilon = 0$ and $c = 1$. To model the effects of sending suggestions, the teammates are given a probability of following the most recent suggestion received from the ad hoc agent, using the probability $0 \leq s \leq 1$.

3.2.1.3 Externally-Created Teammates

In addition to the set of hand-coded teammates, we also consider a number of *externally-created teammates*. These agents represent a spread of possible teammates that the ad hoc agent may encounter. It is important to note that these agents are not designed for ad hoc teamwork, so there are no guarantees about how they will interact with the ad hoc agent. These agents were designed by undergraduate and graduate students as part of an assignment for a course on agent design taught by Sarit Kraus at Bar Ilan University in the fall of 2012. To prevent any bias in the creation of the agents, the students designed the entire team without considering ad hoc teamwork. These agents use the same three types of messages available to the ad hoc agent. However, not all of the students' agents employ all of the message types available, so they may ignore some messages sent by the ad hoc agent.

3.2.1.4 Dimension Analysis

Let us now describe where the multi-armed bandit domain falls on the dimensions described in Sect. 2.3. All values are calculated using the sampling-based approach specified in the same section. In our study of the bandit domain, the ad hoc agent's team knowledge (TeamK) varies based on the prior knowledge of the ad hoc agent as well as the number of rounds and arms. In the simplest setting where the ad hoc agent exactly knows its teammates' behaviors, TeamK = 1. Values for TeamK and Reactivity in a number of different scenarios in the bandit domain are summarized in Table 3.2. In the majority of our tests, the ad hoc agents knows the true payoffs of the arms. Therefore, the ad hoc agent has perfect environmental knowledge in

Table 3.2 TeamK and reactivity for various settings in the bandit domain given that the ad hoc agent believes that its teammates are drawn from {ε-greedy, UCB(c)}

Teammate type	Num arms	Num teammates	TeamK	Reactivity
Hand-coded	2	1	0.410	0.300
Hand-coded	2	2	0.391	0.410
Hand-coded	3	4	0.420	0.610
Hand-coded	7	1	0.484	0.508
Hand-coded	7	2	0.520	0.374
Externally-created	3	4	0.418	0.150

these settings, i.e. EnvK $= (1, 1)$. However, we also consider the case where the Bernoulli payoff probabilities are not known be the ad hoc agent. In these situations, EnvK $= (0.5, 0.5)$ given that the ad hoc agent's prior is uniform for the acting phase, $\text{EnvK}_{\text{act}} = (0, 0)$, and complete for the communication phase, $\text{EnvK}_{\text{comm}} = (1, 1)$.

Given the high reactivity of the teammates, it is helpful for the ad hoc agent to explicitly model its teammates' mental state so that it can cooperate with them effectively. However, the low reactivity of the externally-created teammates in the unbiased scenario motivates our tests about whether the ad hoc agent can help correct its teammates' biased knowledge of the arms as described in Chap. 7. The limited knowledge of its teammates' behaviors means that the ad hoc agent needs to quickly learn about its teammates. In addition, the high environmental knowledge allows the ad hoc agent to focus on learning about its teammates rather than learning about the domain. However, when the payoff distributions of the arms are not known by the ad hoc agent, it must balance exploring the domain as well as its teammates. The bandit domain is investigated from a theoretical standpoint in Chap. 6, and empirical results are presented in Chap. 7.

3.2.2 Pursuit Domain

While the previous section described a simple decision-making domain, this section introduces a more complex sequential decision-making domain. Specifically, this section describes the pursuit domain, also called predator-prey. The pursuit domain is a popular problem in multiagent systems literature as it requires cooperation between all of the teammates to capture the prey while remaining simple enough to evaluate approaches well [23]. There are many versions of the pursuit domain with different rules, but the pursuit domain revolves around a set of agents called predators trying to capture an agent called the prey in minimal time.

In the version of the pursuit domain used in this book, the world is a rectangular, toroidal grid, where moving off one side of the grid brings the agent back on

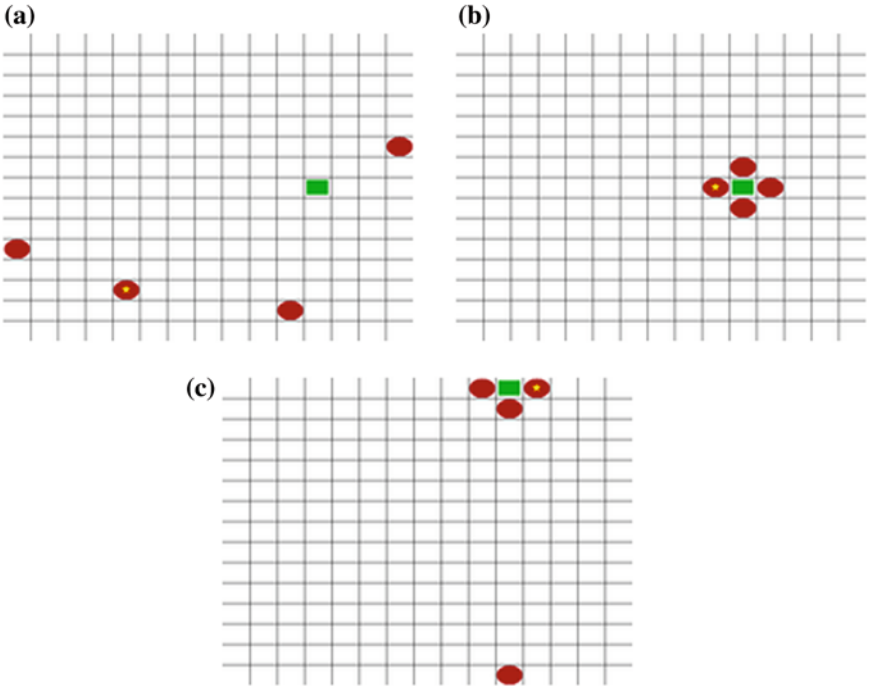

Fig. 3.2 A view of the pursuit domain, where the *rectangle* is the prey, the *ovals* are predators, and the *oval with the star* is the ad hoc predator being evaluated. **a** Random starting position. **b** A valid capture position. **c** A second valid capture position

the opposite side. Four predators attempt to capture the randomly moving prey by surrounding it on all sides in as few time steps as possible. At each time step, each agent can select to move in any of the four cardinal directions or to remain in its current position. All agents pick their actions simultaneously, and collisions are handled using priorities that are randomized at each time step. In addition, each agent is able to observe the positions of all other agents. A view of the domain is shown in Fig. 3.2, and videos of the domain can be viewed online.[1]

3.2.2.1 Hand-Coded Teammates

In order to meaningfully test the proposed ad hoc teamwork algorithms, four hand-coded predator algorithms with varying and representative properties were used. Short descriptions of these predators are below, and a full explanation of them can be found in Appendix A. The *greedy* (GR) predator moves towards the nearest open cell that neighbors the prey, ignoring its teammates' actions. On the other

[1] http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_Pursuit.

hand, the *teammate-aware* (TA) predator considers its teammates, and allows the predator that is farthest from the prey have the cell closest to it. In addition, the teammate-aware predator uses the A* path planning algorithm to select its actions while the greedy predator only considers immediate collisions. It is expected that the differences between these teammates will require the ad hoc agent to adapt and reason about how its actions will influence its teammates' actions. In addition to these two deterministic agents, two stochastic agents are used that each select an action distribution at each time step. The *greedy probabilistic* (GP) predator moves similarly to the greedy predator except that it has a chance of taking a longer path to the greedy destination. Finally, the *probabilistic destinations* (PD) predator chooses a new destination near the prey at every time step, slowly encircling the prey before converging on it.

These behaviors were chosen to provide a spread of representative behaviors. The deterministic GR predator largely ignores its teammates' actions while the deterministic TA predator tries to move out of the way of its teammates, but it also assumes that they will move out of its way when needed. It is expected that the ad hoc agent will need to cooperate differently with these two types of agents based on their reactivity (0.855 for TA teammates and 0.655 for GR teammates). In addition to these two deterministic agents, the stochastic GP and PD agents provide behaviors that are harder for the agent to quickly differentiate. Therefore, the ad hoc agent will be forced to reason about the uncertainty of its teammates' behaviors for longer. Furthermore, these behaviors are significantly different from the deterministic behaviors, and interacting with them requires reasoning about noise in future outcomes.

3.2.2.2 Externally-Created Teammates

While the set of hand-coded teammates attempts to be representative, this set is limited and possibly biased as the agents were designed by someone planning for ad hoc teamwork. Therefore, we also consider externally-created teammates to provide a broader range of agents less biased towards cooperating with agents following other behaviors. Specifically, we use two additional sets of teammates in this book, both created by undergraduate and graduate computer science students. These agents were created for an assignment in two workshops on agent design with no discussion of ad hoc teams; instead, the students were asked to create a team of predators that captured the prey as quickly as possible. The agents produced varied wildly in their approaches as well as their effectiveness. Both sets of agents come from a workshop taught by Sarit Kraus at Bar Ilan University, one taught in the spring of 2010, and the other taught in the spring of 2011. The first set of agents contains the best 12 student agents taken from the first class of 41 students, filtered by their ability to capture a randomly moving prey in a 5×5 world in less than 15 steps on average (i.e. $s_{min} = 15$ in Algorithm 2.1). This set of agents is called $\text{Student}_{\text{Selected}}$. The second set of agents, $\text{Student}_{\text{Broad}}$, comes from a second offering of the course and contains 29 agents from a class of 31 students. One student team was removed for not capturing the prey at all and the second was removed for taking excessively long times

Table 3.3 TeamK and reactivity for various settings in the pursuit domain

Teammate type	Prior knowledge	World size	TeamK	Reactivity
Hand-coded	Hand-coded	5×5	0.719	0.717
Hand-coded	Hand-coded	20×20	0.360	0.801
$\text{Student}_{\text{Selected}}$	Hand-coded	20×20	0.156	0.801
$\text{Student}_{\text{Selected}}$	Known learned set	20×20	0.318	0.801
$\text{Student}_{\text{Broad}}$	Hand-coded	20×20	0.098	0.800
$\text{Student}_{\text{Broad}}$	Known learned set	20×20	0.301	0.800
$\text{Student}_{\text{Broad}}$	Leave-one-out known learned set	20×20	0.280	0.800

to calculate their actions. $\text{Student}_{\text{Broad}}$ contains a wider range of performance than $\text{Student}_{\text{Selected}}$ as it is filtered less heavily. The better quality of agents in $\text{Student}_{\text{Broad}}$ is due to the improvements to the directions and architecture provided to the second class of students based on the lessons learned from the first offering of the course.

3.2.2.3 Dimension Analysis

Given the problem description, we can analyze how the pursuit domain is described by the dimensions introduced in Sect. 2.3. As in the bandit domain, the ad hoc agent's knowledge about its team (TeamK) varies in the different tests, as does the reactivity of the teammates. When the ad hoc agent knows its teammates' behaviors, TeamK $=$ 1. A variety of other scenarios are summarized in Table 3.3, where we vary the type of teammates as well as the prior knowledge the ad hoc agent has about its teammates. The ad hoc agent completely knows the environment dynamics, leading to EnvK $= (1, 1)$.

The high reactivity values for all teammate types implies that it is vital to understand and model the ad hoc agent's teammates. The lower values of team knowledge for the student teammates shows the importance of quickly narrowing the field of models to descriptive ones to allow for better planning. As these values are generally similar to those of the bandit domain, we expect that similar approaches should work in these two domains. This hypothesis and approaches to learning about teammates are explored empirically in Chap. 7.

3.2.3 Half Field Offense in the 2D RoboCup Simulator

While the pursuit domain provides an interesting multiagent domain for testing teamwork, it is still fairly simple compared to real world problems. In order to test the scalability of our approach, it is important to also consider more complex problems.

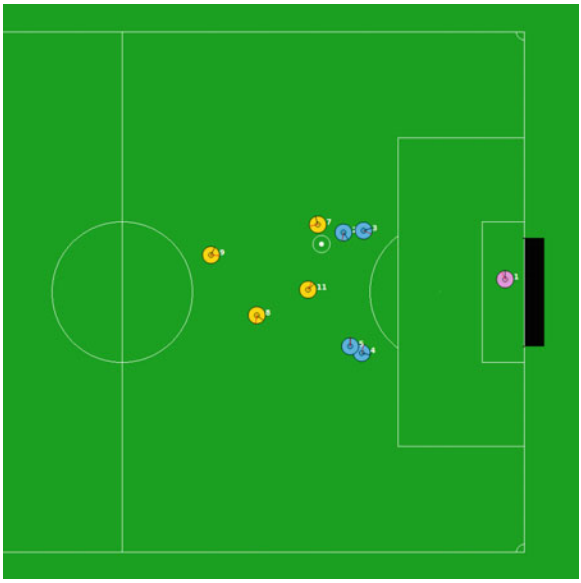

Fig. 3.3 A screenshot of half field offense in the 2D soccer simulation league. The *yellow* agent number 11 is under our control, and remaining *yellow* players are its externally created teammates. These agents are trying to score against the *blue* defenders

Therefore, we also consider a simulated robot soccer domain used in the 2D RoboCup Simulation League.

The 2D Simulation League is one of the oldest leagues in RoboCup and is therefore one of the best studied, both in competition and research. In this domain, teams of 11 autonomous agents play soccer on a simulated 2D field for two 5 min halves. The game lasts for 6,000 simulation steps, each lasting 100 ms. At each of these steps, these agents receive noisy sensory information such as their location, the location of the ball, and the locations of nearby agents. After processing this information, agents select abstract actions that describe how they move in the world, such as dashing, kicking, and turning. The 2D simulation server and the full manual that includes the perception and action models can be found online.[2] This domain is used as it provides a testbed for teamwork in a complex domain without requiring focus on areas such as computer vision and legged locomotion.

Rather than use full 10 min 11 on 11 game, this book instead uses the quicker task of half field offense introduced by Kalyanakrishnan et al. [24]. In Half Field Offense (HFO), a set of offensive agents attempt to score on a set of defensive agents, including a goalie, without letting the defense capture the ball. A view of this game is shown in Fig. 3.3, and more information and videos can be found online.[3] This task is useful as it allows for much faster evaluation of team performance than running full games as well as providing a simpler domain in which to focus on ways to improve ball control. In this book, we consider two versions of the HFO domain: (1) a *limited version* with two offensive agents and two defensive agents including the goalie and

[2] http://sourceforge.net/projects/sserver/.

[3] http://www.cs.utexas.edu/~AustinVilla/sim/halffieldoffense/.

(2) the *full version* with four offensive agents and five defensive agents including the goalie. Videos of both versions of this domain can be viewed online.[4]

If the ball leaves the offensive half of the field or the defense captures the ball, the offensive team loses. If the offensive team scores a goal, they win. In addition, if no goal is scored within 500 simulation steps (50 s), the defense wins.

At the beginning of each episode, the ball is moved to a random location within the 25 % of the offensive half closest to the midline. Let *length* be the length of the soccer pitch. Offensive players start on randomly selected vertices forming a square around the ball with edge length $0.2 \cdot length$ with an added offset uniformly randomly selected in $[0, 0.1 \cdot length]$. The goalie begins in the center of the goal, and the remaining defensive players start randomly in the back half of their defensive half. A variety of start conditions are shown in Fig. 3.4.

3.2.3.1 Externally-Created Teammates

Unlike in the multi-armed bandit and pursuit domains, we do not use hand-coded teammates as it is difficult to define what a set of representative policies might be in this domain. Instead, it is more productive to consider agents created for the RoboCup competition. It is expected that these agents represent a far better spread of possible behaviors than any hand-coded teammates, given the years of improvements implemented for the competitions.

As part of the 2D simulation league competition, teams are required to release binary versions of their agents following the competition.[5] Therefore, we use the binary releases from the 2013 competition held in Eindhoven. These agents provide an excellent source of *externally-created* teammates with which to test the possible ad hoc team agents. Specifically, we use 6 of the top 8 teams from the 2013 competition, omitting 2 as they do not support playing games faster than real time. In addition, we use the team provided in the code release by Helios [25], commonly called agent2d. Therefore, there are a total of 7 possible teams that our agent may encounter: agent2d, aut, axiom, cyrus, gliders, helios, and yushan.

In order to run some existing teams used in the RoboCup competition, it is necessary to field the entire 11 player team for the agents to behave correctly. Therefore, it is necessary to create the entire team and then constrain the additional players to stay away from play, only using the agents needed for half field offense. These additional players are moved to the other side of the field every time step. This approach may affect the players used in the HFO, but empirical tests have shown that the teams still perform well and that our ad hoc team agent can still adapt to these teams. We choose a fixed set of player numbers for the teammates, based on which player numbers tended to play offensive positions in observed play. In the limited HFO task, defensive players use the helios behavior, while in the full HFO task, they use the agent2d behavior.

[4] http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_HFO.

[5] http://www.socsim.robocup.org/files/2D/.

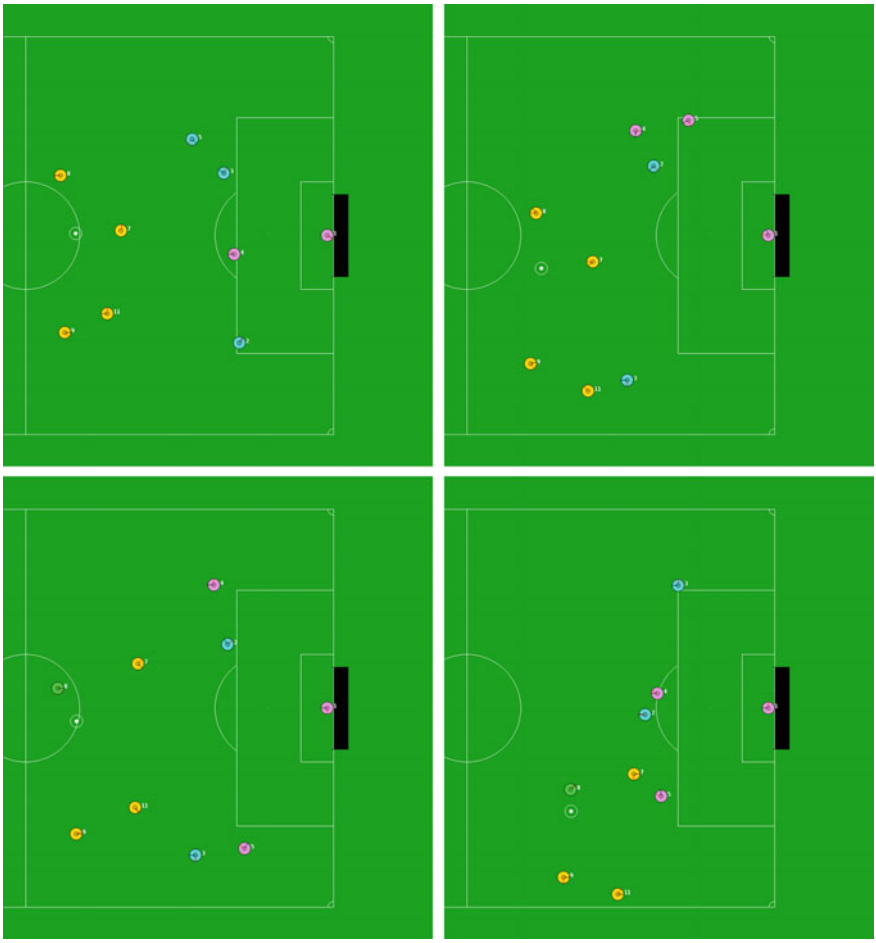

Fig. 3.4 A screenshot of a selection of random start positions in half field offense

3.2.3.2 Dimension Analysis

In order to better understand the properties of the half field offense domain and the teammates that the ad hoc agent may encounter, we can use the dimensions described in Sect. 2.3. We approximate the Jensen-Shannon divergence measure using Monte Carlo sampling. Recall from Chap. 2 that $\text{JS}(P, Q) = \frac{1}{2}(\text{KL}(P, M) + \text{KL}(Q, M))$ where $M = \frac{1}{2}(P + Q)$ and the Kullback-Leibler divergence is defined as

$$\text{KL}(P, M) = \int P(X) \log \frac{P(x)}{M(x)}$$

The Monte Carlo approximation is given by

$$\widehat{\mathrm{KL}}(P, M) = \frac{1}{n} \sum_{i}^{n} \log \frac{P(x_i)}{M(x_i)}$$

where $M(x_i) = \frac{1}{2}(P(x_i) + Q(x_i))$. As the number of samples goes to infinity, this approximation converges to the true value of KL.

Given that the actions are also continuous, we need to consider an infinite number of joint actions. In addition, the ad hoc agent does not directly observe the actions of its teammates. Therefore, we use the resulting locations of the agents as an estimate of their actions. We assume that the effects of these actions are noisy modeled with a Gaussian distribution with standard deviation of 40 and 40° for distances and angles respectively.

Applying this methodology with the calculation introduced in Sect. 2.3.1 leads us to an approximate value of 0.425 for TeamK in the limited HFO task. In this task, the ad hoc agent knows that its teammate's behavior is drawn from the set of 7 potential behaviors. In the full HFO task, this methodology calculates that TeamK ≈ 0.295.

We can similarly approximate the value of Reactivity by using the calculation introduced in Sect. 2.3.3. The 7 possible teams that the ad hoc agent may encounter have an average reactivity of Reactivity $= 0.263$ in the limited HFO task and Reactivity $= 0.507$ in the full version of the task. Given that the 2D RoboCup simulator is open source and all domain parameters are passed on to the players, the ad hoc agent completely knows the environment dynamics. In addition, note that the opponents' behaviors are known by the ad hoc agent. Therefore, EnvK $= (1, 1)$. However, it is worth noting that it is complex to model the full domain, so in our tests, the ad hoc agent does not explicitly model the HFO dynamics.

The fairly high reactivity means that the ad hoc agent can help its team and should consider how its actions affect its teammates, especially in the full HFO domain. In addition, the perfect environmental knowledge means that the agent does not need to explore the environment. On the other hand, the lower teammate knowledge means that it is helpful to explore the teammates' behaviors, especially in the full HFO domain, where the space of its teammates' behaviors is larger. Notice that these values are close to those arising from the bandit and pursuit domains. Therefore, we once again expect that a similar approach should be effective in this domain. However, the complexity of fully modeling the domain means that methods applied to the other two domains may run into issues here. Therefore, we expect that a model-free approach may be more effective, but using teammate knowledge similarly should be effective. The model-free approach is analyzed in more depth with empirical results in Chap. 7.

Table 3.4 Ranges of the dimensions for scenarios in the 3 domains used in this book

Domain	TeamK	EnvK	Reactivity
Bandit	0.391–1	(0.5,0.5)–(1,1)	0.150–0.610
Pursuit	0.156–1	(1,1)	0.717–0.801
Limited HFO	0.425	(1,1)	0.263
Full HFO	0.295	(1,1)	0.507

3.3 Chapter Summary

This chapter presents the Markov Decision Process model that we use to analyze problems in this book. This model has been well analyzed in the past, resulting in a number of effective algorithms for tackling learning how to act in domains that fit the MDP model. Therefore, this chapter reports on the learning algorithms that the remainder of the book builds upon. In addition, this chapter describes the domains in which we explore ad hoc teamwork, as well as the teammates that our ad hoc agent will encounter. Using these descriptions, we analyze each domain and set of teammates using the dimensions presented in Sect. 2.3. A summary of these values is given in Table 3.4. The values achieved through this analysis advise the approach we use to tackle these problems, which is described in Chap. 5. To better understand the problems tackled in this book, it is important to investigate how the problems relate to existing research. Thus, the following chapter describes the research relevant to this book.

References

1. Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement learning: An introduction*. Cambridge: MIT Press.
2. Kocsis, Levente, and Csaba Szepesvari. 2006. Bandit based Monte-Carlo planning. In *Prooceedings of the seventeenth European conference on machine learning (ECML)*.
3. Gelly, Sylvain, and Yizao Wang. 2006. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *Advances in neural information processing systems 19 (NIPS)*, Dec 2006.
4. Silver, David, and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems 23 (NIPS)*.
5. Silver, David, Richard S. Sutton, and Martin Müller. 2008. Sample-based learning and search with permanent and transient memories. In *Proceedings of the twenty-fifth international conference on machine learning (ICML)*.
6. Hester, Todd, and Peter Stone. 2013. TEXPLORE: Real-time sample-efficient reinforcement learning for Robots. *Machine Learning (MLJ)* 90(3): 385–429.
7. Ernst, Damien, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)* 503–556.
8. Watkins, Christopher John Cornish Hellaby. 1989. Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge, UK, May 1989.

9. Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters. 2013. A survey on policy search for Robotics. *Foundations and Trends in Robotics* 2(1–2): 1–142.
10. Albus, James S. 1971. A theory of cerebellar function. *Mathematical Biosciences* 10(12): 25–61.
11. Albus, J.S. 1975. A new approach to manipulator control cerebellar model articulation control (CMAC). *Transactions on ASME, Journal of Dynamic Systems, Measurement, and Control* 97(9): 220–227.
12. Hsu, David, Wee Sun Lee, and Nan Rong. 2007. What makes some POMDP problems easy to approximate? In *Advances in neural information processing systems 20 (NIPS)*.
13. Dai, Wenyuan, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for transfer learning. In *Proceedings of the twenty-fourth international conference on machine learning (ICML)*, 193–200.
14. Pardoe, David, and Peter Stone. 2010. Boosting for regression transfer. In *Proceedings of the twenty-seventh international conference on machine learning (ICML)*, June 2010.
15. Kamishima, T., M. Hamasaki, and S. Akaho. 2009. TrBagg: A simple transfer learning method and its application to personalization in collaborative tagging. In *Ninth IEEE international conference on data mining (ICDM)*, Dec 2009, 219–228.
16. Yao, Yi, and G. Doretto. 2010. Boosting for transfer learning with multiple sources. In *Proceedings of the conference on computer vision and pattern recognition (CVPR)*, June 2010.
17. Huang, Pipei, Gang Wang, and Shiyin Qin. 2012. Boosting for transfer learning from multiple data sources. *Pattern Recognition Letters* 33(5): 568–579.
18. Zhuang, Fuzhen, Xiaohu Cheng, SinnoJialin Pan, Wenchao Yu, Qing He, and Zhongzhi Shi. 2014. Transfer learning with multiple sources via consensus regularized autoencoders. In *Machine learning and knowledge discovery in databases*, Lecture notes in computer science, ed. Toon Calders, Floriana Esposito, Eyke Hllermeier, and Rosa Meo, vol. 8726, 417–431. Berlin: Springer.
19. Fang, Min, Yong Guo, Xiaosong Zhang, and Xiao Li. 2015. Multi-source transfer learning based on label shared subspace. *Pattern Recognition Letters* 51: 101–106.
20. Ge, Liang, Jing Gao, and Aidong Zhang. 2013. OMS-TL: A framework of online multiple source transfer learning. In *Proceedings of the 22nd ACM international conference on information & knowledge management, CIKM'13*, 2423–2428, New York, NY, USA. ACM.
21. Stone, Peter, and Sarit Kraus. 2010. To teach or not to teach? Decision making under uncertainty in ad hoc teams. In *Proceedings of the ninth international conference on autonomous agents and multiagent systems (AAMAS)*, May 2010.
22. Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning (MLJ)* 47: 235–256.
23. Stone, Peter, and Manuela Veloso. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8(3): 345–383.
24. Kalyanakrishnan, Shivaram, Yaxin Liu, and Peter Stone. 2007. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup-2006: Robot Soccer world cup X*. Lecture notes in artificial intelligence, vol. 4434, 72–85. Berlin: Springer.
25. Akiyama, Hidehisa. 2010. Agent2d base code release. http://sourceforge.jp/projects/rctools.

Chapter 4
Related Work

While the preceding sections discuss the research problem and the approach used to tackle that problem, this section focuses instead on the problem of situating this book in the literature. We first provide overviews of some areas that are closely related to ad hoc teamwork, beginning with general multiagent coordination research in Sect. 4.1 and then moving on to research into opponent modeling in Sect. 4.2. In Sect. 4.3, we discuss selected works that use the domains discussed in this book. Finally, we conclude with a more detailed discussion of current research on ad hoc teamwork in Sect. 4.4. Any works that overlap between the other areas and ad hoc teamwork are covered in the section on ad hoc teamwork.

4.1 Multiagent Coordination

There is a large body of research on coordinating multiagent teams, as multiagent teams are considered an important topic for artificial intelligence research, especially in the area of robotics. This field is too large to be surveyed completely in this book, especially as the methods employed are only tangentially related to ad hoc teamwork due to their strong assumptions. Instead, we discuss some methods that best represent the core themes of research in this area.

One important algorithm for multiagent coordination is STEAM [1], which builds upon the SharedPlans coordination algorithm [2] as well as the joint intentions framework [3]. STEAM accomplishes a shared joint task using a flexible architecture for representing team plans through a hierarchical decomposition tree. Agents keep their beliefs consistent with the rest of the team through the use of communication. In addition, agents monitor the progress of their plans, repairing these plans as conditions change. STEAM allows developers to specify general rules as well as domain specific rules, allowing it to be easily applied to new domains. STEAM has been shown to be effective for a wide array of domains including RoboCup and large-scale military simulations. STEAM represents a sophisticated approach for coordinating multiagent

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_4

teams through intelligent rules for task decomposition and shared communication protocols. While this method are very effective for organized teams, they require all of the agents to share the same rules and communication protocols. Therefore, it is difficult to apply STEAM to ad hoc teams, where we cannot assume that all agents will have this shared background.

Another approach to coordinating multiagent teams is the Generalized Partial Global Planning (GPGP) framework [4]. GPGP allows for a number of modular coordination mechanisms. Compared to STEAM, GPGP represents a more decentralized, dynamic approach for team coordination. Agents in GPGP have a set of individual plans that represent different ways that goals can be achieved. Agents begin by ignoring their teammates and then use communication to figure out the relationships between their tasks. Using these relationships, agents can build a model of tasks and create a plan that accomplishes their goals. While GPGP builds plans in a less centralized fashion than STEAM, it still requires all of the agents to share a common communication protocol as well as a similar way of representing tasks and plans. Therefore, GPGP is not directly applicable to ad hoc teams, where the agents may not know how to communicate with one another and may model tasks very differently.

STEAM and GPGP expect that agents know the team's goal as well as the tasks required to accomplish the goal. The difficulty arises from how to assign agents to different tasks and how to order these tasks. However, another line of research explores domains in which this information is not known; instead, agents must learn the tasks required to accomplish their goals. Multiagent learning is a broad field, including homogeneous and heterogeneous teams with and without communication. Importantly, multiagent learning research covers fully cooperative, fully competitive, and partially competitive settings. We discuss a representative publication in the fully cooperative setting as it most closely compares to the fully cooperative setting of ad hoc teamwork. The fully competitive setting is discussed in Sect. 4.2.

Lauer and Riedmiller's work introduces Distributed Q-learning [5] for learning in multiagent, fully cooperative settings. The authors model the problem as a multiagent MDP and adopt a model free approach. In Distributed Q-learning, each agent maintains a local policy and Q-value function that depends only on its own actions. The local Q-value function is only updated when the update leads to an improvement, ensuring that the local Q-value maintains the maximum value of the joint-actions. Using this approach, the agents' policies converge to the optimal joint-actions in deterministic domains. The distributed Q-learning approach is effective for learning intelligent team behaviors, but, like many cooperative multiagent learning algorithms, it assumes that the whole team follows the same algorithm. If agents deviate from this approach, the team's performance will suffer. Thus, distributed Q-learning is not directly applicable to ad hoc teamwork settings, where teammates' behaviors may deviate significantly from that expected by distributed Q-learning. Many other approaches take a similar high-level approach to learning in multiagent systems [6–9], and more information about multiagent reinforcement learning can be found in Busoniu et al.'s work [10].

Multiagent coordination is a large and growing area of research. While this section only presents a small number of papers, these publications represent some core approaches to the problems faced in multiagent coordination. Methods for coordinating multiagent teams largely rely on specifying standardized protocols for communication as well as shared algorithms for coordination. These approaches do not directly apply to ad hoc teams given their strong assumptions about this sharing of prior knowledge. In comparison, PLASTIC does not require any shared communication or coordination protocols and does not assume that the teammates are necessarily adapting to the ad hoc team agent.

4.2 Opponent Modeling

The work discussed in the previous section assumes that the whole team is cooperative, trying to accomplish a shared goal. Another scenario that may occur in multiagent teams is the fully competitive setting, where agents attempt to achieve mutually exclusive goals. Opponent modeling research explores this problem, explicitly modeling the other agents in the domain. While research into cooperative teams appears to be more similar to ad hoc teamwork given that agents are trying to accomplish shared goals in both settings, opponent modeling is often more similar to ad hoc teamwork. This similarity stems from the importance of understanding and reasoning about the other agents in the domain in opponent modeling, which is also necessary for robust ad hoc team agents. Rather than trying to represent the entire field of opponent modeling, this section summarizes some of the major lines of inquiry into the problem that are relevant to this book.

One such line of inquiry is theoretically motivated, exploring what can be proven about interacting with opponents. An algorithm that shows this reasoning is the AWESOME algorithm [11]. AWESOME is a learning algorithm for repeated normal form games. When it plays against opponents that use the same algorithm, the AWESOME agents will converge to playing the Nash equilibrium, the optimal behavior if all agents are rational. When playing against stationary opponents, an AWESOME agent learns to exploit them optimally. These results show that the same algorithm can exploit simple opponents while still not getting exploited by smart agents. In the same vein of theoretical analysis, Chakraborty and Stone developed the CMLeS algorithm [12]. CMLeS extends to exploiting memory-bounded teammates while retaining the convergence to the Nash equilibrium in self play. CMLeS reverts to playing a maximin strategy when playing adversaries that are not memory-bounded, retaining good payoffs in the worst case scenario.

This style of theoretical approaches to opponent modeling leads to algorithms that handle the worst case scenario of other agents very well. If the other agents in the domain are fully competitive and are intelligent, this type of algorithm ensures that our agent will not be exploited. However, in the ad hoc teamwork scenario, we can treat the other agents in the domain as teammates. Rather than expecting them to exploit any weakness our agent exhibits, we can assume that they are trying

to accomplish a shared goal. Therefore, we can reason more about the common case rather than the worst case scenario, which allows us to design algorithms that learn more quickly. This line of research in extended to the ad hoc team scenario in a more recent paper by Chakraborty and Stone [13] discussed in Sect. 4.4. Other researchers use a similar approach to theoretically analyze how to adapt to opponents' behaviors [7, 14–16].

Another exciting line of opponent modeling research is into using game theory to solve real world security problems. For example, Korzhyk et al. [17] discuss the use of the Stackelberg game model to design intelligent strategies. This work has been applied for deploying security at the LAX airport as well as scheduling the US Federal Air Marshals. In Stackelberg games, the leaders act first, and their opponents can observe these actions before responding. The solution to this problem is robust to the opponents' actions, minimizing the risk. This paper shows that in some scenarios, any Stackelberg strategy is also a Nash equilibrium strategy. Additionally, this paper shows that in other scenarios, Stackelberg and Nash strategies differ. This line of research shows that game theoretic approaches can be applied to real world problems with great effect, minimizing the resources required to protect a resource while maximizing its safety. These assumptions differ from that of ad hoc teamwork in that other agents are assumed to be opponents rather than teammates. In addition, the authors generally assume that opponents are intelligent and observe the agent's actions (e.g. [18–20]), though they relax that assumption in this work [17]. In ad hoc teamwork, there is no guarantee that the other agents are as intelligent as the ad hoc agent, so expecting them to optimally adapt to the ad hoc agent's actions is not reliable.

One more avenue of research that combines theoretical analysis with empirical results is in the area of computer poker. For example, Bard et al. [21] look at how to adapt to an opponent's strategy in heads-up limit Texas hold'em poker. The authors approximate the Nash equilibrium strategy through the use of counterfactual regret (CFR) [22], which limits the amount that their agent can be exploited. In order to exploit weaknesses in other players, it is possible to compute the best response strategy to their strategies. However, if the opponents' strategies are not completely known or if the opponents adapt, this best response strategy can in turn be exploited. Therefore, it is desirable to limit the amount of adaptation that is performed given the limited information about the opponents. Rather than explicitly modeling the opponent's behavior, Bard et al. use an implicit model that summarizes its opponents with a portfolio of counter strategies that are computed offline. These counter strategies are limited best responses, adapting to the opponents, but not allowing this adaptation to add too much exploitability. Then, their agent selects between these strategies online, treating this selection as a bandit problem. To aid in this selection, their agent uses variance reduction to estimate the effectiveness of the strategies and prunes the number of strategies in the portfolio. PLASTIC is related to this approach. Instead of computing the Nash equilibrium, PLASTIC can plan knowing that the other agents are trying to cooperate. Rather than limiting the best response to teammates, PLASTIC can use the full best response given that it does not need to

worry about its teammates exploiting it. In addition, we improve over using bandit selection algorithms, instead using updates based on Bayes' rule.

Opponent modeling is closely related to ad hoc teamwork. Both areas revolve around modeling and adapting to other agents in the domain. However, opponent modeling expects that other agents are intelligent opponents that are trying to exploit our agent. This assumption requires algorithms to focus on bounding the worst case scenario and never make mistakes. Ad hoc teamwork is more forgiving, given that the other agents are assumed to be teammates, so exploring more and learning more quickly is considered safe.

4.3 Experimental Domains

In addition to seeing how this book relates to work on multiagent teams and opponent modeling, it is also helpful to see how it relates to prior research on the domains used in the book. The three domains explored in this book are well studied domains in research into artificial intelligence. This sections gives a brief overview of the most relevant research on these domains.

4.3.1 Multi-armed Bandits

The first domain used in this book is the multi-armed bandit (MAB) problem, which has been studied extensively [23]. The bandit problem is interesting due to its simplicity, while still modeling the trade-off between exploration and exploitation. Many decision making problems can be modeled as bandit problems, leading it to be an area of continued research. It is a well studied problem in economics in addition to its presence in the artificial intelligence literature. While the vast majority of research in this area focuses on the single agent setting, several variations have been considered in which there are multiple agents that can observe the actions or outcomes of each other. We discuss a selection of these multiagent bandit settings here.

One multiagent version of the bandit domain was investigated by Keller and Rady [24]. In this scenario, there are two arms: a predictable arm that returns a small positive payoff and a risky arm that distributes lump-sum payoffs according to a Poisson distribution. In this work, Keller and Rady construct an asymmetric equilibrium in which the agents take turns pulling the risky arm. Their experiments explore giving agents rewards for pulling the risky arm, finding that having too high of a reward may decrease average payoffs in some cases. The focus of this work is in designing the right incentives to encourage the agents to explore the riskier option. This work differs from ad hoc teamwork in that the behaviors of both teammates follow fixed policies and the focus is on how the problem designer can modify the incentives to achieve good team performance.

Another line of research into multiagent teams in the bandit domain is the work by Aoyagi [25]. Aoyagi focuses on a two-armed bandit problem with multiple players that can only observe the actions of other players rather than the outcomes of these actions. This information reveals some information about the other players' beliefs about the payoffs of the arms. Under some restrictions of the arms' payoff distributions, he proves that all players will settle on the same arm. This work shows that agents can coordinate without explicit communication, reaching the same conclusions about the effectiveness of different actions. This work differs from the work in this book in that the agents in Aoyagi's formulation are not teammates sharing rewards. In addition, we explore how explicit communication can help the team's performance.

Explicit communication in the bandit domain is investigated by Goldman et al. [26]. In this work, the authors consider agents that are learning to communicate and explore how to handle issues such as misinterpretation of messages. They introduce a theoretical framework for analyzing the problem where agents learn to communicate and maximize system performance simultaneously. Then, the authors go on to show that solving the problem optimally is often intractable. Despite this difficulty, the authors propose an approach that allows agents to converge on a common language and empirically show the effectiveness of this approach. Given that they assume very little about the meaning of messages, their approach needs to learn over a long period of time compared to PLASTIC in our bandit domain. This faster adaption is enabled by the assumption that all agents share a common language for communicating, though the ad hoc agent does not necessarily know how its messages will be used by its teammates.

In summary, the bandit domain is a well studied problem for investigating decision making tasks. The majority of work in the bandit domain considers the single agent case, but there is some work on the multiagent setting, e.g. [27–29]. However, these works differ from ours in whether the authors control the whole team's behavior and how communication is used. There has also been prior research into ad hoc teamwork in the bandit domain, which is discussed in Sect. 4.4.

4.3.2 Pursuit Domain

The second domain explored in this book is the pursuit domain. Isaacs performed seminal research on pursuit and evasion [30], and the problem was further explored by Benda et al. [31]. Benda et al. explore varying the predators' ability to communicate, even considering a central strategy, but with communication carrying a cost. In this setting, communication may be limited or may lower the amount of computation time taken per step. Many variants of the pursuit domain have been investigated as it remains a useful domain for evaluating teamwork algorithms. This section only covers a sampling of relevant research, but more research into the range of research possibilities in the pursuit domain is explained in the survey paper by Stone and Veloso [32].

Most previous research focused on developing coordinated predators before deploying them, rather than learning to adapt to unseen teammates. Some of the versions of the pursuit domain are increasingly difficult, due to noise and partial observability. One recent example of this type of research is the work by Undeger and Polat [33]. The authors consider a version of the pursuit problem with a continuous, partially observable environment that contains obstacles. In this work, the prey intelligently plans its actions to avoid capture, requiring more intelligent coordination by the predators. To address this problem, Undeger and Polat introduce the MAPS algorithm, which uses two coordination strategies to position the predators in locations to cover possible escape directions of the prey. This work represents a foray into the coordination required in complex versions of the pursuit domain. In this book, we consider a simpler version of the pursuit domain, but do not assume that the team shares a common coordination algorithm.

Another investigation of the pursuit domain is that of Ishiwaka et al. [34], which considers heterogeneous agents in a continuous state-action world with partial, noisy observations. In this work, the authors investigate how the predators can learn online using Q-learning. The predators attempt to predict the locations of the other agents as well as the movement of the prey. Similar to ad hoc teamwork, not all of the agents use the same behavior. The predators start as homogeneous agents, but diverge during the learning process, specializing in parts of the task. This work presents one approach to incorporating agents with differing action policies that learn to adapt to each other. Compared to this book, this learning takes place over a much longer time, though it considers a more complex version of the pursuit domain. In addition, while the agents learn different policies, they do employ the same learning algorithm.

One work that investigates when agents have differing amounts of knowledge about the domain was performed by Chakraborty and Sen [35]. These authors examine a pursuit scenario in which experienced agents attempt to teach novice predators. This work requires that the agents share a known training protocol, where the expert selects example situations to train and teach the novice agent. This framework allows the novice agents to have different learning algorithms and knowledge representations from that of the experts. While this work investigates how an agent can learn to adapt to its teammates, it is not directly applicable to ad hoc teamwork due to its reliance on a shared training protocol.

In summary, the pursuit domain is commonly used to explore multiagent coordination. Some of this work focuses on scaling these coordination algorithms to more complex versions of the pursuit domain. Other work explores teams with heterogeneous behaviors or how to teach and learn from other teammates. While research on the pursuit domain pushes multiagent research in exciting directions, it largely requires teams to share some kind of coordination protocols, which are not available in ad hoc teamwork scenarios.

4.3.3 RoboCup

The final domain used to evaluate PLASTIC in this book is the 2D simulation league from RoboCup. RoboCup is an annual robotics competition where teams from around the world compete on a number of tasks. It serves as a common domain to test many artificial intelligence algorithms and pushes research in robotics. RoboCup pushes research in a number of areas, including walking [36–39], positioning [40, 41], and vision [42, 43]. Instead of discussing all research from RoboCup, we discuss the most relevant papers on the multiagent aspects of it, specifically looking at designing teamwork, analyzing teamwork, and modeling opponents in RoboCup.

One early exploration of learning in the RoboCup simulation domain was performed by Stone [44]. This book describes a flexible team structure as well as a novel learning approach called layered learning. Furthermore, it introduces a new multiagent reinforcement learning algorithm and describes the resulting complete team. This work shows the complexity of the domain as well as the novel research motivated by RoboCup. In particular, it introduces the concept of a "locker-room agreement." Locker-room agreements are pre-determined multiagent protocols that define the flexible teamwork structure and the inter-agent communication protocols. However, this work relies on the entire team sharing this locker-room agreement, which cannot be assumed in ad hoc teamwork.

Another aspect of research in RoboCup is how to characterize the teams' behaviors in these complex multiagent systems. Almeida et al. [45] explore this problem in the RoboCup 2D simulated robotic soccer league, using logs of play as their input information. The authors explore the complexity of team's behaviors as well as discovering guidelines for creating new plans for teamwork. This work represents how one might characterize a team's behaviors, similar to how an ad hoc agent may wish to understand its teammates' behaviors. However, the proposed approach requires a substantial amount of observations of the team, which is not usually available in ad hoc teamwork scenarios.

One set of online adaptations to other agents is from the Small Size League (SSL) of RoboCup. Biswas et al. [46] explore how to plan about opponents' strategies. Their approach attempts to pull opponents out of position, leaving openings that their strategy can then exploit. In addition, they detect potential threats based on the positions of opponents and adapt to defend these threats. These online adaptations show that agents can adapt to other agents in the domain on the fly, in just a single game. While this general approach could be applied to other multiagent settings, the current version of it relies on strong assumptions about the domain. For example, this work tries to encourage specific opponents to mark their players, pulling them out of position for the planned play.

RoboCup encourages a substantial amount of research into multiagent systems in complex domains. The majority of this research focuses on coordinated teams of agents and is thus not directly applicable to ad hoc teamwork.

4.4 Ad Hoc Teamwork

While previous sections explore different aspects of multiagent research, the most relevant research to this book is in the area of ad hoc teamwork. Ad hoc teamwork has been explored under various names and methodologies over recent years. However, the problem formulation and evaluation framework used in this book was proposed by Stone et al. [47] in a 2010 AAAI challenge paper. This paper has encouraged the growth of the field, leading to an increase in the amount of research exploring this exciting area. The remainder of this section identifies and summarizes work in this field.

The types of problems that these works address are summarized in Table 4.1. We evaluated each of the works on a number of axes. The first axis is whether they control an agent's actions, i.e. whether the output of the ad hoc teamwork algorithm is direct control decisions (a policy) for one or more agent's behavior on the team. Almost all algorithms surveyed do directly output a policy. The notable exception along this axis is Liemhetcharat and Veloso's work [48] where the task is to select a team instead of control an agent acting on a team. Then, we look at whether these approaches consider multiple teammates followed by whether these teammates are unknown to the ad hoc agent prior to the beginning of the interaction. We continue by looking at whether these papers evaluate their approaches in a domain more complex than the bandit domain or matrix games. Next, we evaluate whether these approaches are generally applicable or specific to the domain studied. Additionally, we consider the speed that the ad hoc agent adapts: whether it can learn over the course of a handful of interactions. Finally, we consider whether the approaches can automatically reuse knowledge learned about previous teammates. As shown in the table, this book is the first to address all of these issues using a single algorithm.

We report the current state of research into ad hoc teamwork in Sect. 4.4.1 before exploring one area of research in more depth in Sect. 4.4.2, namely that of the drop-in player challenge held at RoboCup. Then, Sect. 4.4.3 discusses other models that can be used to formalize problems including ad hoc teamwork scenarios. Finally, we describe where these domains lie on the dimensions of ad hoc teamwork problems in Sect. 4.4.4.

4.4.1 Survey of Ad Hoc Teamwork

A significant portion of research on ad hoc teamwork takes a theoretical approach to the problem. These approaches focus on simple settings such as the bandit domain or matrix games and try to prove the optimality of their approach under certain conditions. Other researchers focus on empirical approaches, showing that their algorithms apply to ad hoc teamwork problems in practice. Let us first consider theoretical contributions coming from analysis of ad hoc teamwork in the bandit domain.

Table 4.1 This table shows existing research into ad hoc teamwork and what types of problems they address

Paper	Control agent	Multiple teammates	Unknown teammates	Evaluated in a complex domain	Generally applicable	Adapt quickly	Automatically reuse knowledge
Stone and Kraus [49]	**Yes**	No	No	No	No	**Yes**	No
Barrett and Stone [50]	**Yes**	No	No	No	No	**Yes**	No
Brafman and Tennenholtz [51]	**Yes**	No	No	No	No	No	No
Stone et al. [52]	**Yes**	No	No	No	No	**Yes**	No
Agmon and Stone [53]	**Yes**	**Yes**	No	No	No	**Yes**	No
Agmon et al. [54]	**Yes**	**Yes**	Partially	No	No	**Yes**	No
Chakraborty and Stone [13]	**Yes**	No	**Yes**	No	No	No	No
Hao et al. [55]	**Yes**	**Yes**	No	No	No	No	No
Wu et al. [56]	**Yes**	No	**Yes**	No	Partially	No	No
Albrecht and Ramamoorthy [57]	**Yes**	**Yes**	Partially	**Yes**	**Yes**	**Yes**	No
Wray and Thompson [58]	**Yes**	**Yes**	No	**Yes**	No	**Yes**	No
Bowling and McCracken [59]	**Yes**	**Yes**	Partially	**Yes**	No	No	No
Jones et al. [60]	**Yes**	**Yes**	No	**Yes**	Partially	**Yes**	No
Su et al. [61]	**Yes**	**Yes**	No	**Yes**	No	**Yes**	No
Han et al. [62]	**Yes**	**Yes**	No	**Yes**	No	**Yes**	No
Genter et al. [63]	**Yes**	**Yes**	No	**Yes**	No	**Yes**	No
Genter and Stone [64]	**Yes**	**Yes**	No	**Yes**	No	**Yes**	No
Liemhetcharat and Veloso [48]	No	**Yes**	**Yes**	**Yes**	**Yes**	No	No
This book	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

None of these papers address the combination of problem types that we address in this book

One example of the theoretical approach using the bandit domain is Stone and Kraus's research [49]. In this work, the authors consider a multiagent version of the setting with a knowledgeable agent attempts to teach its novice teammate. The knowledgeable agent knows the true distributions of the arms while the novice agent starts with no initial knowledge about the arms. The authors only control the knowledgeable agent, which is called the teacher. This work differs from existing teaching literature in that the teacher is embedded in the domain, so its teaching actions have an explicit cost. They consider the case where the agents know the number of time steps remaining and have undiscounted rewards. We expanded this work to the discounted reward setting with infinite pulls [50]. This line of research proves that the ad hoc agent acting as teacher can optimally lead its teammate to achieve the best team payoff. These papers differ from the work in this book in that they assume that the novice teammate's policy is known and their results are only directly applicable to the bandit domain, while we consider domains that are too large to be proven to be tractable in a similar fashion.

While the bandit domain allows for multiagent research, the majority of work on it is single agent and therefore not focusing on the ad hoc teamwork problem. A more common domain for looking at interactions between agents is in matrix games, where agents act simultaneously and receive rewards. This domain allows for multiagent interactions, but remains simple enough for theoretical analysis.

An early paper that looks into ad hoc teamwork in matrix games was that of Brafman and Tennenholz [51]. In their paper, they investigate agents performing a repeated joint task, where one agent attempts to teach a novice agent. The authors could only affect the ad hoc agent, i.e. the agent acting as a teacher. In this work, they use a game-theoretic framework and only consider teammates that either maximize their expected utility or use reinforcement learning. Overall, they consider a number of strategies for the agent to play, including a reinforcement learning agent as well as some well established hand-coded policies. In different settings, they find different approaches work better, and conclude that effective agents in this domain need to reason about how to punish and reward their teammates to affect their actions. This book explores more complex domains than those used in this work, considers unknown teammates, and uses more types of teammates.

Building on this idea, Stone et al. [52] investigate ad hoc teamwork in matrix games with a theoretical focus. They explore how an ad hoc agent should cooperate with a best response teammate while maximizing the team's shared rewards. Best response agents choose the action that gives the highest payoff assuming that its teammates continue playing their last observed action. In this work, the ad hoc agent knows the payoff matrix as well as the teammate's behavior, so the difficulty is to plan the optimal path to lead the best response teammate to the best payoff. This work was expanded by Agmon and Stone [53] to include more than one teammate. Agmon and Stone show that the best payoff is not always reachable when the team is larger than two agents, but they come up with a way of describing the optimal team payoff as the optimal steady cycle and show how to lead a team to that cycle. This work was further expanded to the case where the teammates' behaviors are not fully known [54], instead assuming that the ad hoc agent knows that its teammates

are using a behavior from a known set. A common approach to uncertainty in matrix games is to assume the worst case scenario, which in this setting corresponds to the teammates following the weakest behavior from the set. However, this paper shows that incorrectly assuming that the teammates are following the weakest behavior can lead to long term poor payoffs even after more information is gathered. Therefore, the authors describe an algorithm, REACT, for balancing the potential costs of different assumptions about the teammates' behaviors and show that REACT empirically performs well on a large number of matrices. This line of research differs from that of this book due to its focus on theoretical analysis, which limits the work to the simple matrix game setting. In addition, this book considers a wider range of possible teammate behaviors as well as cases where the ad hoc agent has less knowledge of its teammates.

Stone et al. [52] and Agmon and Stone [53] both assume that the teammates' behaviors are known, though Agmon et al. [54] relax this assumption, instead assuming that the teammates' behaviors are drawn from a known set. Chakraborty and Stone [13] further relax this knowledge of the teammates' behaviors in ad hoc teamwork scenarios in matrix games. This work extends earlier work by Chakraborty and Stone [12] for opponent modeling, as discussed in Sect. 4.2. The authors propose a new algorithm, LCM, that tries to achieve optimal performance with teammates that use a limited features derived from the history. With other teammates, LCM ensures that the team receives the security value of the matrix game. LCM does this by determining which features best explain its teammate's behavior, and, if no set of features explains its behavior, LCM reverts to playing the safety strategy. This approach performs optimally with some teammates, but this form of learning takes substantially longer than PLASTIC. Unlike PLASTIC, LCM does guarantee a safety value with any teammates, but in practice this safety value is often low compared to what the team could receive and ensuring a safety value in more complex tasks requiring coordination is often impossible.

While Chakraborty and Stone's work [13] performs very well, it requires complex calculations and reasoning. An approach to simplifying ad hoc teamwork in matrix games was explored by Hao et al. [55]. They explore how agents can coordinate under the networked social learning framework. This research proposed two types of learners: one that models its teammates as part of the environment and one that considers the joint actions with its teammates. The authors explore how the neighborhood size of the agents affects these two approaches. This work focuses on how the other agents affect the learning process. As opposed to this book, this work considers agents with a shared learning approach, instead investigating the effects of network topology on learning.

While matrix games serve as a good testbed for looking at interactions between agents in ad hoc teamwork scenarios, they are limited to stateless interactions. Wu et al. [56] scale theoretical analysis of ad hoc teamwork to some more complex, though still theoretically tractable, domains. In this work, the authors investigate ad hoc teamwork with few assumptions about the behaviors of the teammates. Their ad hoc agent plans using MCTS and uses biased adaptive play to predict the actions of teammates. Biased adaptive play can be used to estimate the policies of teammates

from their previous actions. They test their agent on three domains: cooperative box pushing, meeting in a 3×3 grid, and multi-channel broadcast. They consider the case where the ad hoc agent knows the environment, but not its teammates. These teammates are referred to as unknown teammates (UTM), and two types of teammates are used in each domain: UTM-1 agents that follow a fixed set of actions and UTM-2 agents that try to play the optimal behavior but have partial observations. Their work shows that their approach can adapt to these teammates to accomplish their tasks. While this work explores several domains, all of the domains used are fairly simple. Additionally, their ad hoc agent is given a large amount of expert knowledge and the set of possible teammates is limited compared to this book.

Another approach to scaling ad hoc teamwork beyond only matrix games is the work of Albrecht and Ramamoorthy [57, 65], though they also consider matrix games in their work. In this work, they consider the case where the ad hoc agent is given a set of possible types of its teammates and introduce a new formal model to represent this problem. In their setting, the problem is for the ad hoc agent to determine the type of its teammates. Their approach (HBA) combines the idea of Bayesian Nash equilibria with the Bellman optimality equation. HBA maintains the probability of each of the provided teammate types and maximizes its expected payoffs according to the Bellman principle. In later research [66], Albrecht and Ramamoorthy explore the convergence bounds of HBA. Specifically, they prove convergence bounds when an ad hoc agent knows its teammates are drawn from a known set and consider how accurate the expert-provided types need to be for HBA to solve its task. This line of research is closely related to that of this book, but differs in some notable ways. Given the similarity of HBA and PLASTIC, we expect that much of their analysis could be generalized to PLASTIC. However, we also consider scenarios where the ad hoc agent is not provided with expert knowledge about possible teammates and uses PLASTIC to reuse knowledge learned from previous teammates. Furthermore, we show that PLASTIC scales to more complex domains than those used to evaluate HBA.

While theoretical analysis of problems can create exciting new algorithms for ad hoc teamwork, an important question is how well these algorithms fare in more complex empirical analyses. One line of research that considers a more complex ad hoc teamwork scenarios is that of Wray and Thompson [58]. They look at a problem with a continuous state space with limited observability. In this work, the authors investigate a version of the pursuit domain, where a number of predators that have to intercept an unknown number of prey without any explicit communication. The key problem in the domain is for predators to avoid trying to capture the same prey. Their solution employs fictitious play to dynamically assign predators to prey despite the limited information. While the authors focus on a more complex version of pursuit than is used in this book, their agents know more about their teammates. Though they do not know the prey that these predators will try to capture, they know how the other predators decide which prey to capture and employ this information to prevent the predators from attempting to capture the same prey. In addition, their work only explores how to match predators to prey as opposed to the more complex coordination required in this book.

While Wray and Thompson [58] consider a more complex domain, the necessary coordination between agents is limited to ensuring that the predators do not pursue the same prey. A work that looks at more complex coordination in ad hoc teams is that of Bowling and McCracken [59]. In the domain of robot soccer, Bowling and McCracken measure the performance of a few ad hoc agents, where each ad hoc agent is given a playbook that differs from that of its teammates. In this paper, a play refers to a team plan that specifies when the play applies, termination conditions, and roles for the players. In this domain, the teammates implicitly assign the ad hoc agent a role, and then react to it as they would any teammate. The ad hoc agent analyzes which plays work best over hundreds of games and predicts the roles that its teammates will play. This work explores a very similar setting to that used in this work, though it learns over a significantly longer time scale. However, they focus on agents that are similarly designed but have a different playbook, rather than completely unknown teammates. In addition, their approach relies on having a playbook of possible plays that specifies roles for all agents on the team. In many domains, agents may not have such a playbook, so this approach cannot be directly applied to these domains.

Jones et al. [60] also consider robotic ad hoc teams, but they expand their analysis to heterogeneous robots. The authors explore ad hoc teams operating in the treasure hunt domain and implement their algorithms on real heterogeneous robots searching new environments for treasure. The authors focus on how agents can allocate roles amongst a team in a decentralized fashion. However, they assume that the agents share a communication protocol that they use to bid on different roles in an auction as well as a shared coordination protocol for how to assign tasks given this communication. This book explores scenarios in which these shared protocols do not exist, as they may not always be present in ad hoc teamwork scenarios.

Another work that considers how to allocate roles or tasks on a team in a decentralized fashion is that of Xing Su et al. [61]. The authors explore ad hoc teamwork in disaster response scenarios. They look at how to allocate tasks under spatial and communication constraints by having the agents elect leaders in a decentralized system. This approach allows the agents to limit communication costs while maintaining good allocations and allows the agents to adapt dynamically during task execution. Unlike this book, they assume that the agents share a communication and coordination protocol and focus on how a team can adapt without starting with any structure to the team.

While the previous works mainly focus on small teams of agents, there is also research into how to affect large teams of agents. Specifically, researchers have investigated how to use a small number of agents under their control (ad hoc agents) to affect the behavior of flocks of agents, such as flocks of birds or fish. The ad hoc agents encourage the team to reach a specified goal. This work spans some of the space between theoretical and empirical approaches. An early paper in this area was written by Han et al. [62], prior to Stone et al.'s formulation of ad hoc teamwork. This work focused on adding a "shill" agent to the flock, that corresponds to the ad hoc team agent in our terminology. This agent was designed by the authors and attempts to move the flock in a desired direction. Agents in the flock are based on the Boid model [67]. in which each agent chooses its current heading by averaging

the heading of its neighbors located within some radius. Further research on this line includes the work of Genter et al. [63] and Genter and Stone [64]. This research proves bounds on the number of actions required to control the flock's behavior. In addition, they provide an algorithm that empirically outperforms other methods by using short-term lookahead planning. The authors expand the problem to consider multiple ad hoc agents. This line of research differs from this book in that it focuses on scenarios in which the teammates' behaviors are known, rather than needing to learn about teammates. The difficulty of their problem is in planning the optimal behavior rather than balancing exploring the teammates' behaviors and exploiting the current knowledge.

While the previous works all consider how an ad hoc agent should act to improve the performance of its team, another consideration is how to choose agents to form a new team given a much larger set of possible agents. Liemhetcharat and Veloso [48] explore this idea, selecting which agents will form a new ad hoc team. Given that different agents are better at performing different roles on the team, it is important to select agents that fill the roles in a beneficial way. In addition, there are synergies in the team, where some pairs of agents work better with each other than with other agents. These complexities lead to interesting questions into how to select teammates from this set of agents. The authors come up with a novel representation of this problem, called a synergy graph, and show how to learn this graph. While it also investigates ad hoc teamwork, this research focuses on the problem of selecting agents for an ad hoc team rather than the question explored in this book, how an agent should act on the ad hoc team.

In summary, this section presented a survey of the research on ad hoc teamwork that is relevant to this book. Table 4.1 gives an overview of these works, describing where they fall on a number of axes. Table 4.1 shows that none of these works address all of the problems tackled in this book. A large amount of these works focus on simple domains and provide theoretical analyses. In addition, a substantial number of them assume that they know their teammates or share some communication or coordination protocols, but these works are still ad hoc teamwork because not all agents are designed by the same developers and the provided protocols are decentralized. However, these works do not consider the ad hoc teamwork problems investigated in this book, where the teammates may be completely unknown prior to the coordination. Finally, this book is the only work that we are aware of that learns about previous teammates and reuses this knowledge to quickly adapt to new teammates.

4.4.2 Drop-In Player Challenge

One especially relevant area of research into ad hoc teamwork is the drop-in player challenge that was held in the 2013 and 2014 RoboCup competitions in the standard platform league, the 3D simulation league, and the 2D simulation league [68]. The

Standard Platform League (SPL) involves teams of 5 humanoid Nao robots.[1] These robots play on a color-coded field that has no unique markers, and robots need to figure out the locations of important objects using two cameras mounted in the head. The 3D simulation league uses robot models based on the Nao robots in a physics-based simulation called SimSpark.[2] Games in this league involve teams of 11 robots, where robots receive noisy observations of objects on the field. This information allows developers to concentrate on higher level team coordination than in SPL, while still retaining the complexity of humanoid robot control. The 2D simulation league is described in Sect. 3.2.3 and involves teams of 11 robots on a simulated 2D pitch. These robots receive noisy observations and act using higher level movement commands, allowing teams to concentrate on team behaviors more than in the 3D simulation league.

The drop-in player challenge involved having players from different teams attempt to combine to form a coherent team without any prior coordination. These teams then played other teams created in the same fashion, and the different players were evaluated based on their contributions to the team. All leagues used the scoring differential between teams containing each agent to measure their contributions, which matches the evaluation paradigm presented in Sect. 2.2. The standard platform league refined these scores by additionally including scores from human judges, which rated agents based on their *subjective perceived* teamwork performance. Players were given a common format for communicating basic information such as their location and that of the ball.

In the 2013 SPL competition, teams did not release any reports about their behavior, so we can only report about the behaviors of the team in which we were directly involved, UT Austin Villa. UT Austin Villa employed the same basic strategy as in the main competition, where agents bid on chasing the ball and the other players are assigned to field positions based on their current locations. In the drop-in setting, our agent estimated the bids of its teammates and assumed that its teammates would move to the "assigned" positions. While this approach worked to some degree, its effectiveness was limited by other agents' problems with communication and due to its strong assumptions.

The following year, in the 2014 competition, UT Austin Villa changed its behavior to use a potential-based positioning system. Robots tried to avoid being too close to their teammates while being attracted to the ball and a defensive position between the ball and their own goal. The chaser was decided based on communicated preferences as well as robots' positions relative to the ball. This approach worked adequately, but was limited by issues with the low level skills.

In the 2014 SPL competition, teams were required to compete in the drop-in challenge and also to write a short description of their behaviors.[3] The descriptions indicate that most teams played a behavior similar to their behavior in the main

[1] http://www.aldebaran.com.

[2] http://simspark.sourceforge.net.

[3] https://tzi.de/spl/bin/view/Website/DropinStrat2014.

competition. Most teams listen to the other players and let the player closest to the ball play a chasing role and otherwise play a field role. These roles vary based on the team, either playing a defensive role to block shots or playing upfield to wait for a pass. Often these roles were selected statically, so any adaptation was performed by the developers choosing which role their robot should play or based on dynamically avoiding other players as obstacles. The B-Human team did additionally track the trustworthiness of its teammates and used this information when deciding whether to chase the ball. The majority of teams did not specially design behaviors and strategies for the drop-in player challenge; there is still a substantial amount of work to be explored in this domain.

For the RoboCup 2D simulation league in 2013, UT Austin Villa altered its role assignment for the drop-in player challenge. Specifically, it used a dynamic role assignment system [69] that minimizes the time for all agents to reach their target positions while avoiding collisions. Given that teams communicated their positions, this approach allowed UT Austin Villa's agents to adapt to their teammates online, based on their movements.

For the 2014 RoboCup 3D simulation drop-in player challenges, UT Austin Villa used a simple role assignment scheme. If the agent was closest to the ball, it played as chaser. Otherwise, it took up a position two meters behind the ball as this position was found to be especially important [69]. In addition, the UT Austin Villa player tracks the trustworthiness of other players' information. To do this, the UT Austin Villa player listens to the locations they report for themselves and the ball. When it can observe these players or the ball, the UT Austin Villa players compares its observed positions to the communicated positions, updating the trustworthiness of the players' based on the difference between these values. When their trustworthiness drops below a threshold, the UT Austin Villa player ignores their messages.

RoboCup's drop-in player challenges serve as an exciting testbed for ad hoc team research. However, current approaches in these challenges are still fairly simple due to the relative youth of the challenge. Therefore, there is space to explore more advanced techniques. This book presents one such technique, and we hypothesize that PLASTIC could be applied successfully to these problems, but leave testing that hypothesis for future work. We did not apply PLASTIC to this setting yet because both PLASTIC–Model and PLASTIC–Policy require either some models or policies of possible teammates, which can be learned or given by an expert. In the SPL, not enough information has been stored to learn about existing teammates, and there is no release of full agents, either via full source code releases or binary releases. In addition, it is difficult to hand-code possible models or policies for teammates' behaviors given the complexity of the domain and the amount of changes to teams' strategies between competitions where their behaviors can be observed. The experiments in this book show that using approximate hand-coded models works well in some scenarios, but applying this approach to the drop-in player challenges remains future work.

4.4.3 Alternative Models

While we model the ad hoc teamwork problem as either an MDP or a POMDP in this book, there are alternative ways of modeling the problem. One such way is as an Interactive POMDP (I-POMDP) [70]. I-POMDPs model adversarial interactions of agents by examining what an agent believes about the other agents and these agents' beliefs. The graphical counterparts of I-POMDPs are known as Interactive Dynamic Influence Diagrams (I-DIDs) [71]. I-DIDs provide concise representations of the beliefs about other agents and allow for nesting I-DIDs to represent these beliefs. Both I-POMDPs and I-DIDs could be used to model the problems studied in this book. However, both have issues with the potential exponential blowup of beliefs as the size of the problem grows. While work has been performed to increase the efficiency of algorithms for these models [72–74], they remain computationally intractable for the size of problems studied in this book.

An alternative approach to modeling multiagent problems is the Network of Influence Diagrams (NID) model [75]. This model represents an agent's mental models as graphical structures. NIDs are representationally equivalent to Bayesian games but are more compact. While NIDs perform well on some problems, they also run into scalability issues as the size of the problem grows.

4.4.4 Dimension Analysis

For this related work in ad hoc teamwork, it may be helpful or informative to consider where these problems fall on the dimensions described in Sect. 2.3. We would like to calculate the exact values of the dimensions for each of the domains as we did in Sect. 3.2, but this calculation requires more information about the exact formulations of the domains and teammates than is typically available in the publications that are available to us. Therefore, we instead give some rough estimates of where these problems lie on the dimensions. The three dimensions we consider are team knowledge, environment knowledge, and the reactivity of teammates.

We begin by discussing the team knowledge (TeamK) of ad hoc agents in these domains, which shows how much the ad hoc agent knows about its teammates prior to cooperating with them. As shown in Table 4.1, the majority of the related research considers cases where the teammates are known, so TeamK is 1 or close to 1. Notable exceptions of this include Liemhetcharat and Veloso's work [48] as well as Wu et al.'s work [56] which consider completely unknown teammates, where TeamK is 0. Also, Agmon et al. [54] and Albrecht and Ramamoorthy [57] assume that their teammates are drawn from a known set, so TeamK is between 0 and 1; we estimate that TeamK lies in the range [0.3, 0.7] for these works. Additionally, Bowling and McCracken [59] consider situations with teammates that do not share a codebook with the ad hoc agent. We estimate that TeamK is fairly high in these settings, in the range [0.6, 0.8], given that effective soccer plays are similar compared to random movement of teammates.

From this analysis, we can see that most existing works focus on situations where the teammates are fairly well known; only a few consider scenarios where the teammates are initially unknown.

Let us now consider the environmental knowledge (EnvK) of these domains, where the environmental knowledge explains how much the ad hoc agent knows about the transition and reward dynamics of the domain before beginning. The majority of the works in this section assume that the ad hoc agent has full knowledge of the domain, so EnvK $= (1, 1)$ for these settings. While there may be noise in these domains, the ad hoc agent is expected to know the level of noise and therefore know the true distribution of next states. Exceptions to this include Jones et al. [60], Liemhetcharat and Veloso [48], and Hao et al. [55] where the ad hoc agent does not initially know the reward function and may have limited knowledge of the transition function. We estimate that the knowledge of the transition function lies in [0.7, 1.0] for these works, and the reward knowledge lies in [0, 0.5]. This analysis suggests that research into ad hoc teamwork has not focused on learning about the domain. Instead, agents are assumed to know the domain and instead focus on learning about teammates and planning how to cooperate with them.

The reactivity of the teammates in these domains (Reactivity) covers a large spread of values. All of the domains assume that the teammates are at least partially reactive to the ad hoc agents, or it would not be worth considering the problem as a multiagent setting. This reactivity varies significantly based on the domain. When ideal actions are fairly obvious to teammates, interactions with the ad hoc agent are unlikely to change the teammates' behaviors, leading Reactivity to be close to 0. On the opposite end of the spectrum, when the teammates have high uncertainty about the best actions ahead of time, the ad hoc agent's actions can significantly affect their actions, leading to values of Reactivity close to 1. Our analysis of the bandit domain in Sect. 3.2.1.4 shows how much the reactivity of teammates can vary in a single domain. Given that so much research assumes that ad hoc agents know their teammates and the domain well, the majority of focus has been on how to plan to cooperate effectively with teammates. Exploring planning in ad hoc teamwork encourages researchers to investigate settings with varying amounts of teammate reactivity, as this dimension is the most influential on planning.

While calculating the exact values for each of the three dimensions (TeamK, EnvK, and Reactivity) for each domain studied in the related work would be useful, it is impossible to calculate these values without complete knowledge of the domain and teammates. Even so, these rough estimates of the dimensions for these problems lead to some interesting conclusions. Specifically, existing research has done a good job of exploring how to plan to cooperate with teammates, covering the gamut of teammate reactivity. On the other hand, ad hoc team research has focused largely on problems with high team knowledge and high environmental knowledge, with less work exploring how agents can learn about their teammates and the domain. Future work in ad hoc teamwork should address this gap and explore settings in which the ad hoc agent needs to learn more about its teammates and the domain. In real world scenarios, robots will need to be constantly adapting to their changing environments as well as new teammates they may encounter. Therefore, it is important for ad hoc

teamwork research to explore these settings, where agents must reason about the tradeoff between exploring the domain, exploring their teammates, and exploiting their current knowledge. We discuss this problem in more depth in the future work presented in Sect. 8.4.

4.5 Summary

While there is a large amount of work related to ad hoc teamwork, this book brings many new ideas to the table. Rather than requiring shared communication and coordination protocols like past research on multiagent teams, this work describes agents that can cooperate without these shared protocols. Compared to opponent modeling, this work creates agents that adapt more quickly to other agents at the price of making the stronger assumption that they are cooperating towards a shared goal. With respect to ad hoc teamwork, this book moves ad hoc teamwork to an empirical setting and tackles more complex problems than most studied previously. However, the main differentiating factor of this work is that it considers how to adapt to unknown teammates in a non-domain specific way that can be applied generally.

References

1. Tambe, Milind. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)* 7: 81–124.
2. Grosz, B., and S. Kraus. 1996. Collaborative plans for complex group actions. *Artificial Intelligence (AIJ)* 86: 269–368.
3. Levesque, Hector J, Philip R Cohen, and José HT Nunes. 1990. On acting together. In *Proceedings of the eighth conference on artificial intelligence (AAAI)*, vol. 90, 94–99.
4. Decker, Keith S, and Victor R. Lesser. 1995. Designing a family of coordination algorithms. In *International conference on multi-agent systems (ICMAS)*, 73–80, June 1995.
5. Lauer, Martin, and Martin Riedmiller. 2000. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the seventeenth international conference on machine learning (ICML)*, 535–542. Morgan Kaufmann.
6. Kapetanakis, Spiros, and Daniel Kudenko. 2002. Reinforcement learning of coordination in cooperative multi-agent systems. In *American association for artificial intelligence eighteenth national conference on artificial intelligence*, 326–331, Menlo Park, CA. : American Association for Artificial Intelligence.
7. Fischer, Felix, Michael, Rovatsos, and Gerhard Weiss. 2004. Hierarchical reinforcement learning in communication-mediated multiagent coordination. In *Proceedings of the third international conference on autonomous agents and multiagent systems (AAMAS)*, 1334–1335, Washington, DC, 2004. IEEE Computer Society.
8. Weiss, Gerhard. 1995. Distributed reinforcement learning. In *The Biology and technology of intelligent autonomous agents*, vol. 144, ed. Luc Steels, 415–428., NATO ASI Series Berlin: Springer.
9. Tan, Ming. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning (ICML)*, vol. 337. Amherst, MA, 1993.

10. Busoniu, L., R. Babuska, and B. De Schutter. 2008. A comprehensive survey of multiagent reinforcement learning. In *IEEE transactions on systems, man, and cybernetics, Part C: Applications and reviews*, 38(2): 156–172.
11. Conitzer, Vincent, and Tuomas Sandholm. 2007. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. Machine Learning (MLJ), 67, May 2007.
12. Chakraborty, Doran, and Peter Stone. 2010. Convergence, targeted optimality and safety in multiagent learning. In *Proceedings of the twenty-seventh international conference on machine learning (ICML)*, June 2010.
13. Chakraborty, Doran, and Peter Stone. 2013. Cooperating with a markovian ad hoc teammate. In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*, May 2013.
14. Bowling, Michael. 2005. Convergence and no-regret in multiagent learning. In *Advances in neural information processing systems (NIPS)* 18, 209–216. MIT Press.
15. Bowling, Michael, and Manuela Veloso. 2002. Multiagent learning using a variable learning rate. *Artificial Intelligence (AIJ)* 136(2): 215–250.
16. Junling, Hu, and Michael P. Wellman. 2003. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research (JMLR)* 4: 1039–1069. December.
17. Korzhyk, Dmytro, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. 2011. Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research (JAIR)*, 41(2):297–327, May 2011.
18. Jiang, Albert Xin, Zhengyu Yin, Chao Zhang, Milind Tambe, and Sarit Kraus. 2013. Game-theoretic randomization for security patrolling with dynamic execution uncertainty. In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*, 207–214, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.
19. Shieh, Eric, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. 2012. PROTECT: A deployed game theoretic system to protect the ports of the United States. In *Proceedings of the eleventh international conference on autonomous agents and multiagent systems (AAMAS)*, 13–20, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.
20. Yang, Rong , Benjamin Ford, Milind Tambe, and Andrew Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *Proceedings of the thirteenth international conference on autonomous agents and multiagent systems (AAMAS)*, 453–460, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.
21. Bard, Nolan, Michael Johanson, Neil Burch, and Michael Bowling. 2013. Online implicit agent modelling. In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*, 255–262.
22. Zinkevich, Martin, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *Advances in neural information processing systems 20 (NIPS)*, 905–912.
23. Bergemann, D., and J. Valimaki. 2006. *Bandit problems*. Technical report Cowles Foundation Discussion Paper.
24. Keller, Godfrey, and Sven Rady. 2009. Strategic experimentation with poisson bandits. Technical report, Free University of Berlin, Humboldt University of Berlin, University of Bonn, University of Mannheim, University of Munich, 2009. Discussion p. 260.
25. Aoyagi, Masaki. 1998. Mutual observability and the convergence of actions in a multi-person two-armed bandit model. *Journal of Economic Theory* 82: 405–424.
26. Goldman, Claudia V, Martin Allen, and Shlomo Zilberstein. 2007. Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 15(1).
27. Kalathil, D, N. Nayyar, and R. Jain. 2012. Multi-player multi-armed bandits: Decentralized learning with IID rewards. In *50th Annual Allerton conference on communication, control, and computing (Allerton)*, 853–860, Oct 2012.

28. Liu Keqin, Qing Zhao, and B. Krishnamachari. 2010. Decentralized multi-armed bandit with imperfect observations. In *48th Annual Allerton conference on communication, control, and computing (Allerton)*, 1669–1674, Sept 2010.
29. Stranders, Ruben, Long Tran-Thanh, Francesco M. Delle Fave, Alex Rogers, and Nicholas R. Jennings. 2012. DCOPs and bandits: Exploration and exploitation in decentralised coordination. In *Proceedings of the eleventh international conference on autonomous agents and multiagent systems (AAMAS)*, 289–296, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.
30. Isaacs, Rufus. 1965. *Differential games: a mathematical theory with applications to warfare and pursuit*. Control and Optimization: Dover Publications.
31. Benda, M, V. Jagannathan, and R. Dodhiawala. 1986. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, July 1986.
32. Stone, Peter, and Manuela Veloso. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8(3): 345–383.
33. Undeger, Cagatay, and Faruk Polat. 2010. Multi-agent real-time pursuit. *Autonomous Agents and Multi-Agent Systems (JAAMAS)* 21(1): 69–107.
34. Ishiwaka, Yuko, Takamasa Sato, and Yukinori Kakazu. 2003. An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. *Robotics and Autonomous Systems* 43(4): 245–256.
35. Chakraborty, Doran, and Sandip Sen. 2006. Teaching new teammates. In *Proceedings of the fifth international conference on autonomous agents and multiagent systems (AAMAS)*, 691–693.
36. Behnke, S. 2006. Online trajectory generation for omnidirectional biped walking. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, 1597–1603, May 2006.
37. Farchy Alon, Samuel Barrett, Patrick MacAlpine, and Peter Stone. 2013. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of 12th international conference on autonomous agents and multiagent systems (AAMAS)*, May 2013.
38. Graf, Colin, and Thomas Röfer. 2010. A closed-loop 3D-LIPM gait for the robocup standard platform league humanoid. In *Proceedings of the fifth workshop on humanoid soccer robots in conjunction with the 2010 IEEE-RAS international conference on humanoid robots*, ed. Enrico Pagello, Changjiu Zhou, Sven Behnke, Emanuele Menegatti, Thomas Röfer, and Peter Stone. TN, USA: Nashville.
39. MacAlpine, Patrick, Samuel Barrett, Daniel Urieli, Victor Vu, and Peter Stone. 2012. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proceedings of the twenty-sixth AAAI conference on artificial intelligence (AAAI)*, July 2012.
40. Akiyama, Hidehisa, and Itsuki Noda. 2008. Multi-agent positioning mechanism in the dynamic environment. In *RoboCup 2007: Robot soccer world cup XI*, vol. 5001, ed. Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, 377–384., Lecture Notes in Computer Science Berlin: Springer.
41. MacAlpine, Patrick, Eric Price, and Peter Stone. 2014. SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning. In *AAMAS autonomous robots and multirobot systems workshop* (ARMS 2014), May 2014.
42. Bruce, James, Tucker Balch, and Manuela Veloso. 2000. Fast and inexpensive color image segmentation for interactive robots. In *proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*, vol. 3, 2061–2066.
43. Härtl, Alexander, Ubbo Visser, Thomas Röfer, and Robust and efficient object recognition for a humanoid soccer robot. In RoboCup, 2013. *Robot Soccer World Cup XVII*. Lecture Notes in Artificial Intelligence: Springer.
44. Stone Peter. 2000. *Layered learning in multiagent systems: A winning approach to robotic soccer*. MIT Press.
45. Almeida, Fernando, Pedro Henriques Abreu, Nuno Lau, and LusPaulo Reis. 2013. An automatic approach to extract goal plans from soccer simulated matches. *Soft Computing* 17(5): 835–848.

46. Biswas, Joydeep, Juan P. Mendoza, Danny Zhu, Benjamin Choi, Steven Klee, and Manuela Veloso. 2014. Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team. *Proceedings of the thirteenth international conference on autonomous agents and multiagent systems (AAMAS)*, January 2014.
47. Stone, Peter, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the twenty-fourth conference on artificial intelligence (AAAI)*, July 2010.
48. Liemhetcharat, Somchaya, and Manuela Veloso. 2014. Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents. *Artificial Intelligence (AIJ)* 208: 41–65.
49. Stone, Peter, and Sarit Kraus. 2010. To teach or not to teach? Decision making under uncertainty in ad hoc teams. In *Proceedings of the ninth international conference on autonomous agents and multiagent systems (AAMAS)*, May 2010.
50. Barrett, Samuel, and Peter Stone. 2011. Ad hoc teamwork modeled with multi-armed bandits: An extension to discounted infinite rewards. In *AAMAS workshop on adaptive learning agents workshop (ALA)*, May 2011.
51. Brafman, Ronen I., and Moshe Tennenholtz. 1996. On partially controlled multi-agent systems. *Journal of Artificial Intelligence Research (JAIR)* 4: 477–507.
52. Stone, Peter, Gal A. Kaminka, and Jeffrey S. Rosenschein. 2010. Leading a best-response teammate in an ad hoc team. In *AAMAS workshop on agent mediated electronic commerce (AMEC)*. November 2010.
53. Agmon, Noa, and Peter Stone. 2012. Leading ad hoc agents in joint action settings with multiple teammates. In *proceedings of the eleventh international conference on autonomous agents and multiagent systems (AAMAS)*, June 2012.
54. Agmon, Noa, Samuel Barrett, and Peter Stone. 2014. Modeling uncertainty in leading ad hoc teams. In *Poceedings of the thirteenth international conference on autonomous agents and multiagent systems (AAMAS)*, May 2014.
55. Hao, Jianye, Dongping Huang, Yi Cai, and Ho-Fung Leung. 2014. Reinforcement social learning of coordination in networked cooperative multiagent systems. In *AAAI workshop on multiagent interaction without prior coordination (MIPC 2014)*, July 2014.
56. Wu, Feng, Shlomo Zilberstein, and Xiaoping Chen. 2011. Online planning for ad hoc autonomous agent teams. In *The 22nd international joint conference on artificial intelligence (IJCAI)*.
57. Albrecht, S.V, and S. Ramamoorthy. 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems (extended abstract). In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*, St. Paul, Minnesota, USA, May 2013.
58. Wray, Kyle, and Benjamin Thompson. 2014. An application of multiagent learning in highly dynamic environments. In *AAAI workshop on multiagent interaction without prior coordination (MIPC 2014)*, July 2014.
59. Bowling, Michael, and Peter McCracken. 2005. Coordination and adaptation in impromptu teams. In *Proceedings of the twentieth conference on artificial intelligence (AAAI)*, 53–58.
60. Jones, Edward, Brett Browning, M. Bernardine Dias, Brenna Argall, Maria Manuela Veloso, and Anthony (Tony) Stentz. 2006. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, 570–575, May 2006.
61. Su, Xing, Minjie Zhang, Dayong Ye, and Quan Bai. 2014. A dynamic coordination approach for task allocation in disaster environments under spatial and communicational constraints. In *AAAI workshop on multiagent interaction without Prior coordination (MIPC 2014)*, July 2014.
62. Han, Jing, Ming Li, and Lei Guo. 2006. Soft control on collective behavior of a group of autonomous agents by a shill agent. *Journal of Systems Science and Complexity* 19: 54–62.
63. Genter, Katie, Noa Agmon, and Peter Stone. 2013. Ad hoc teamwork for leading a flock. In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*, May 2013.

64. Genter, Katie, and Peter Stone. 2014. Influencing a flock via ad hoc teamwork. In *Proceedings of the ninth international conference on swarm intelligence (ANTS)*, September 2014.
65. Albrecht, S.V, and S. Ramamoorthy. 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. Technical report, School of Informatics, The University of Edinburgh, United Kingdom, February 2013.
66. Albrecht, S.V, and S. Ramamoorthy. 2014. On convergence and optimality of best-response learning with policy types in multiagent systems. In *Proceedings of the 30th conference on uncertainty in artificial intelligence (UAI)*, Quebec City, Canada, July 2014.
67. Reynolds. Craig W. 1987. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on computer Graphics and interactive techniques, SIGGRAPH* '87, 25–34, New York.
68. MacAlpine, Patrick, Katie Genter, Samuel Barrett, and Peter Stone. 2014. The RoboCup 2013 drop-in player challenges: Experiments in ad hoc teamwork. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*, September 2014.
69. MacAlpine, Patrick, Francisco Barrera, and Peter Stone. 2013. Positioning to win: A dynamic role assignment and formation positioning system. In *RoboCup-2012: Robot soccer world cup XVI*. Berlin: Springer.
70. Gmytrasiewicz, Piotr J., and Prashant Doshi. 2005. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR)* 24(1): 49–79.
71. Doshi, Prashant, and Yifeng Zeng. 2009. Improved approximation of interactive dynamic influence diagrams using discriminative model updates. In *Proceedings of the eighth international conference on autonomous agents and multiagent systems (AAMAS)*.
72. Sonu, Ekhlas, and Prashant Doshi. 2012. Generalized and bounded policy iteration for finitely-nested interactive POMDPs: Scaling up. In *Proceedings of the eleventh international conference on autonomous agents and multiagent systems (AAMAS)*, 1039–1048, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.
73. Zeng, Yifeng, and Prashant Doshi. 2012. Exploiting model equivalences for solving interactive dynamic influence diagrams. *Journal of Artificial Intelligence Research (JAIR)* 43(1): 211–255.
74. Zeng, Yifeng, Yingke Chen, and Prashant Doshi. 2011. Approximating model equivalence in interactive dynamic influence diagrams using top k policy paths. *IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology* 2: 208–211.
75. Gal, Y., and A. Pfeffer. 2008. Network of influence diagrams: Reasoning about agents' beliefs and decision-making processes. *Journal of Artificial Intelligence Research (JAIR)* 33: 109–147.

Chapter 5
The PLASTIC Algorithms

This chapter introduces the Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC) algorithms that enable an ad hoc team agent to cooperate with a variety of different teammates. One might think that the most appropriate thing for an ad hoc team agent to do is to "fit in" with its team by following the same behavior as its teammates. However, if the teammates' behaviors are suboptimal, this approach will limit how much the ad hoc agent can help its team. Therefore, in this book, we adopt the approach of learning about different teammates and deciding how to act by leveraging this knowledge. This approach allows an ad hoc agent to reason about how well its knowledge of past teammates predicts its current teammates' actions as well as to convert this knowledge into the actions it needs to take to accomplish its goals. If the knowledge of prior teammates accurately predicts the current teammates and the ad hoc agent is given enough time to plan, this approach will lead to optimal performance of the ad hoc agent, helping its team achieve the best possible outcome. Note that this may not be the optimal performance of any team, but it is optimal for the ad hoc agent given that the behaviors of its teammates are fixed.

5.1 Overview

A visual overview of the PLASTIC is given in Fig. 5.1. The short summary of the approach is that the ad hoc agent either learns about a set of prior teammates or is given some hand-coded information about possible teammates. Then, the agent uses this prior knowledge to select its actions and updates its beliefs about its teammates by observing their reactions to its behavior.

This chapter contains material from three publications: [1–4]. Note that some of Sect. 5.2 is joint work with Sarit Kraus and Avi Rosenfeld in addition to Peter Stone [3].

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_5

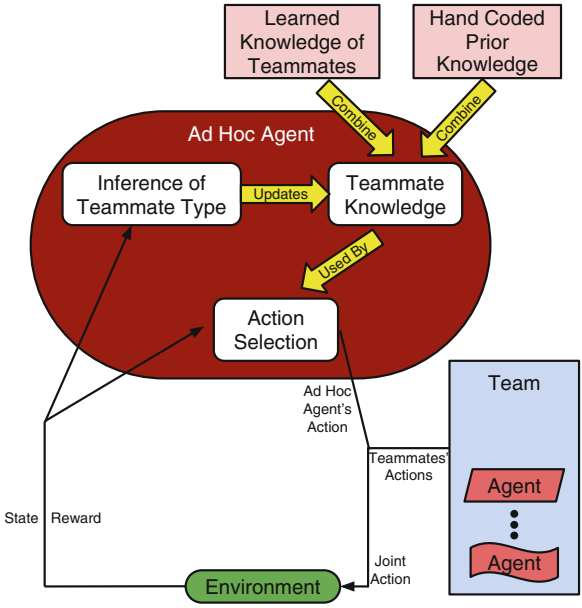

Fig. 5.1 Overview of using PLASTIC to cooperate with unknown teammates

In this book, this general approach is realized in two algorithms. One algorithm, PLASTIC–Model, focuses on a model-based approach. In this approach, the ad hoc agent learns models of its past teammates, selects which models best predict its current teammates, and then uses these models to plan how to act in order to cooperate with the teammates. The second algorithm is called PLASTIC–Policy and uses a model-free approach. In this variant, the ad hoc agent learns a policy to cooperate with each of its past teammates, selects which policies best match how to cooperate with its current teammates, and then selects actions using these policies. These two algorithms are described in the remainder of the chapter. This general approach is specified in Algorithm 5.1. The subroutines LearnAboutPriorTeammate, SelectAction, and UpdateBeliefs are described for each of the two algorithms in the following section.

As shown in Algorithm 5.1, PLASTIC begins by initializing its knowledge using the provided prior knowledge and what it has learned about previous teammates in Lines 2–5. LearnAboutPriorTeammate is defined differently for the two variants, but in both algorithms it learns information about the prior teammate, encoding the knowledge to be used in the SelectAction subroutine. Lines 6–10 show how PLASTIC selects the agent's actions. PLASTIC updates its beliefs over the teammate models or policies by observing their actions and using the UpdateBeliefs function implemented in the two variants.

Algorithm 5.1 Pseudocode of PLASTIC

```
1: function PLASTIC:
       inputs:
         PriorTeammates                ▷ past teammates the agent has encountered
         HandCodedKnowledge            ▷ prior knowledge coded by hand
         BehaviorPrior                 ▷ prior distribution over the prior knowledge
   ▷ initialize knowledge using information from prior teammates
2:     PriorKnowledge = HandCodedKnowledge
3:     for t ∈ PriorTeammates do
4:         PriorKnowledge = PriorKnowledge ∪ {LearnAboutPriorTeammate(t)}
5:     BehaviorDistr = BehaviorPrior(PriorKnowledge)          ▷ initialize beliefs

   ▷ act in the domain
6:     Initialize s
7:     while s is not terminal do
8:         a = SelectAction(BehaviorDistr, s)
9:         Take action a and observe r, s′
10:        BehaviorDistr = UpdateBeliefs(BehaviorDistr, s, a)
```

5.2 PLASTIC–Model

When an agent has a good model of its *environment*, it can use this model to plan good actions using a limited number of interactions with the environment. For an ad hoc agent to plan, it also needs to model its *teammates*; therefore, it is useful for the ad hoc agent to build models of its teammates' behaviors. Given that learning new models online takes many samples, it is useful to reuse information learned from past teammates. This section describes PLASTIC–Model, a variant of the PLASTIC approach that learns models of prior teammates and selects which models best predict its current teammates. An overview of this approach is given in Fig. 5.2 and the specification of the LearnAboutPriorTeammate, SelectAction, and UpdateBeliefs functions are given in Algorithm 5.2. These functions are described in depth in the remainder of this section.

5.2.1 Model Selection

In Algorithm 5.2, it is also necessary to select from a set of possible teammate models using SelectAction. Performing the simulations for the Monte Carlo rollouts or other planners requires that the ad hoc agent has a model of how its teammates behave. If there is a (presumably correct or approximately correct) single model for this behavior, the planning is straightforward. On the other hand, if the ad hoc agent is given several possible models to choose from, the problem is more difficult. Assuming that the ad hoc agent starts with some prior belief distribution over

Algorithm 5.2 Instantiation of functions from Algorithm 5.1 for PLASTIC–Model.

function UpdateBeliefs:
  inputs:
    BehaviorDistr ▷ probability distr. over possible teammate behaviors
    s ▷ the current environment state
    a ▷ previously chosen action
  outputs:
    BehaviorDistr ▷ updated probability distr.
  params:
    η ▷ bounds the maximum allowed loss
  for $m \in$ BehaviorDistr **do**
    loss $= 1 - P(a|m, s)$
    BehaviorDistr$(m) * = (1 - \eta$loss$)$
  Normalize BehaviorDistr
  return BehaviorDistr

function SelectAction:
  inputs:
    BehaviorDistr ▷ probability distr. over possible teammate behaviors
    s ▷ the current environment state
  outputs:
    a ▷ the best action for the agent to take
  params:
    p ▷ an MDP planner that selects actions, such as UCT
  ▷ simulateAction is derived from the known environment model and
  ▷ sampling from BehaviorDistr (the teammate behavior distribution)
  $a = p(s)$
  return a

function LearnAboutPriorTeammate:
  inputs:
    t ▷ the prior teammate
  outputs:
    m ▷ model of the teammate's behavior
  params:
    learnClassifier ▷ supervised learning algorithm
  Data $= \emptyset$
  repeat
    Collect s, a for t
    Data = Data $\cup \{(s, a)\}$
  m = learnClassifier(Data)
  return m

which model correctly reflects its teammates' behaviors, the ad hoc agent can update these beliefs by observing its teammates. Specifically, it can update the models using Bayes theorem:

$$P(\text{model}|\text{actions}) = \frac{P(\text{actions}|\text{model}) * P(\text{model})}{P(\text{actions})}$$

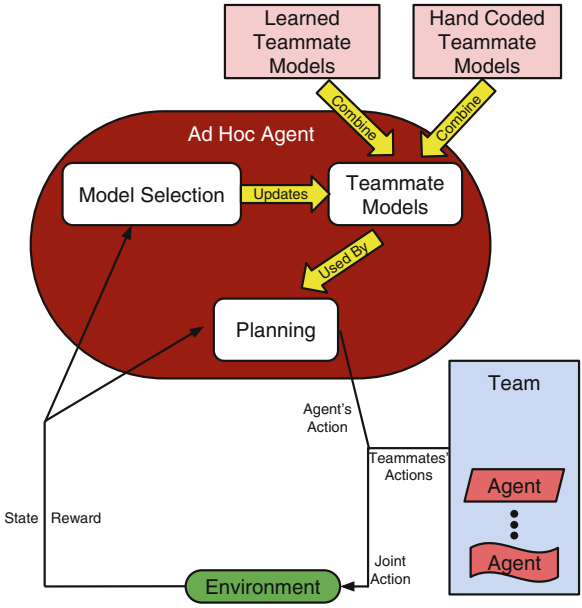

Fig. 5.2 Overview of using the model-based approach of PLASTIC–Model to cooperate with unknown teammates

If the correct model is in the given set of models, then the ad hoc agent's beliefs will converge to this model or a set of models that are not differentiable from this model.

On the other hand, if the correct model is not in the set, using Bayes rule may drop a good model's posterior probability to 0 for a single wrong prediction. This update may punish generally well-performing models that make a single mistake, while leaving poor models that predict nearly randomly. Therefore, it may be advantageous to update the probabilities more conservatively. Research in regret minimization has shown that updating model probabilities using the polynomial weights algorithm is near optimal if examples are chosen adversarially [5]. Since it is expected that the ad hoc agent's models are not perfect, the agent updates its beliefs using polynomial weights:

$$\text{loss} = 1 - P(\text{actions}|\text{model})$$
$$P(\text{model}|\text{actions}) \propto (1 - \eta * \text{loss}) * P(\text{model})$$

where $\eta \leq 0.5$ is a parameter that bounds the maximum loss, where higher values converge more quickly. This scheme ensures that good models are not prematurely removed, but it does reduce the rate of convergence. In practice, this scheme performs very well as the observed examples of the teammates may be arbitrarily unrepresentative of the agent's overall decision function.

5.2.2 Planning

This section describes the SelectAction function in Algorithm 5.2. When an ad hoc agent has a model of both the environment and its teammates, it can use this model to plan about the effects of its actions and how it should adapt to its teammates. Specifically, the ad hoc agent uses UCT to quickly determine the effects of its actions and plan a sequence of actions that will be most beneficial to the team. UCT is used due to its speed and ability to handle large action and state spaces, allowing it to scale to large numbers of teammates in complex domains. The modified version of the UCT algorithm that is used in this book is explained in Sect. 3.1.3. Other planning algorithms such as Value Iteration (VI) or other approximate planners can also be used, but UCT is chosen here as it shows good empirical performance in many large domains.

Given the current belief distribution over the models, the ad hoc agent can sample teammate models for planning, choosing one model for each rollout similar to the approach adopted by Silver and Veness [6]. Sampling the model once per rollout is desirable compared to sampling a model at each time step because this resampling can lead to states that no model predicts. Ideally, state-action evaluations would be stored and performed separately for each model, but that would require many more rollouts to plan effectively. Instead, the state-action evaluations from all the models are combined to improve the generalization of the planning.

5.2.3 Learning Teammate Models

This section describes how teammate models are learned in the LearnAboutPriorTeammate function of Algorithm 5.2. The previous sections described how the ad hoc agent can select the most accurate model and use it for planning, but they did not specify the source of these models. One option is that the ad hoc agent is given hand-coded models from human experts, as shown in Line 2 of Algorithm 5.1 and in Fig. 5.2. However, there may not always be a source of these models, or the models may be imperfect. Therefore, a more general solution is for the ad hoc agent to learn the models. Learning allows the agent to gain a good set of diverse models over its lifespan, allowing better performance with arbitrary new teammates. The ad hoc agent builds models of past teammates' behaviors offline and then selects from these learned models online while cooperating with new teammates. It is expected that the past teammates are representative of the distribution of future teammates, though the future teammates have not yet been seen.

PLASTIC–Model treats building teammate models as a supervised learning problem, where the goal is to predict the teammates' actions using the features extracted from the world state. The model predicts the next action of each teammate; when this teammate model is combined with a model of the domain, the ad hoc agent can plan far into the future. In this book, our agent uses C4.5 decision trees as implemented

in the Weka toolbox [7] to learn these models. Several other classifiers were tried including SVMs, naive Bayes, decision lists, and nearest neighbor approaches as well as boosted versions of these classifiers. However, decision trees outperformed these methods in initial tests in a combination of prediction accuracy and training time. All model learning is performed offline, reflecting past experience in the domain, but the ad hoc agent updates its belief over the models online.

To capture the notion that the ad hoc agent is expected to have extensive prior general domain expertise (as is assumed in the ad hoc teamwork setting), though not with the specific teammates at hand, we pre-train the ad hoc agent with observations of a pool of past teammates. We treat the observations of previous teammates as experience given to PLASTIC prior to deploying the ad hoc agent.

5.2.4 Adapting Existing Teammate Models

The previous sections discuss how an ad hoc agent should cooperate with teammates it has interacted with before as well as how the agent should cooperate with completely new teammates. However, in many cases, an ad hoc agent may have a limited amount of time to observe its current teammates before it interacts with them. In addition, it has extensive observations from past interactions with other teammates. For example, in pickup soccer, this scenario corresponds to having past experience in pickup soccer, showing up to a new game, and watching a couple minutes before joining in. This scenario fits the *transfer learning* (TL) paradigm, but requires the ability to leverage multiple sources of related data. In this section, we introduce a new transfer learning algorithm (TwoStageTransfer) to leverage such information to speed up learning about new teammates. TwoStageTransfer is a general algorithm that can apply to ad hoc teamwork, and we will present it in general terms here. In the ad hoc teamwork scenario, the observations of prior teammates correspond to the source data sets and observations of the current teammates form the target set.

While the transfer learning algorithms discussed in Sect. 3.1.7 are effective on some problems, they do not directly address the problem of transferring knowledge from multiple sources. In general, they lump all source data into a single data set and expect the learning algorithms to handle this data. TwoStageTransfer is inspired by the TwoStageTrAdaBoost algorithm [8], and it is designed to explicitly leverage multiple source data sets. Specifically in this setting, the ad hoc agent has observed many other agents, some of which are more similar to the target teammate than others. Therefore, tracking the source of the data may be important as it allows the ad hoc agent to discount data from agents that differ greatly from it. Recent research into transfer learning has shown this information may improve results [9–13]. These approaches are promising, but are not directly evaluated here due to the complexity of applying them to our setting coupled with the recency of these algorithms compared to TwoStageTransfer.

TwoStageTransfer's goal is to find the best possible weighting for *each* set of source data and create a classifier using these weights, as described in Algorithm 5.3. TwoStageTransfer takes in the target data set T, the set of source data sets $\mathbb{S} = \{S_1, \ldots, S_n\}$, a number of boosting iterations m, a number of folds k for cross validation, and a maximum number of source data sets to include b. We use the annotation S^w to mean the data set S taken with weight w spread over the instances. The base model learner used in this book is a decision tree learning algorithm that handles weighted instances, but other learning algorithms can be used.

Ideally, TwoStageTransfer would try every combination of weightings, but having n source data sets and m different weightings leads to m^n possible combinations (in the case considered in Chap. 7 there are 10^{28} combinations). Rather than trying all of them, TwoStageTransfer first evaluates each data source independently, and calculates the ideal weight of that data source using cross validation. Then, it adds the data sources in decreasing order of the calculated weights. As it adds each data set, it finds the optimal weighting of that set with the data that has already been added. Finally, it adds the data with the optimal weight and repeats the procedure with the next data set. This algorithm requires only $nm + nm = 2nm$ combinations to be evaluated (in the case from Chap. 7 there are 560 combinations), with nm for the initial evaluations and then m when adding each n data sets. To achieve this efficiency, this approach does not guarantee optimality.

TwoStageTransfer is a general transfer learning algorithm that can be used in a variety of settings for learning from multiple source domains. For example, when classifying the subject of text documents, you may have labeled data from a variety of sources including newspapers, personal letters, and books. When trying to build a classifier for a new magazine, it is useful to transfer information about these other sources, bearing in mind that some sources such as newspapers may be more similar to the magazine than letters. In this book, TwoStageTransfer is used to learn models of teammates' behaviors by transferring knowledge from previously encountered teammates.

In order to integrate TwoStageTransfer with PLASTIC–Model, only a minor change needs to be made to Algorithm 5.1. Specifically, after Line 4, we insert the lines:

$m = TwoStageTransfer(\text{PriorKnowledge}, \text{Observations}(\text{Teammates}))$
$\text{PriorKnowledge} = \text{PriorKnowledge} \cup \{m\}$

This alteration adds a new model to PriorKnowledge that is learned using TwoStage-Transfer, combining the information from previous teammates as well as the limited observations of the new teammates.

Algorithm 5.3 TwoStageTransfer: transfer learning from multiple sources

function TwoStageTransfer:
  inputs:
    T ▷ target data set
    $\mathbb{S}$ ▷ source data sets
  outputs:
    c ▷ learned classifier
  params:
    b ▷ maximum number of source data sets to include
  for all S_i in $\mathbb{S}$: **do**
    $w_i \leftarrow \text{CalculateOptimalWeight}(T, \emptyset, S_i, m, k)$
  Sort $\mathbb{S}$ in decreasing order of w_i's
  $F \leftarrow \emptyset$
  for i from 1 to b **do**
    $w \leftarrow \text{CalculateOptimalWeight}(T, F, S_i, m, k)$
    $F \leftarrow F \cup S_i^w$
  Train classifier c on $T \cup F$
  return c

function CalculateOptimalWeight:
  inputs:
    T ▷ target data set
    F ▷ fixed data set
    S ▷ source data set under consideration
  outputs:
    w^* ▷ best weighting of the source data set
  params:
    m ▷ number of boosting iterations
    k ▷ number of folds for cross validation
  for i from 1 to m **do**
    $w_i = \frac{|T|}{|T|+|S|}(1 - \frac{i}{m-1})$
    Calculate err_i from k-fold cross validation on T using F and S^{w_i} as additional training data
  $w^* = w_j$ such that $j = \text{argmin}_i(\text{err}_i)$
  return w^*

5.3 PLASTIC–Policy

In complex domains, planning algorithms such as UCT may perform poorly due to the inaccuracies of their models of the environment. Therefore, it may be desirable to directly learn a policy for acting in this environment rather than planning online. Learning a policy directly prevents the ad hoc agent from learning to exploit actions that work well in the model, but not in the real environment. Given that the policy learned will depend heavily on the teammates that the agent is cooperating with, it is desirable to learn a policy for each type of teammate. Then, the ad hoc agent will try to pick which policy best fits new teammates it encounters. The remainder

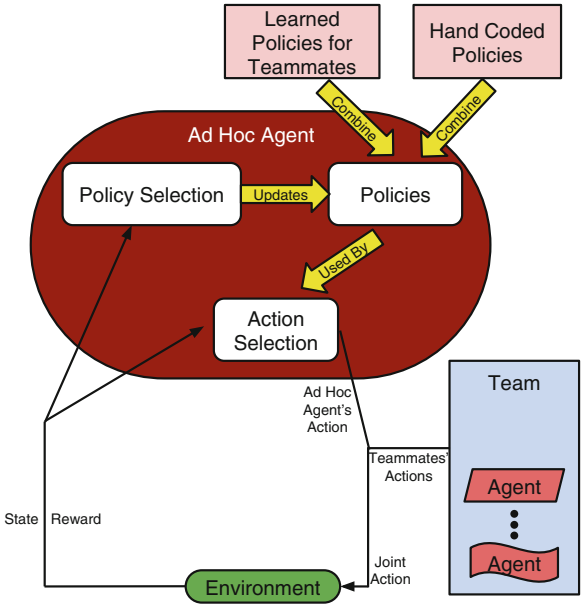

Fig. 5.3 Overview of using the model-based approach of PLASTIC–Policy to cooperate with unknown teammates

of this section describes the PLASTIC–Policy algorithm that uses this approach, summarized in Fig. 5.3. The subroutines used in PLASTIC–Policy are specified in Algorithm 5.4.

5.3.1 Learning the Policy

PLASTIC–Policy learns about its teammates using the LearnAboutPriorTeammate function. Rather than explicitly modeling the MDP's transition function as in PLASTIC–Model, the agent directly uses samples taken from environment with its current teammates. However, online learning is sequential and can take a long time to learn a useful policy on complex domains. Therefore, it is desirable to use a distributed approach that takes advantage of the ability to run many tests simultaneously. To this end, PLASTIC–Policy performs a number of interactions in which it explores the available actions in parallel. It stores its experiences as the tuple $\langle s, a, r, s' \rangle$, where s is the original state, a is the action, r is the reward, and s' is the resulting state.

Using these observations, PLASTIC–Policy can learn a policy for cooperating with its teammates using existing learning algorithms. In this book, the agent uses Fitted Q Iteration (FQI) [14], as described in Sect. 3.1.4. Alternative policy learning algorithms can be used, such as Q-learning [15] or policy search [16].

Algorithm 5.4 Instantiation of functions from Algorithm 5.1 for PLASTIC–Policy.

function LearnAboutPriorTeammate:
  inputs:
    t ▷ the prior teammate
  outputs:
    π ▷ policy for cooperating with teammate t
    m ▷ nearest neighbors model of the teammate's behavior
  params:
    Q-learning parameters: α, λ, and the function approximation
  Data $= \emptyset$
  repeat
    Collect s, a, r, s' for t
    Data $=$ Data $\cup \{(s, a, r, s')\}$
  Learn a policy π for Data using Q-Learning
  Learn a nearest neighbors model m of t using Data
  return (π, m)

function UpdateBeliefs:
  inputs:
    BehaviorDistr ▷ probability distr. over possible teammate behaviors
    s ▷ the previous environment state
    a ▷ previously chosen action
  outputs:
    BehaviorDistr ▷ updated probability distr.
  params:
    η ▷ bounds the maximum allowed loss
  for $(\pi, m) \in$ BehaviorDistr **do**
    loss $= 1 - P(a|m, s)$
    BehaviorDistr$(m) *= (1 - \eta$loss$)$
  Normalize BehaviorDistr
  return BehaviorDistr

function SelectAction:
  inputs:
    BehaviorDistr ▷ probability distr. over possible teammate behaviors
    s ▷ the current environment state
  outputs:
    a ▷ the best action for the agent to take
  $(\pi, m) =$ argmax BehaviorDistr ▷ select most likely policy
  $a = \pi(s)$
  return a

5.3.2 Selecting Policies

This section describes the UpdateBeliefs and SelectAction functions from Algorithm 5.4. When an agent joins a new team, it must decide how to act with these teammates. If it has copious amounts of time, it can learn a policy for cooperating with these teammates. However, if its time is more limited, it must adapt more efficiently.

We assume that the agent has previously played with a number of different teams, and the agent learns a policy for each of these teams. When it joins a new team, the agent can then reuse the knowledge it has learned from these teams to adapt more quickly to the new team. One way of reusing this knowledge is to select from these learned policies. If the agent knows its teammates' identities and has previously played with this team, the agent can directly used the learned policy. However, if it does not know their identities, the agent must select from a set of learned policies.

Many similar decision-making problems can be modeled as multi-armed bandit problems when the problem is stateless. In this setting, selecting an arm corresponds to playing one of the learned policies for an episode. Over time, the agent can estimate the expected values (expected chance of scoring) of each policy by selecting that policy a number of times and observing the outcome.

However, this type of learning may require a large number of trials as the outcomes of playing each policy may be very noisy depending on the complexity of the domain. Therefore, it is desirable to select from the policies more quickly. To this end, PLASTIC–Policy employs an approach based on maintaining the probability of the new team being similar to a previously observed team. These probabilities are updated by observing the actions the team performs and using Bayes' theorem. However, Bayes' theorem may drop the posterior probability of a similar team to 0 for a single wrong prediction. Therefore, as in Sect. 5.2.1, PLASTIC–Policy adopts the approach of updating these probabilities using the polynomial weights algorithm from regret minimization [5]:

$$\begin{aligned} \text{loss} &= 1 - P(\text{actions}|\text{model}) \\ P(\text{model}|\text{actions}) &\propto (1 - \eta * \text{loss}) * P(\text{model}) \end{aligned}$$

where $\eta \leq 0.5$ is a parameter that bounds the maximum loss, where higher values converge more quickly.

The learned policies do not directly give the probability of a past team taking an action. However, the experiences ($\langle s, a, r, s' \rangle$) used in learning the policies can help because they provide estimates of the teams' transition function. When the agent observes a state s and the next state s', it can update the probability of the new team being similar to each old team. For each old team, the agent finds the stored state $\hat{s}$ closest to s and its next state $\hat{s}'$. Then, for each component of the state, it computes the difference between s' and $\hat{s}'$. We assume that the MDP's noise is normal, so each difference results in a probability that it was drawn from the noise distribution. Multiplying these factors together results in a point estimate of the probability of the previous team taking the observed action.

5.4 Chapter Summary

This chapter presents the PLASTIC algorithms used in the remainder of this book. This approach is instantiated in two algorithms: PLASTIC–Model and PLASTIC–Policy. These two algorithms allow an ad hoc agent to reuse knowledge learned from past teammates in order to efficiently learn with new teammates. PLASTIC–Model uses a model-based approach, where it learns a model of past teammates, updates the probabilities of these models online, and then plans using the distribution over models. PLASTIC–Policy is a model-free approach that attacks problems that are less tractable for modeling. PLASTIC–Policy learns policies for cooperates with past teammates, updates the probabilities of using each policy by observing its teammates, and selects actions using the most likely policy. Given this approach, we first look at its theoretical properties in the bandit domain in Chap. 6 and then investigate its empirical performance in Chap. 7.

References

1. Barrett, Samuel, and Peter Stone. 2014. Cooperating with unknown teammates in robot soccer. In *AAAI workshop on multiagent interaction without prior coordination (MIPC 2014)*, July 2014.
2. Barrett, Samuel, Peter Stone, and Sarit Kraus. 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proceedings of the tenth international conference on autonomous agents and multiagent systems (AAMAS)*, May 2011.
3. Barrett, Samuel, Peter Stone, Sarit Kraus, and Avi Rosenfeld. 2013. Teamwork with limited knowledge of teammates. In *Proceedings of the twenty-seventh conference on artificial intelligence (AAAI)*, July 2013.
4. Barrett, Samuel, and Peter Stone. 2015. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Proceedings of the twenty-ninth conference on artificial intelligence (AAAI)*, January 2015.
5. Blum, A, and Y. Mansour. 2007. *Algorithmic game theory, chapter learning, regret minimization, and equilibria*. Cambridge University Press.
6. Silver, David, and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems 23 (NIPS)*. 2010.
7. Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11: 10–18.
8. Pardoe, David, and Peter Stone. 2010. Boosting for regression transfer. In *Proceedings of the twenty-seventh international conference on machine learning (ICML)*, June 2010.
9. Yao, Yi, and G. Doretto. Boosting for transfer learning with multiple sources. In *Proceedings of the conference on computer vision and pattern recognition (CVPR)*, June 2010.
10. Huang, Pipei, Gang Wang, and Shiyin Qin. 2012. Boosting for transfer learning from multiple data sources. *Pattern Recognition Letters* 33(5): 568–579.
11. Zhuang, Fuzhen, Xiaohu Cheng, SinnoJialin Pan, Yu. Wenchao, Qing He, and Zhongzhi Shi. 2014. Transfer learning with multiple sources via consensus regularized autoencoders. In *Machine learning and knowledge discovery in databases*, vol. 8726, ed. Toon Calders, Floriana Esposito, Eyke Hllermeier, and Rosa Meo, 417–431., Lecture notes in computer science Berlin Heidelberg: Springer.
12. Fang, Min, Yong Guo, Xiaosong Zhang, and Xiao Li. 2015. Multi-source transfer learning based on label shared subspace. *Pattern Recognition Letters* 51: 101–106.

13. Ge,Liang, Jing Gao, and Aidong Zhang. 2013. OMS-TL: A framework of online multiple source transfer learning. In *Proceedings of the 22nd ACM international conference on information & knowledge management*, CIKM '13, 2423–2428, ACM, New York, NY, USA, 2013.
14. Damien Ernst, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of machine learning research (JMLR)*, 503–556.
15. Christopher John Cornish Hellaby Watkins. 1989. Learning from Delayed Rewards. Ph.D thesis, King's College, Cambridge, May 1989.
16. Deisenroth, Marc Peter. 2013. Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics* 2(1–2): 1–142.

Chapter 6
Theoretical Analysis of PLASTIC

Chapter 5 introduced the algorithms used in this book for solving ad hoc teamwork problems. Before moving on to the empirical analysis of these algorithms in Chap. 7, it is useful to first investigate the theoretical attributes of PLASTIC. Our analysis focuses on whether the multi-armed bandit domain described in Sect. 3.2.1 is tractable for PLASTIC–Model. We chose to analyze the bandit domain because of its simplicity, which lends itself to more complete theoretical analysis. In addition, the bandit domain is interesting due to its use of communication, which is an important aspect of ad hoc teamwork that is not explored in the other domains. Note that we do not investigate the model learning aspect of PLASTIC–Model. Instead, we analyze whether the PLASTIC–Model can select from a set of known models (from HandCodedKnowledge) and plan its response to these models in polynomial time.

This analysis is performed in the context of the multi-armed bandit domain described in Sect. 3.2.1. Recall that in this domain, several agents simultaneously pull from one of two Bernoulli arms. These agents can broadcast three types of messages to one another: their last observation, the number of pulls and successes they have observed of an arm, or a suggestion to pull an arm. Sending these messages has a cost, so the optimal behavior for the ad hoc agent is to send the minimal amount of messages required to convince its teammates to pull the optimal arm.

Our analysis begins with the simplest version of the problem, in which the ad hoc agent knows its teammates' behaviors and the distributions of the arms. We show that this version of the problem is tractable for PLASTIC–Model to compute the optimal policy. Then, we relax the assumptions of the problem and analyze whether PLASTIC–Model can still compute an optimal or ε-optimal policy for cooperating with its teammates in polynomial time. Specifically, Sects. 6.2–6.5 show that a number of ad hoc team problems in the bandit setting are provably tractable, as summarized in Table 6.1. The problems vary in how much the agent knows about its

This chapter contains material from the publication: [1]. Note that all work presented in this chapter is joint work with Noa Agmon, Noam Hazon, and Sarit Kraus in addition to my advisor Peter Stone.

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_6

Table 6.1 Problems that are solvable in polynomial time

Knowledge of teammates	Teammate type	Knowledge of environment	Solution type	Sections
Known	Stochastic	Known	Exact	6.2
Finite set	Deterministic	Known	Exact	6.3
Parameterized set	Stochastic	Known	Approx.	6.4
Parameterized set	Stochastic	Unknown	Approx.	6.5

teammates, whether its teammates are deterministic, and whether the agent has prior knowledge of the environment in terms of the underlying payoff distributions of the arms. As the agent's knowledge of its teammates and the environment is reduced and it encounters stochastic agents, the problem becomes more difficult. However, the agent is still able to select the best messages and actions in polynomial time, though these selections change from being optimal to only approximately optimal as the problem becomes more difficult. These results prove that ad hoc team agents can plan approximately optimal behaviors involving communication without taking more than polynomial time. The empirical results in Chap. 7 show that communication does improve the team's performance, as well as investigating situations not covered in the theoretical analysis.

6.1 Model and Notation

As discussed in Chap. 5, when the ad hoc agent knows its teammates' behaviors, it can model the bandit problem as an MDP. We now describe the resulting MDP and introduce the notation used in the remainder of this chapter. The MDP's state is composed of the pulls and observations of the ad hoc agent's teammates as well as the messages it has sent. Let $K = (p_0, s_0, p_1, s_1)$ be the knowledge about the arms where p_i and s_i are the number of pulls and successes of arm_i. Then, the state is given by the vector $(K_t, K_a, K_c, r, phase, sugg)$, where K_t is the team's knowledge from their pulls, K_a is the ad hoc agent's knowledge from its pulls, K_c is the knowledge that the ad hoc agent has communicated, r is the current round number, *phase* is the phase of the round, and *sugg* is the ad hoc agent's most recent suggestion. As the n agents on the team are coordinated, their actions depend on K_t and K_c and *not* directly on K_a. We split K_c from K_t to model how the ad hoc agent's messages will affect the team. For example, if the ad hoc agent already communicated an observation, communicating its observations of the same arm will replace its teammates' memory of this observation.

Next, we reason about the number of states and actions of the resulting MDP. Given that there are R rounds and n teammates, p_i and s_i in K_t are each bounded by nR, p_i and s_i in both K_a and K_c are each bounded by R. The round r is bounded

by R, and there are 2 possible phases of a round. Finally, the most recent suggestion *sugg* takes on one of 3 values (arm_0, arm_1, or none). Therefore, the state space has at most $(nR)^4 \times R \times R^4 \times R^4 \times 2 \times 3 = 6n^4R^{13}$ states. This polynomial bound means that the problem is tractable and existing algorithms for solving the MDP can be applied.

The actions of the MDP are the possible arms and the available messages. Arms other than arm_* are considered because their observations affect the messages that the ad hoc agent can send to affect its teammates' actions, so there are 2 actions in the action phase. Let us now look at actions available in the communication phase. In the communication phase, the ad hoc agent can send one message of each type. For the last observation, the ad hoc agent can choose to send its last observation or not, resulting in 2 options. For the observed arm mean, the ad hoc agent can send the mean of either arm or choose to send no message, resulting in 3 options. Finally, as a suggestion, the ad hoc agent can suggest either arm or choose not to suggest an arm, resulting in 3 options. Therefore, there are $2 \times 3 \times 3 = 18$ possible actions in the communication phase, and 2 in the action phase.

The transition function P is composed of the *act* and *comm* functions, the arms' payoff distributions, and the effects of the ad hoc agent's messages. Specifically, *act* and the ad hoc agent's chosen arms affect the p_i values in K_t and K_a respectively, while the arm distributions specify how these actions affect the s_i values in K_t and K_a. The ad hoc agent's messages and K_a define the changes to K_c and *sugg*. The reward function R is a combination of the rewards coming from the arms and the costs of communication.

6.2 Known Teammates and Arms

Given this model, we first analyze the most straightforward setting. In this setting, the ad hoc agent knows the true distributions of the arms and can observe its teammates' actions and the resulting payoffs. In addition, it knows the true stochastic behavior (*act* and *comm*) of its teammates. Therefore, the ad hoc agent has a full model of the problem described in Sect. 6.1. It is possible to find the optimal solution to an MDP using DP in time polynomial in the MDP's size, which is polynomial in the number of rounds R and teammates n. Therefore, Proposition 1 directly follows.

Proposition 1 *An ad hoc agent that knows the true arm distributions and its teammates' behaviors can calculate its optimal behavior for maximizing the team's shared payoffs in poly(R, n) time.*

Proof Using Dynamic Programming (DP), it is possible to find the optimal solution to an MDP in polynomial time in terms of the number of states and actions [2]. As shown in Sect. 6.1, the resulting MDP has a state space that has size polynomial in R and n, and the number of actions is bounded by 18. Therefore, the ad hoc agent can use DP to optimally plan its actions in poly(R, n) time.

6.3 Teammates from a Finite Set

In this section, we relax the constraint on knowing the teammates' behaviors. Rather than knowing the specific behavior of its teammates, the ad hoc agent instead knows that the behaviors are drawn from a known, finite set of deterministic behaviors. In addition, it still knows the true distributions of the arms. This case is of interest because a finite set of behaviors can often cover the space of likely behaviors. For example, analysis of bandit problems [3], ad hoc teamwork [4], and using machine learning with psychological models [5] suggests that a small number of behaviors can represent the spread of possible behaviors.

In general, this finite set of behaviors can vary, but in this analysis, we consider two types of teammates: (1) greedy agents and (2) ones that choose arms using confidence bounds in the form of UCB1 [6]. The UCB1 agents select actions using

$$arm = \underset{i}{\text{argmax}} \frac{s_i}{p_i} + c\sqrt{\frac{\ln(p_0+p_1)}{p_i}} \tag{6.1}$$

where $c = 1$. The ad hoc agent is given a prior probability distribution over teams following either of these behaviors. The teammates are assumed to use the ad hoc agent's communicated pulls when selecting their actions. Additionally, we assume that these teammates share all information with each other and send messages that the ad hoc agent can hear, but these messages do not reveal the teammates' behaviors.

To analyze this problem, we add the ad hoc agent's beliefs about its teammates into the state space that the agent plans over. As the teammates are deterministic, there are three possibilities for the belief space: both models are still possible, only the greedy model is possible, or only the UCB1 model is possible. Therefore, the combined belief and world state space is three times larger than the world state space, and the resulting MDP has state space of size $18n^4R^{13}$. In general, the increase in size is $2^k - 1$ where k is the number of models, but we assume that k is fixed and not a problem parameter. The transition function can be modified to simultaneously update the ad hoc agent's beliefs as well as the world state based on whether a teammate model predicts the observed actions. Therefore, the MDP can again be solved using DP in polynomial time. Proposition 2 follows directly from this reasoning.

Proposition 2 *An ad hoc agent that knows the true arm distributions and that its teammates' behaviors are drawn from a known set of two deterministic behaviors can calculate its optimal behavior for maximizing the team's shared payoffs in poly*(R, n) *time.*

Proof Similar to Theorem 1, we know that the resulting MDP has a state space that has size polynomial in R and n. In addition, using DP it is possible to optimally solve an MDP in polynomial time with respect to the size of the state and action spaces. Given that the MDP's state space is polynomial in R and n and there are 18 actions, it is possible to find the optimal behavior in poly(R, n) time using DP.

6.4 Teammates from a Continuous Set

In this section, we further relax the constraints on the teammates' behaviors, considering a continuous set of stochastic behaviors rather than the discrete set of deterministic behaviors used in the last section. We still consider a small number of possible behaviors, specifically ε-greedy and UCB(c). For these behaviors, ε is the probability of taking a random action, and c is the scaling factor of the confidence bound in Eq. 6.1. Therefore, the ad hoc agent must maintain a belief distribution over values of ε, values of c, and p the probability of the teammates being ε-greedy. The ad hoc agent is given the prior knowledge that ε, c are uniformly distributed over [0, 1], and it starts with an initial estimate of p. While we use two models for simplicity, this analysis can be extended for any fixed number of parameterized models.

To analyze this problem, we model the problem as a POMDP as discussed in Sect. 3.1.5. The transition function for the fully observable state variables remains the same as in the original MDP. In this setting, the belief space has three partially observed values: ε, c, and p the probability of the teammates being ε-greedy versus UCB(c). The value of p is updated using Bayes' rule given the probability of the models predicting the observed actions, and the updates to the probability distributions of ε and c are described in Lemma 1. The remainder of the POMDP remains as defined above.

In Lemma 1 and Theorem 1, we show that in this expansion of the problem, the ad hoc agent can perform within η of the optimal behavior with calculations performed in polynomial time. This result comes from reasoning about the δ-covering of the belief space, which defines the difficulty of solving the POMDP as discussed in Sect. 3.1.5.

Lemma 1 *The belief space of the resulting POMDP has a δ-covering with size $poly(R, n, 1/\delta)$.*

Proof The resulting size of the δ-covering is a product of the contributing factors. These factors come from the underlying MDP state s, ε, c, and p. Using Proposition 1 of Hsu et al.'s work [7], we know that the fully observed state variables result in a multiplicative factor that is polynomial in R and n. Therefore, since the ad hoc agent directly observes s, it only results is a factor of poly(R, n). The probability of the two models p is a single real value in [0,1], resulting in a factor of $1/\delta$. The parameter ε has a uniform prior, so the posterior is a beta distribution, relying on two parameters, α and β. These parameters correspond to the (fully observed) number of observed greedy and random pulls; thus, each are integers bounded by nR. Therefore, the probability distribution over ε can be represented using a factor of size $(nR)^2$.

The parameter c has a uniform prior, and UCB agents select arms using Eq. 6.1, combining the communicated and team's pulls by setting $p_j = p_j^t + p_j^c$ and $s_j = s_j^t + s_j^c$. The teammates will only select the lower arm when c is above a certain value and the higher arm when c is below a certain value. Therefore, the top and bottom ranges of c can be updated using linear programming from observing their actions. Note that the posterior remains uniform; only the range changes. Therefore,

the probability distribution over c can be represented using two real values in [0, 1] that are the top and bottom of the uniform range of c, resulting in a factor of $1/\delta^2$. Multiplying all of these factors results in a δ-covering of size poly$(R, n, 1/\delta)$.

As discussed in Sect. 3.1.5, a POMDP can be solved approximately in polynomial time given a covering set. Given this result and Lemma 1, Theorem 1 follows directly.

Theorem 1 *Consider an ad hoc agent that can observe its teammates' actions, knows the true arm distributions, and knows that its teammates are drawn from a known, continuous set of ε-greedy and UCB teammates. This agent can calculate an η-optimal behavior in poly$(n, R, 1/\eta)$ time.*

Proof From Theorem 1 in Kurniawati et al.'s work [8], it is known that a POMDP can be solved in time polynomial in terms of the size of its covering number. While this theorem applies to the infinite horizon, discounted rewards case, any finite horizon POMDP can be converted into an infinite horizon POMDP by adding a sink state that results in no rewards. In addition, the undiscounted rewards can be converted to discounted rewards by multiplying by the inverse of the discount factor. Therefore, a finite horizon POMDP can also be solved in polynomial time with respect to the size of its covering number. From Lemma 1, we know that the combined state and belief space of the POMDP has a proper δ-covering of size polynomial in R, n, and $1/\delta$. Kurniawati et al. [8] showed that for the result to hold, δ needs to be polynomial in terms of $1/\eta$, the horizon, and the one step reward, which is bounded by n. Therefore, the η-optimal behavior can be calculated in poly$(n, R, 1/\eta)$ time.

6.5 Unknown Arms

The previous sections assumed that the ad hoc agent already knew the underlying distributions of the arms (i.e. the POMDP's transition function), but in many cases the ad hoc agent may not have this information. Therefore, it is desirable for the ad hoc agent to reason about trading off between exploring the domain, exploring its teammates, and exploiting its current knowledge. In this section, we prove that the ad hoc agent can optimally handle this tradeoff while planning in polynomial time. We again assume that the ad hoc agent knows its teammates' pulls and results, either by observing them directly or by listening to its teammates' messages.

The belief space of the POMDP is increased to track two additional values, one for the Bernoulli success probability of each arm. The probabilities of these values can be tracked using a beta distribution similar to ε in Lemma 1, resulting in an additional multiplicative factor of $(nR)^2$. Therefore, the covering number has size poly$(R, n, 1/\delta)$. Theorem 2 follows naturally from this result and the reasoning in Theorem 1.

Theorem 2 *Consider an ad hoc agent that does not know the true arm distributions, but has a uniform prior over their success probabilities, knows that its teammates' behaviors are drawn from a continuous set of ε-greedy and UCB teammates, and can observe the results of their actions. This agent can calculate an η-optimal behavior in poly$(n, R, 1/\eta)$ time.*

Proof We know that a POMDP can be solved in time polynomial in its covering number. From Theorem 1, we know that the ad hoc agent's beliefs about its teammates' behaviors and the observed pulls can be covered in a polynomial number of points. In this setting, the ad hoc agent must also track its beliefs about the success probability of each arm. The reasoning proceeds similarly to the reasoning about ε in Lemma 1. The agent starts with uniform beliefs about each arms' success probability, which leads the posterior to be a beta distribution, which can be represented using two integer parameters. These parameters correspond to the (fully observed) numbers of successes and pulls observed; thus the integers can are be bounded by $(n + 1)R$ for each arm. Representing the probability distribution of the two arms' success probabilities leads to a factor of size $((n + 1)R)^2$. Therefore, the η-optimal behavior can still be calculated in poly$(n, R, 1/\eta)$ time.

6.6 Chapter Summary

This chapter presents theoretical analysis of the PLASTIC–Model algorithm in the multi-armed bandit setting described in Sect. 3.2.1. These results show that PLASTIC–Model can calculate an ϵ-optimal policy for the ad hoc agent to follow in a variety of scenarios in polynomial time. The analysis proceeds by bounding the number of states and actions in the resulting MDPs and POMDPs. When the ad hoc agent is uncertain about its teammates' behaviors or the true success probabilities of the arms, it can efficiently represent its uncertainty about these beliefs. The compactness of these beliefs and the size of the state space enables us to prove that there are efficient ways to calculate the optimal behavior for the ad hoc agent. This result suggests that empirical approaches for solving POMDPs will be effective in this domain, a hypothesis which is explored in Chap. 7.

References

1. Barrett, Samuel, Noa Agmon, Noam Hazon, Sarit Kraus, and Peter Stone. 2014. Communicating with unknown teammates. In *Proceedings of the twenty-first european conference on artificial intelligence*, Aug 2014.
2. Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement learning: An introduction*. Cambridge: MIT Press.
3. Mayo-Wilson, Conor, Kevin Zollman, and David Danks. 2012. Wisdom of crowds versus groupthink: learning in groups and in isolation. *International Journal of Game Theory* 42: 695–723

4. Barrett, Samuel, Peter Stone, Sarit Kraus, and Avi Rosenfeld. 2013. Teamwork with limited knowledge of teammates. In *Proceedings of the twenty-seventh conference on artificial intelligence* (*AAAI*), July 2013.
5. Rosenfeld, Avi, Inon Zuckerman, Amos Azaria, and Sarit Kraus. 2012. Combining psychological models with machine learning to better predict people's decisions. *Synthese* 189: 81–93.
6. Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* (*MLJ*) 47: 235–256.
7. Hsu, David, Wee Sun Lee, and Nan Rong. 2007. What makes some POMDP problems easy to approximate? In *Advances in Neural Information Processing Systems 20* (*NIPS*).
8. Kurniawati, Hanna, David Hsu, and Wee Sun Lee. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of robotics: Science and systems*.

Chapter 7
Empirical Evaluation

While the previous chapter describes the theoretical analysis of the PLASTIC algorithm, this chapter presents its empirical analysis. This empirical analysis covers the three domains introduced in Sect. 3.2: the multi-armed bandit domain, the pursuit domain, and half field offense in the 2D RoboCup simulator. An overview of the experiments is presented in Table 7.1. This table lists the domains and teammate types used in each experiment as well as whether the teammates have been previously seen or provided in HandCodedKnowledge. Furthermore, for each experiment, we describe whether the ad hoc agent knows the environment, how many teammates it is cooperating with, whether it uses communication to cooperate with its teammates, and whether the domain provides continuous states and actions. Finally, we specify whether we test PLASTIC–Model or PLASTIC–Policy in each experiment. In the table, we bold the factors that result in extra complexities and show that PLASTIC is applicable to other complex domains. Specifically, we highlight when the teammates were externally created, when the teammates are previously unseen or only seen briefly, when the domain has continuous states and actions, when the environment in unknown, and when PLASTIC has to select from a set of parameterized models.

This analysis tests the hypothesis that PLASTIC is effective for enabling agents to quickly adapt to new teammates in a variety of possible ad hoc teamwork scenarios. In Sect. 7.1, we start by evaluating whether PLASTIC–Model can efficiently communicate with teammates given a limited language as well as whether it can select good models from a set of parameterized hand-coded models for HandCodedKnowledge including the case where these models do not cover its teammates' true behaviors. Then, in Sect. 7.2, we test the hypothesis that PLASTIC–Model can use models it learned from previous teammates to adapt quickly to new teammates. Furthermore,

This chapter contains material from four publications: [1–5]. Note that the work in Sect. 7.1 is joint work with Noa Agmon, Noam Hazon, and Sarit Kraus in addition to my advisor Peter Stone [1]. In addition, Sects. 7.2.1, 7.2.6, and 7.2.7 are joint work with Sarit Kraus and Avi Rosenfeld in addition to Peter Stone [4].

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_7

Table 7.1 An overview of the experiments described in this chapter. We denote hand-coded by HC, Externally-created by Ext., and parameterized by Param

Sections	Domain	Teammate type	Teammate knowledge	Teammates previously seen	Environment known	Number of teammates	Uses comm.	Continuous state/actions	PLASTIC–Model or PLASTIC–Policy
7.1.3	Bandit	HC	**Param.** HC set	Yes	Yes	7	**Yes**	No	Model
7.1.4	Bandit	**Ext.**	**Param.** HC set	**No**	Yes	1–9	**Yes**	No	Model
7.1.5	Bandit	HC and **Ext.**	**Param.** HC set	Yes and **No**	**No**	1–9	**Yes**	No	Model
7.2.3	Pursuit	HC	Known	Yes	Yes	3	No	No	Model
7.2.4	Pursuit	HC	HC set	Yes	Yes	3	No	No	Model
7.2.5	Pursuit	**Ext.**	HC set	**No**	Yes	3	No	No	Model
7.2.6	Pursuit	**Ext.**	Learned set	Yes and **No**	Yes	3	No	No	Model
7.2.7	Pursuit	**Ext.**	Learned set + **TwoStage-Transfer**	**Briefly**	Yes	3	No	No	Model
7.3.4	Limited HFO	**Ext.**	Learned set	Yes	Yes	1	No	**Yes**	Policy
7.3.5	Full HFO	**Ext.**	Learned set	Yes	Yes	3	No	**Yes**	Policy

we assess whether TwoStageTransfer is effective for learning models of new teammates while transferring knowledge about past teammates. Specifically, we perform this evaluation by using the learned models in PLASTIC–Model for cooperating with these new teammates. Finally, we test the hypothesis that PLASTIC–Policy allows ad hoc agents to quickly adapt to new teammates in a complex domain (HFO) in Sect. 7.3.

7.1 Multi-armed Bandits

We start with the simplest domain studied in this book, specifically the multi-armed bandit setting. The multiagent, multi-armed bandit setting used in this analysis is described in depth in Sect. 3.2.1. While Chap. 6 showed that this bandit problem can be modeled as a POMDP, and thus is theoretically tractable to solve in polynomial time. Given this theoretical result, we expect that PLASTIC–Model should be effective in this domain. Therefore, this section investigates whether PLASTIC–Model is practical and shows that it enables an ad hoc agent to cooperate with its teammates better than alternative approaches. In addition, we also consider a number of scenarios that go beyond those handled in the theoretical analysis. The results presented in this section show that modeling the problem as a (PO)MDP and planning using this model significantly improves the performance of the team compared to several intuitive baseline behaviors in several scenarios.

7.1.1 Methods

To cooperate effectively in the bandit problem, our agent uses the PLASTIC–Model algorithm. We model the bandit problem as a POMDP, as discussed in Sect. 6.1. Chapter 6 showed that calculating the approximately optimal behavior in the resulting POMDPs takes polynomial time, but the polynomial quickly becomes too large for practical computation as the size of the problem grows. Therefore, PLASTIC–Model uses Monte Carlo planning to find an inexact solution in a practical amount of time. In addition, as the beliefs over the teammate parameters may be continuous, PLASTIC–Model approximates the belief update for the sake of efficiency. To this end, PLASTIC–Model uses Partially Observable Monte Carlo Planning (POMCP) [6] to perform the planning and belief updates. POMCP is presented in more depth in Sect. 3.1.6. Past research has shown that POMCP is effective for scaling to large POMDPs, producing effective policies for behaving in these POMDPs. While POMCP is not guaranteed to find an optimal solution given limited computation, our results show that it plans an effective behavior in our setting.

In these tests, PLASTIC–Model is given prior knowledge in the form of HandCodedKnowledge, a set of hand-coded behaviors of possible teammates. Specifically, HandCodedKnowledge is composed of teammates following the

ε-greedy or UCB(c) algorithms, described in Sect. 3.2.1.2. This set provides a uniform distribution of ε and c drawn from [0,1]. Similarly, the probability of the team following the ad hoc agent's suggestion s is also drawn from [0,1]. Therefore, the problem is for PLASTIC–Model to determine which of these behaviors best fits the teammates it encounters, as well as which actions to take based on these beliefs.

7.1.2 Evaluation Setup

In our evaluations, we compare four potential behaviors for the ad hoc agent to follow:

- **Match**—Plays as if it were another agent of the team's type, but can observe all agents' results
- **NoComm**—Pulls the best arm and does not communicate
- **Obs**—Pulls the best arm and sends its last observation
- **PLASTIC–Model**—Selects arms and messages using PLASTIC–Model

Match, NoComm, and Obs serve as baselines. Pulling the best arm and sending other messages were tested, but generally produced worse results than either NoComm or Obs. Match is only used as a baseline when the arms' payoffs are unknown.

In our theoretical analysis in Chap. 6, we prove that problems with 2 arms and teammates that are coordinated and use ε-greedy or UCB behaviors are tractable. These tests will investigate scenarios with more arms and externally-created teammates that are not coordinated and do not use ε-greedy or UCB behaviors. Unless otherwise specified, the version of the bandit problem used for the evaluations has 10 rounds, 7 teammates, and 3 arms. The costs for sending messages are known by all agents and randomly selected for each run. These costs are sampled from $[0, m|c|]$, where $|c|$ is the size of the message (3 for mean, 2 for obs, and 1 for sugg) and $m = 0.75$ unless otherwise specified.

In all of the evaluations, we assume that the ad hoc agent can observe its teammates' actions and payoffs. The ad hoc agent knows the true distributions of the arms except where otherwise noted (Fig. 7.3). Our evaluations use 100 trials. The randomness of the trials is fixed across the different ad hoc agent behaviors to allow for paired statistical tests. The results are average team rewards normalized by the average reward if all agents repeatedly pull the best arm with no communication costs. Statistical significance is tested using a Wilcoxon signed-rank test with $p < 0.05$, denoted by "+" in the figures when comparing PLASTIC–Model to all other methods.

7.1.3 Hand-Coded Teammates

In this first set of experiments, we test the hypothesis that PLASTIC–Model can effectively cooperate with unknown teams of coordinated ε-greedy and UCB(c)

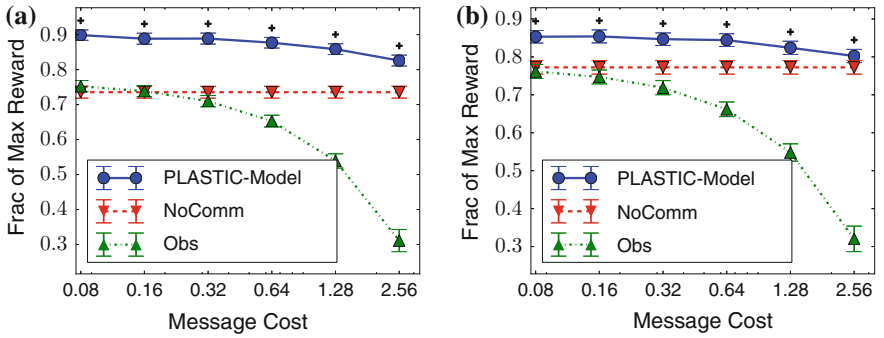

Fig. 7.1 Normalized rewards with varied message costs with a logarithmic x-axis. Significance is denoted by "+". **a** ε-greedy teammates **b** UCB teammates

teammates. These experiments evaluate whether PLASTIC–Model can determine the behaviors of its teammates and the parameters of these behaviors on the fly. We hypothesize that PLASTIC–Model will outperform the baselines introduced above.

These evaluations are over 100 trials with teams where ε, c, s, and the arms' success probabilities are selected randomly uniformly between 0 and 1. PLASTIC–Model is initialized with the beliefs that both ε-greedy and UCB(c) teammates are equally likely, so its prior beliefs include the correct behavior. Therefore, PLASTIC–Model attempts to determine the correct teammate type and parameters for its teammates. These hand-coded teammates are explained in detail in Sect. 3.2.1.2.

Figure 7.1 presents the results when the ad hoc agent encounters the problem discussed in Sect. 6.4, cooperating with teams that are ε-greedy or UCB, with varied message costs. Note that NoComm is unaffected by the message costs as it does not communicate. The results indicate that the agent can effectively plan its actions, significantly outperforming the baselines. The performance of PLASTIC–Model diminishes as the cost of messages rises because affecting the teammates becomes more costly. However, PLASTIC–Model will plan not to communicate when the message costs get too high, so its performance never drops below that of NoComm. The results are similar when the ad hoc agent knows its teammates' true behavior, rather than assuming that both types are possible.

The results of these evaluations show that PLASTIC–Model can quickly learn to cooperate with various teammates when it knows that its teammates' behaviors are drawn from a known, parameterized set. PLASTIC–Model efficiently uses communication to improve the team's performance, thus scaling its communication as the cost of communicating grows.

7.1.4 Externally-Created Teammates

The previous section presented z with a set of possible hand-coded teammates, specifically the ε-greedy and UCB teammates. However, this set of teammates is limited

and may not represent the set of possible teammates the ad hoc agent may encounter. Therefore, we also consider a number of agents that were not created by the authors, denoted *externally-created teammates*. These teammates were created by students for a class assignment and are presented in more depth in Sect. 3.2.1.3. To prevent any bias in the creation of the agents, the students designed the entire team without considering ad hoc teamwork. These agents serve as a sample of the variety of teammates an ad hoc agent might encounter in real scenarios. We hypothesize that PLASTIC–Model will use communication to cooperate effectively with these teammates despite the inaccuracies of its expert-provided prior knowledge (HandCodedKnowledge).

Section 3.2.1.1 specifies that the teammates are assumed to be tightly coordinated and know each other's actions and payoffs via communication. However, the externally-created agents do not always choose to share this information, breaking this assumption. In addition, the externally-created agents follow a variety of behaviors. Rather than being optimal, these agents represent a diverse set of imperfect agents that may be created by a variety of designers attempting to solve a real problem. We specifically did not analyze their behaviors to prevent biasing the design of our ad hoc agent.

Given that the externally-created teams quickly converge to the best arm, all approaches perform similarly with these teammates. Therefore, we investigate how well the ad hoc agent can correct its teammates' knowledge if its teammates have incorrect beliefs. To this end, we look at the worst case scenario for the team: the best arm performs poorly early in the scenario, possibly misleading the team into not pulling the arm later. To create this setting, we consider the case where in the first 5 rounds, the teammates' pulls of the best arm are biased to have a lower chance of success. In this setting, both the teammates and the ad hoc agent are unaware of the initial bias of the arm. Therefore, this test evaluates how well the ad hoc agent can use its prior knowledge to correct the misinformation its teammates have observed.

As in the previous tests, PLASTIC–Model is again provided with the same prior knowledge, specifically HandCodedKnowledge is the set of ε-greedy and UCB(c) hand-coded policies with their various parameters initialized uniformly randomly. Despite this prior knowledge being incorrect, PLASTIC–Model is still able to determine which of these behaviors best fit its teammates and perform well. Figure 7.2 shows the results with externally-created agents, a problem not covered by any theoretical guarantees, as the models do not match the true teammates. In these evaluations, we test the sensitivity of the agent to various problem parameters, investigating under which conditions POMCP outperforms the baselines. Note that the message costs are also applied to the externally-created teammates, which know the current message costs, so the performance of NoComm is now affected by message costs.

As the cost of communicating increases, NoComm becomes closer to the optimal behavior. As the number of rounds increases, communicating is more helpful because there is more time to reap the benefits of better informing the teammates. With more arms, it is harder to get the teammates to select the best arm, so communicating is less helpful. With more teammates, communicating is more likely to be outweighed by

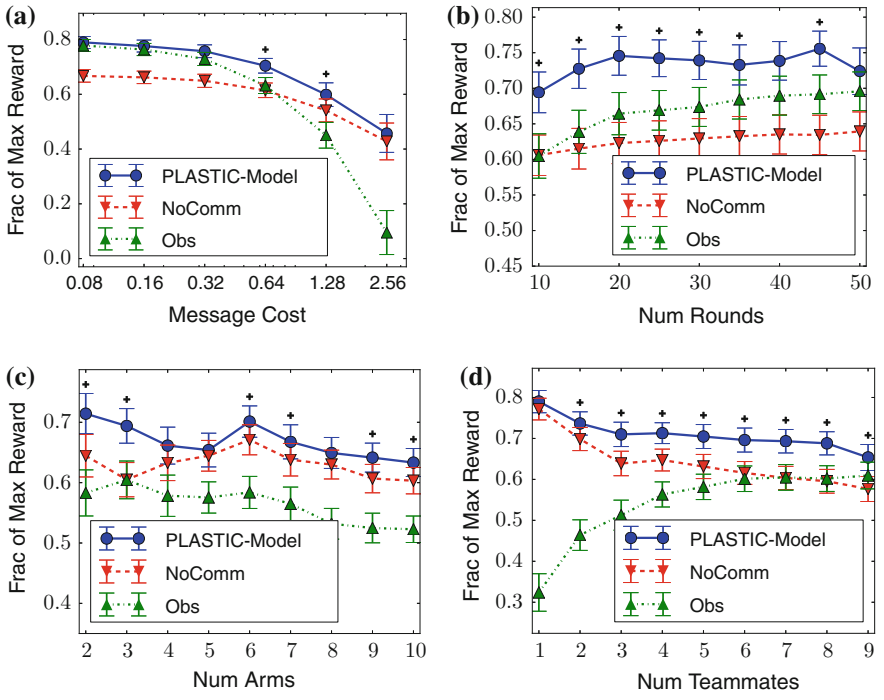

Fig. 7.2 Normalized rewards with varied parameters when cooperating with externally-created teammates. **a** Message costs with logarithmic x-axis. **b** Numbers of rounds. **c** Numbers of arms. **d** Numbers of teammates

other agents' messages, which explains the decreasing performance of PLASTIC–Model. However, communication can still encourage the teammates to pull better arms, leading to PLASTIC–Model outperforming NoComm. Overall, the results in these scenarios tell a similar story, specifically that reasoning about communication helps an ad hoc agent effectively cooperate with various teammates, even when its models of these teammates are incomplete or incorrect.

7.1.5 Unknown Arms

While the previous sections investigated how an ad hoc agent can cooperate with a variety of teammates, the ad hoc agents were provided with prior knowledge about the underlying distributions of the arms. This section investigates a scenario in which the ad hoc agent is also uncertain about the true payoffs of the arms and must simultaneously learn about the world and its teammates, as discussed in Sect. 6.5. We still assume that the ad hoc agent observes the payoffs of its teammates' actions, for example by listening to their messages. These tests evaluate the hypothesis that

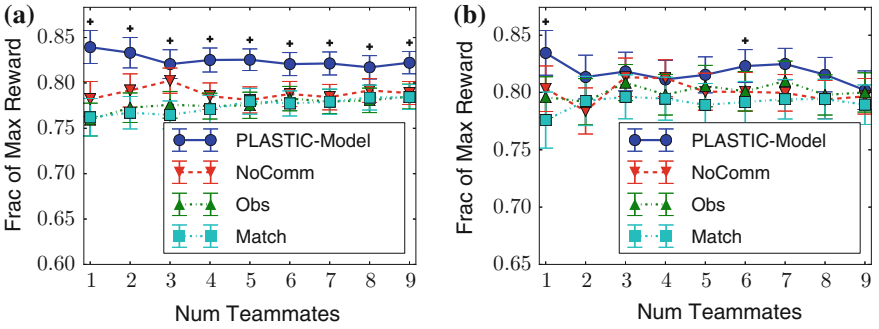

Fig. 7.3 Normalized rewards when dealing with unknown arms and varying numbers of teammates **a** Mix of ε-greedy and UCB teammates. **b** Externally-created teammates

PLASTIC–Model can effectively cooperate with its team using communication while learning about both its teammates and its environment.

Figure 7.3 shows the results for this scenario. When using PLASTIC–Model, the ad hoc agent samples its initial beliefs of the underlying world state by randomly selecting the payoff value of each arm. In the NoComm and Obs settings, the ad hoc agent chooses arms ε-greedily, with $\varepsilon = 0.1$, because it does not know the true best arm. To encourage more sharing, the base message cost is set to $m = 0.04$. The results show that when the ad hoc agent is unsure of the arms' payoffs, PLASTIC–Model enables it to learn about its environment while using communication intelligently to cooperate with unknown teammates.

7.1.6 Summary

This section presented several evaluations of PLASTIC–Model in the bandit domain that test PLASTIC–Model's ability to use communication to cooperate with its teammates. These results show that PLASTIC–Model is capable of reasoning both about what actions to take and what messages to send to its teammates. The first tests demonstrate that PLASTIC–Model can cooperate with a set of hand-coded teammates while selecting from a set of parameterized hand-coded teammate behaviors. In addition, PLASTIC–Model can successfully cooperate with externally-created teammates, for which it has no theoretical guarantees and only imperfect models. Finally, PLASTIC–Model can balance learning about the domain and its teammates at the same time when the payoff distributions of the arms are unknown. These results show that PLASTIC–Model can effectively use limited communication to cooperate with unknown teammates.

7.2 Pursuit

The previous section focused on analyzing PLASTIC in a simple domain, specifically the multi-armed bandit domain. If we are interested in applying PLASTIC to realistic scenarios, it is important to see how well it scales. Therefore, this section looks at PLASTIC's performance on a more complex domain in the form of the pursuit domain introduced in Sect. 3.2.2. As in the bandit domain, we use PLASTIC–Model in the pursuit domain.

7.2.1 Methods

To employ PLASTIC–Model, we model the pursuit domain as an MDP. States in the MDP are the current positions of all agents, and the actions are to move in one of the four cardinal directions or stay still. The transition function is deterministic except for collisions, which are handled based on a random priority assigned each time step. The reward function returns 1.0 when the prey is captured and 0 otherwise.

In Sects. 7.2.6 and 7.2.7, PLASTIC–Model learns models of its teammates, as discussed in Sect. 5.2.3. Learning allows the agent to gain a good set of diverse models over its lifespan, allowing better performance with arbitrary new teammates. The ad hoc agent builds models of past teammates' behaviors offline and then selects from these learned models online while cooperating with new teammates. It is expected that the past teammates are representative of the distribution of future teammates, though the future teammates have not yet been seen.

PLASTIC–Model treats building teammate models as a supervised learning problem, where the goal is to predict the teammates' actions using the features in Table 7.2 with all positions being relative to the modeled teammate. The model predicts the next action of each teammate; when combined with a model of the domain, the ad hoc agent can plan far into the future. With its observations of past teammates, the ad hoc agent learns a decision tree, implemented in the Weka toolbox [7]. Several other classifiers were tried including SVMs, naive Bayes, decision lists, and nearest neighbor approaches as well as boosted versions of these classifiers. However, decision trees outperformed these methods in initial tests in a combination of prediction accuracy and training time. All model learning is performed offline, reflecting past experience in the domain, but the ad hoc agent updates its belief over the models online.

The features in Table 7.2 are mostly the relative locations of other agents in the domain. The features also include whether the predator is currently neighboring the prey and whether each of the four cells around the prey are occupied by predators, which gives information about which direction the predator may move to fill the empty spots. Also, we include which of the four numbers the predator is assigned in case agents on a team are specialized based on their number. Finally, the previous two actions give a succinct, but imperfect summary of the predator's intentions; we expect that predators are likely to continue in their current direction, but the learning algorithm figures out how this history predicts the next action.

Table 7.2 Features for predicting a teammate's actions

Description	Num. Features	Values
Predator number	1	{0, 1, 2, 3}
Prey x position	1	$\{-10, \ldots, 10\}$
Prey y position	1	$\{-10, \ldots, 10\}$
Predator$_i$ x position	3	$\{-10, \ldots, 10\}$
Predator$_i$ y position	3	$\{-10, \ldots, 10\}$
Neighboring prey	1	{true,false}
Cell neighboring prey is occupied	4	{true,false}
Previous two actions	2	$\{\leftarrow, \rightarrow, \uparrow, \downarrow, \bullet\}$

Positions are relative to the teammate

To capture the notion that the ad hoc agent is expected to have extensive prior general domain expertise (as is assumed in the ad hoc teamwork setting), though not with the specific teammates at hand, PLASTIC–Model observes a number of past teammates. Specifically, it watches teams of four predators for 50,000 steps for each past teammate type, and builds a separate model for each type of teammate. Preliminary tests show that less data can still be effective, but the focus of this research is about minimizing observations of the current teammates, not the previous ones. We treat the observations of previous teammates as experience prior to deploying the ad hoc agent. If some observations of the current teammates are available, we can improve our results using transfer learning in the form of TwoStageTransfer as discussed in Sect. 7.2.7.

7.2.2 Evaluation Setup

We evaluate how PLASTIC–Model compares to the baseline of directly copying the teammates' behaviors. Copying the teammates' behaviors tests how the team would perform if it had another teammate that matched the team rather than the ad hoc team agent. We use the following performance metric: given 500 steps, how many times can the predators capture the prey. Whenever the prey is caught, it is randomly relocated and the predators try to capture the prey again. Results are averaged over 1,000 trials, and statistical significance is tested using a Wilcoxon signed-rank test with $p < 0.01$.

7.2.3 Cooperating with Known Teammates

Before analyzing whether PLASTIC–Model is effective at cooperating with unknown teammates, it is first informative to test whether it can cooperate with known teammates on a known task. Specifically, we test its performance with the hand-coded

teammates presented in Sect. 3.2.2.1. PLASTIC–Model is given the prior knowledge in the form of the correct hand-coded policy of its teammates behaviors for HandCodedKnowledge. Although the ad hoc team agent has a full model of its teammates, this scenario is still an ad hoc teamwork setting because there is no opportunity for the team to coordinate prior to starting the task: the agent must determine its strategy online. We hypothesize that PLASTIC–Model will effectively plan to deal with its known teammates and outperform matching their suboptimal behaviors.

When both the teammates and the task are known, finding the optimal behavior with PLASTIC–Model simplifies to a planning algorithm. As presented in Sect. 3.1.2, Value Iteration (VI) is a planning algorithm that is guaranteed to compute the optimal behavior for the ad hoc agent, but it is computationally intensive to calculate. In order to scale to larger problems, it is desirable to use more efficient, approximate methods such as Upper Confidence bounds for Trees (UCT), which is discussed in Sect. 3.1.3. Ideally, the approximate solutions will not lose too much compared to the optimal solutions. Therefore, we look at the performance of these two different planning algorithms for PLASTIC–Model, as well as the baseline of matching the teammates' behaviors.

Results for three sizes of worlds are given in Fig. 7.4. These results show that the ad hoc agent can do much better than just copying the behavior of its teammates by

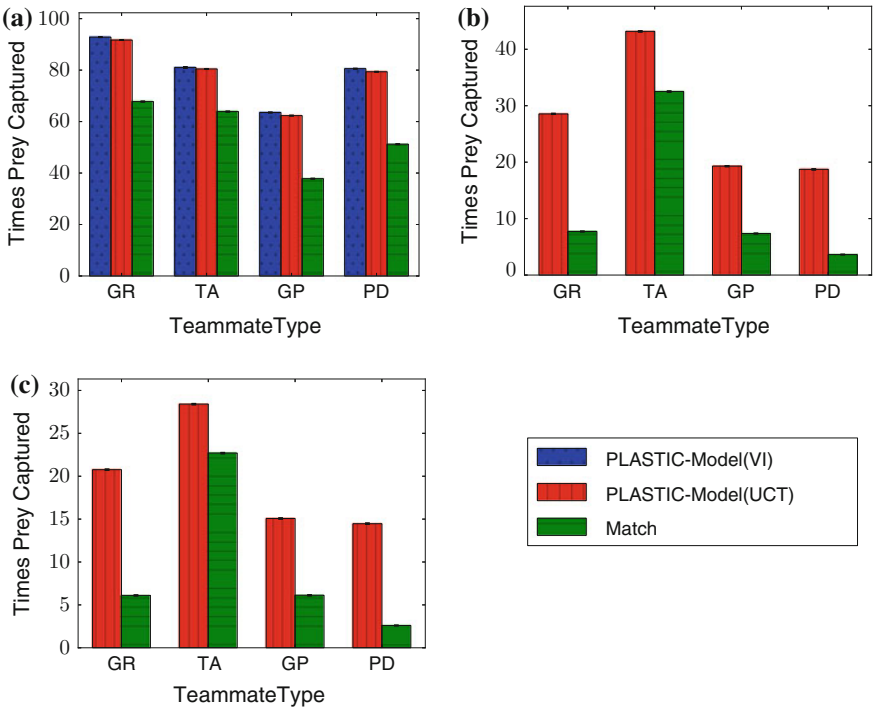

Fig. 7.4 Results with known hand-coded teammates. **a** 5 × 5 world. **b** 10 × 10 world. **c** 20 × 20 world

using PLASTIC–Model. In the 5×5 world, following the optimal behavior found by VI captures the prey an average of 92.82 and 81.04 times respectively when cooperating with Greedy and Teammate-aware teammates as opposed to 67.77 and 63.88 times when mimicking their behavior. The improvements of planning over mimicking the teammates increase as the worlds get larger, although VI does not scale well enough computationally to calculate the optimal behavior for these worlds. For example, on the 20×20 world, using PLASTIC–Model with UCT allows the agent to capture the prey on average 15.07 times per 500 steps when cooperating with Greedy Probabilistic teammates compared to 6.12 times when mimicking the teammates' behavior. Similarly, the agent using PLASTIC–Model captures the prey 14.47 times rather than 2.60 times when paired with Probabilistic Destinations teammates. All differences are statistically significant.

However, using the approximate planning of UCT in PLASTIC–Model is not much of a compromise, since it performs nearly as well as VI despite using much less computation time. In the 5×5 world, the agent captures the prey 91.68 and 80.45 times with Greedy and Teammate-aware agents when planning with UCT, as opposed to 92.82 and 81.043 times with VI. The difference in performance could be lowered by using more playouts in the UCT at the cost of more computation time. Given the close approximation to optimal that UCT provides, the most important difference between the methods is the time it takes to plan. On the 5×5 world, an entire UCT episode takes less than 10 s compared to VI's 12 h computation (although VI only needs to run once, rather than for each episode). Furthermore, UCT is an anytime algorithm, so it can be used to handle variable time constraints and can modify its plan online as the models change. Given the good performance of UCT as well as its computational efficiency, we use it as the planning algorithm for PLASTIC–Model for the remainder of this section.

In summary, these tests show that PLASTIC–Model is effective for cooperating with known teammates. In addition, they show that using UCT as the planning algorithm for PLASTIC–Model leads to good performance of the team while remaining computationally efficient.

7.2.4 Cooperating with Teammates Drawn from a Known Set

While Sect. 7.2.3 considers the case in which the ad hoc agent knows the behaviors of its teammates, the ad hoc agent may not always be this well informed. Instead, ad hoc agents will need to adapt to new teammates on the fly. Therefore, we now expand the problem, considering the case in which the ad hoc agent may encounter any of the four hand-coded predators as teammates, but it does not know which behavior its current teammates are using. The ad hoc agent does know that these teammates are drawn from the set of hand-coded predators. In other words, PLASTIC–Model receives all four hand-coded behaviors as HandCodedKnowledge and needs to determine which one best represents its teammates online. This setting is closer to the general ad hoc teamwork scenario, because it shows how well an ad hoc agent can do if it only

knows that its teammates are drawn from a larger set A of possible teammates. These evaluations test whether PLASTIC–Model can determine which type of teammates it encounters and adapt to them. We hypothesize that PLASTIC–Model will outperform matching their behaviors and perform only marginally worse than when PLASTIC–Model knows their behaviors before interacting with them. In Sects. 7.2.5–7.2.7, we explore a setting with a much larger set of possible teammates.

If it has a set of possible models for its teammates, ideally PLASTIC–Model should be able to determine which model is correct and plan with that model appropriately. In this setting, PLASTIC–Model uses the polynomial weights method described in Sect. 5.2.1 to maintain its beliefs over the teammates' types. PLASTIC–Model is given a uniform prior over the teammate types for BehaviorPrior, but PLASTIC–Model knows that the teammates are homogeneous; i.e. there were no teams with some agents following the Greedy behavior and others following the Teammate-aware behavior. The results for this scenario are displayed in Fig. 7.5. Differentiating the deterministic teammate behaviors is straightforward because as soon as they take one action that is not expected by the deterministic behavior, the incorrect model can be removed. However, the stochastic teammate behaviors are more difficult to differentiate, as there is significant overlap in the actions that are possible for them to take.

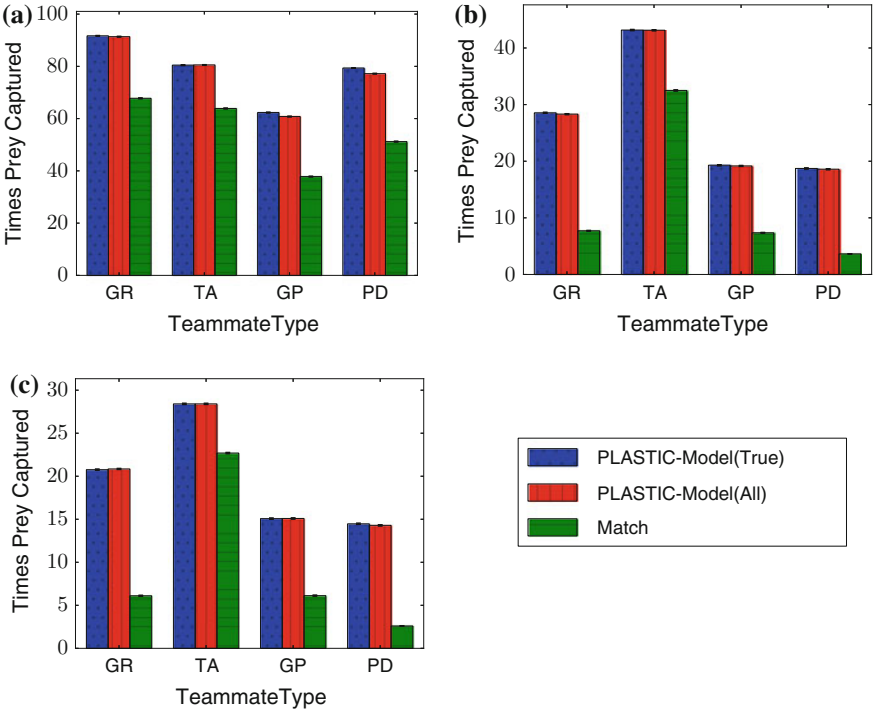

Fig. 7.5 Results with unknown hand-coded teammates. **a** 5 × 5 world. **b** 10 × 10 world. **c** 20 × 20 world

We compare PLASTIC–Model being given the four hand-coded teammate behaviors as HandCodedKnowledgeto a version of PLASTIC–Model that is given only the correct model of its teammates as HandCodedKnowledge. We keep the baseline of trying to fit into the teammates' pre-designed team, denoted Match. The results are shown in Fig. 7.5. PLASTIC–Model is statistically significantly better than Match in all scenarios. In the 5×5 world, PLASTIC–Model(All) is statistically significantly worse that PLASTIC–Model(True) for GR, GP, and PD teammates. In the 10×10 world, PLASTIC–Model(True) is significantly better than PLASTIC–Model(All) only for GR teammates, and in the 20×20 world, PLASTIC–Model(True) is significantly better than PLASTIC–Model(All) for the GR and PD teammates. These results show that PLASTIC–Model is able to quickly determine the behaviors of its teammates, losing only a small amount compared to when it knows the correct teammate behavior ahead of time. In summary, these results show that PLASTIC–Model can learn to cooperate with a number of unknown teammates given prior hand-coded models of its potential teammates' behaviors for HandCodedKnowledge.

7.2.5 Unmodeled Teammates

To this point, PLASTIC–Model has always had the benefit of having the correct model of its teammates in HandCodedKnowledge, even when HandCodedKnowledge includes incorrect models. However, PLASTIC–Model may not always be this fortunate. Therefore, we now consider the case where there are agents in A for which PLASTIC–Model does not have a correct model in HandCodedKnowledge. We again give PLASTIC–Model the four hand-coded teammate behaviors as HandCodedKnowledge, but the ad hoc agent encounters teammates not drawn from this set. To make sure we have not biased the creation of these agents, and that they truly are unknown, we used the externally-created teammates described in Sect. 3.2.2.2 as $\text{Student}_{\text{Broad}}$. Note that all the agents on each team used here are produced by the same student: we did not mix and match agents from different students. However, on some of the students' teams, not all of the agents use the same behavior. For this and all following tests, we focus on the 20×20 world because it is more complex and interesting than the small worlds. We hypothesize that PLASTIC–Model will be able to determine which models best fit its teammates and use them to plan to effectively cooperate with its teammates. Our expectation is that PLASTIC–Model will outperform matching their behaviors and be outperformed by planning when their true behavior is known.

As explained in depth in Sect. 5.2, PLASTIC–Model maintains the probabilities of the four known models and samples from this distribution while planning. While these models are not correct, PLASTIC–Model tries to determine which of these behaviors best matches how its current teammates are behaving.

We compare 3 possible strategies for the ad hoc agent:

1. **Match**—match the teammates' behaviors
2. **PLASTIC–Model(True)**—use PLASTIC–Model with the HandCodedKnowledge initialized to the current teammates' true behavior
3. **PLASTIC–Model(HC)**—use PLASTIC–Model with the 4 hand-coded models provided as HandCodedKnowledge

Strategies 1 and 2 require the ad hoc agent to know the true behavior of its current teammates, which is not always possible. These two strategies therefore serve as baselines to compare strategy 3, which represents the true ad hoc scenario of encountering previously unseen teammates. The results in Fig. 7.6 show that the ad hoc agents do quite well despite the incorrect models. All differences are statistically significant. For example, the PLASTIC–Model agent captures the prey 13.36 times per 500 steps rather than 9.97 times if it matched its teammate's behaviors. This result is surprising because one would assume that planning using an incorrect model would perform worse than playing the behavior of the student's agent that the ad hoc agent replaced. While there is some loss compared to if the ad hoc agent knew the true behavior of its teammates, these 4 hand-coded models are representative enough of these externally-created teammate to achieve good results.

This experiment shows that it is possible for an agent to cooperate with unknown teammates by using a set of known, representative models. In summary, these results demonstrate that PLASTIC–Model can cooperate with a variety of previously unseen teammates given a set of good hand-coded models as HandCodedKnowledge.

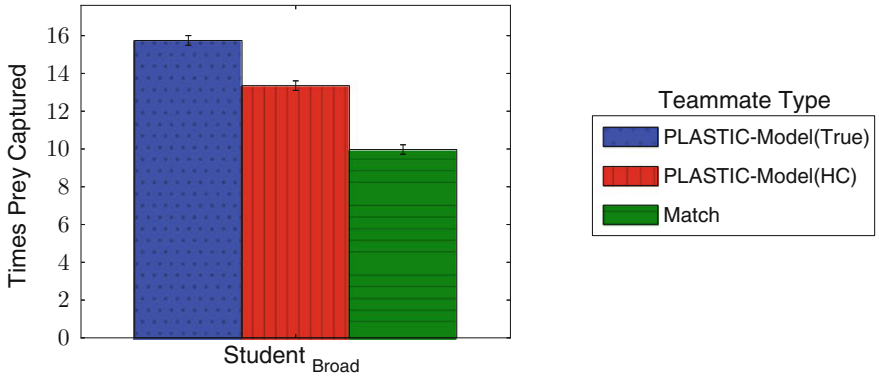

Fig. 7.6 Results with unobserved externally-created teams ($\text{Student}_{\text{Broad}}$) on a 20×20 world

7.2.6 Learning About Teammates

In this section, we explore the scenario where the ad hoc agent has previously observed a number of past teammates. These past teammates are expected to be similar to the current teammates. Ideally, the ad hoc agent should be able to use the observations of its past teammates to better cooperate with its current teammates. To this end, this section evaluates how well PLASTIC–Model learns models of past teammates and then selects between these models. Specifically, PLASTIC–Model observes each past teammate for a total of 50,000 steps. Then, PLASTIC–Model learns a decision tree to represent the behavior of each past teammate, as discussed in Sect. 5.2.3. This section tests the hypothesis that PLASTIC–Model can learn models of past teammates and reuse these learned models to cooperate with new teammates. The results show that this approach only marginally loses compared to knowing the teammates' true behaviors and outperforms matching their behaviors.

The teammates used in this section are those from $\text{Student}_{\text{Broad}}$, described in Sect. 3.2.2.2. These teammates are externally-created, being designed by students for a class assignment. We consider 5 behaviors for the ad hoc agent:

1. **Match**—match the teammates' behaviors
2. **PLASTIC–Model(True)**—use PLASTIC–Model with the HandCodedKnowledge initialized to the current teammates' true behavior
3. **PLASTIC–Model(CorrectLearned)**—use PLASTIC–Model with PriorTeammates being only the current teammates
4. **PLASTIC–Model(SetIncluding)**—use PLASTIC–Model with PriorTeammates including all 29 possible teammates from $\text{Student}_{\text{Broad}}$, including the current ones
5. **PLASTIC–Model(SetExcluding)**—use PLASTIC–Model with PriorTeammates including 28 possible teammates from $\text{Student}_{\text{Broad}}$, excluding the current ones

Once again, strategies 1 and 2 serve as baselines and require knowledge of the current teammates true behaviors. PLASTIC–Model(CorrectLearned) evaluates the performance of the learning algorithm, where PLASTIC–Model knows which teammates the agent is cooperating with and uses its past observations of these teammates to learn a model of them. PLASTIC–Model(SetIncluding) evaluates the more general ad hoc teamwork scenario where the current type of teammate is unknown, but the current teammates have been previously observed. Finally, PLASTIC–Model(SetExcluding) shows the true ad hoc teamwork scenario, when the ad hoc agent has never seen the current teammates, but uses PLASTIC–Model to reuse knowledge it has learned from previous teammates.

Figure 7.7 shows the performance of these five approaches; all differences are statistically significant. PLASTIC–Model(True) shows an unattainable level of performance as it requires perfect knowledge of the current teammates. However, learning a model by observing the current teammates does not lose too much performance, as shown by the PLASTIC–Model(CorrectLearned) line. Furthermore, having observed many teammates and needing to select from these past teammates does not generate too much loss either, as shown by the PLASTIC–Model(SetIncluding) line. Finally,

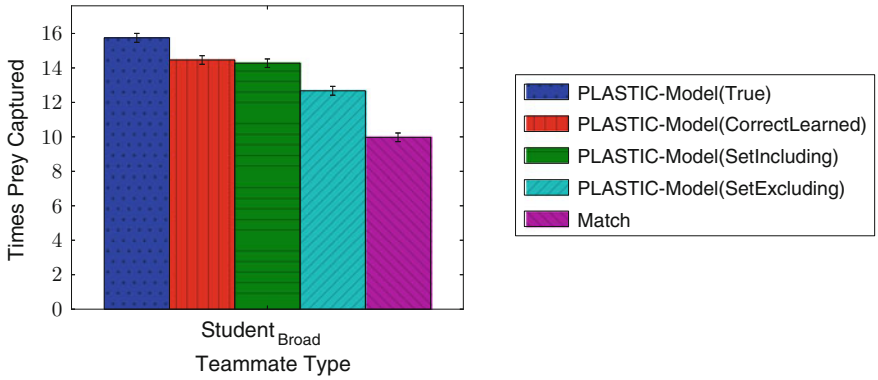

Fig. 7.7 Results with PLASTIC–Model learning models of previously observed teammates when encountering teams from $\text{Student}_{\text{Broad}}$ on a 20×20 world

PLASTIC–Model(SetExcluding) shows the performance of PLASTIC–Model when encountering a previously unseen teammate. Its performance shows that the models learned from previous teammates can do a good job of capturing the behavior of new teammates. This problem is the true ad hoc teamwork problem, when the ad hoc agent encounters teammates for which it has no prior knowledge. The gap between PLASTIC–Model(SetExcluding) and PLASTIC–Model(SetIncluding) shows that there is still room to improve for new teammates.

It is possible that the agents created by the class are biased to be similar, so all agents from $\text{Student}_{\text{Broad}}$ may share some characteristics. Therefore, we would also like to test how these learned models allow PLASTIC–Model to cooperate with teammates drawn from another set. In this scenario, we never learn models on the $\text{Student}_{\text{Selected}}$ teammates. Instead, we evaluate how well PLASTIC–Model performs when it is given $\text{Student}_{\text{Broad}}$ for PriorTeammates, but then encounters teammates from $\text{Student}_{\text{Selected}}$. Specifically, PLASTIC–Model learns 29 models, one for each teammate behavior in $\text{Student}_{\text{Broad}}$, but then encounters a 30th teammate, drawn from $\text{Student}_{\text{Selected}}$.

Figure 7.8 gives the results of these tests, with all differences being statistically significant. Once again, PLASTIC–Model(True) shows the upper bound on performance, when PLASTIC–Model is given information about the true behavior of the teammates, which is not accessible in most scenarios. However, PLASTIC–Model(SetExcluding) performs quite well, showing that the learned models are generally useful. This approach still far outperforms matching the teammates' behaviors, despite the inaccuracies of the models. For visual comparison, videos of ad hoc agents using PLASTIC–Model to adapt to its teammates and videos of ad hoc agents using other strategies can be found online.[1]

In summary, these results demonstrate that PLASTIC–Model can effectively learn models of past teammates and use these models to quickly adapt to unknown team-

[1] http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_Pursuit.

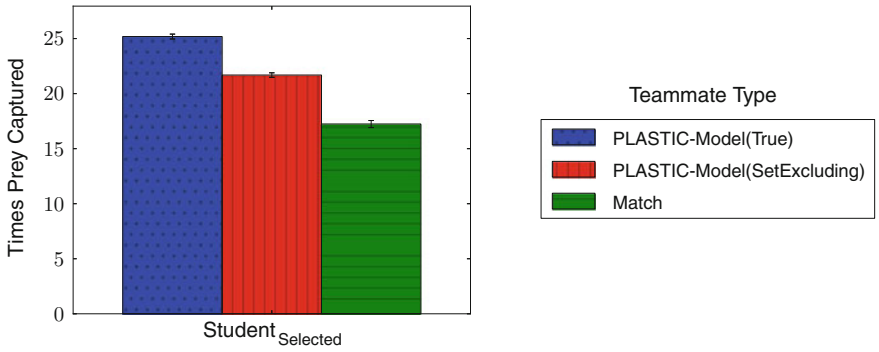

Fig. 7.8 Results with PLASTIC–Model learning models of previously observed teammates when encountering teams from $\text{Student}_{\text{Selected}}$ on a 20×20 world

mates. Furthermore, the results show that PLASTIC–Model is effective even when the new teammates are drawn from a substantially different set than its previous teammates.

7.2.7 Learning About New Teammates

The previous section assumes that PLASTIC–Model has observed previous teammates, but not the current teammates. If instead, PLASTIC–Model observes the current teammates for a small number of steps, it can try to use this information to learn a new model about these teammates. However, given that the learning is about the current teammates, we care about the speed of learning. Therefore, PLASTIC–Model combines this information with that coming from previously observed teammates to learn a better model. Specifically, this setting permits PLASTIC–Model to use transfer learning to learn a better model of its current teammates. These evaluations test the hypothesis that using TwoStageTransfer allows PLASTIC–Model to narrow the gap between PLASTIC–Model(SetExcluding) and PLASTIC–Model(CorrectLearned) seen in the previous section.

In our tests, we assume that the ad hoc agent has previously observed 50,000 training steps of each of the past 28 teammates from $\text{Student}_{\text{Broad}}$. In addition, it has seen 100 training steps of the current teammates. Note that this is significantly less than the testing time of 500 steps, but once testing begins, PLASTIC–Model is not learning online other than adapting its belief distribution over the possible models. PLASTIC–Model could also improve its models, but we focus on evaluating the transfer learning algorithms with a fixed amount of observations of the current teammates. Both the past and current teammates in this test are taken from $\text{Student}_{\text{Broad}}$.

To perform transfer learning, PLASTIC–Model uses the TwoStageTransfer algorithm introduced in Sect. 5.2.4. TwoStageTransfer allows the PLASTIC–Model to learn from a number of different past teammates (source data sets) combined with a few observations of the current teammates (the target data set). The goal of transfer learning is to produce a model that performs well on the target data set by using the source data sets. The advantage of TwoStageTransfer is that it considers that the data comes from several different types of past teammates, some of which are more similar to the current teammates than others. Specifically, TwoStageTransfer attempts to figure out the best weighting of data coming from each past teammate, where data coming from more similar teammates is given a higher weighting. In our tests, TwoStageTransfer considers 10 different weightings for each of the 28 past teammates from $\text{Student}_{\text{Broad}}$. While considering every possible weighting for each teammate would result in a total of 10^{28} weightings to consider. However, TwoStageTransfer approximates its search by choosing teammate weightings greedily, instead considering only $2 \times 10 \times 28 \approx 600$ possible weightings. This efficiency allows TwoStageTransfer to be computationally tractable on this problem.

To evaluate the performance of TwoStageTransfer, we compare it to using TwoStageTrAdaBoost, TrAdaBoost, and TrBagg (discussed in Sect. 3.1.7) in PLASTIC–Model. All of the transfer learning algorithms use decision trees as their base learning algorithm. Each algorithm has some set of parameters that can be tuned, and their values were chosen in preliminary tests based on their performance and computational tractability. To make the evaluations as fair as possible, for TwoStageTransfer and TwoStageTrAdaBoost, 10 different weightings were used. In TrAdaBoost and TwoStageTrAdaBoost, 10 boosting iterations were used. For TrBagg, a total of 1,000 sets were used for training classifiers, and a Naive Bayes classifier served as the fallback model.

Figure 7.9 shows the results of the four transfer learning algorithms used as subroutines of PLASTIC–Model, with all differences being statistically significant. In PLASTIC–Model, all learning of the models is performed offline with only model

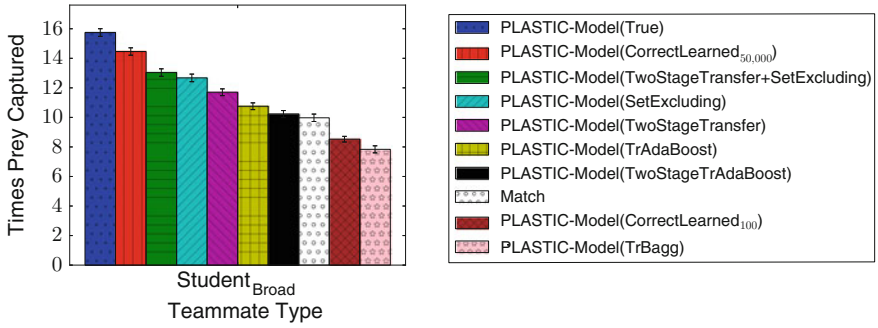

Fig. 7.9 Comparing different transfer learning algorithms in PLASTIC–Model to improve results with ad hoc agents that have limited observations of their current teammates. Tests are in a 20×20 world with $\text{Student}_{\text{Broad}}$ teammates

selection happening online during the evaluation. One baseline for comparison is if PLASTIC–Model ignores the previously observed teammates and learns a new model from just the observed 100 steps of the current teammates, shown as PLASTIC–Model(CorrectLearned$_{100}$). As an upper baseline, we compare to the unattainable performance of using a version of PLASTIC–Model that observes 50,000 steps of the current teammate, shown as PLASTIC–Model(CorrectLearned$_{50,000}$), which represents the best performance attainable using models learned given large amounts of data.

In these results, TwoStageTransfer statistically significantly outperforms the other transfer learning algorithms. In addition, combining the models learned with TwoStageTransfer with the models learned from representative teammates in the PLASTIC–Model(TwoStageTransfer + SetExcluding) setting helps, reaching results that are statistically significantly better than PLASTIC–Model(SetExcluding). TrBagg performed poorly in this setting, mis-transferring information, possibly due to the fallback model used or the balance of target and source data. Several values of these parameters were tested, but performance remained similar.

In addition, it is important to see how much target data TwoStageTransfer needs to perform well. Therefore, we vary the order of magnitude of target data and run PLASTIC–Model(TwoStageTransfer$_i$) where i is the amount of target data provided. Figure 7.10 shows results with varying amounts of target data, but constant amounts of source data. The difference between the results with 1,000 steps of target data and 100 is statistically significant, but the differences between 10,000 and 1,000 or 100 and 10 are not. The results show that the performance of TwoStageTransfer does improve with more target data, but the improvement is not smooth. These results show that as few observations as 10 steps of the current teammates are sufficient for TwoStageTransfer to perform produce useful models in this scenario. We used 100 steps in Fig. 7.9 to give other transfer learning methods enough data to perform adequately, though the results show that TwoStageTransfer still significantly outperforms them.

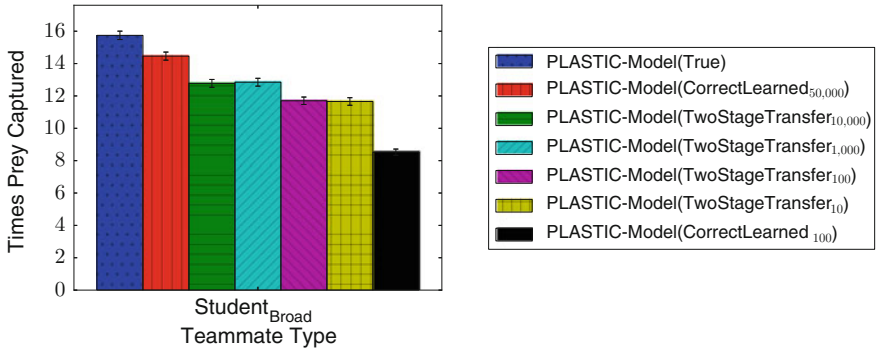

Fig. 7.10 Evaluating PLASTIC–Model using TwoStageTransfer with varying amounts of target data. Tests are in a 20 × 20 world with Student$_{Broad}$ teammates

In summary, TwoStageTransfer is effective for learning models of new teammates using only a small amount of observations of these teammate combined with many observations of past teammates. Using TwoStageTransfer with PLASTIC–Model allows an ad hoc agent to cooperate with a variety of unknown teammates, outperforming only reusing previously learned models.

7.2.8 Summary

This section showed that PLASTIC–Model enables ad hoc team agents to cooperate with a variety of hand-coded and externally-created teammates in the pursuit domain. PLASTIC–Model gets good results when given a set of hand-coded behaviors as HandCodedKnowledge or when it has experienced a number of previous teammates as PriorTeammates. PLASTIC–Model performs well even when it has never seen the current teammates before. Furthermore, TwoStageTransfer is effective for creating improved models for PLASTIC–Model to plan with, outperforming existing transfer learning algorithms. This result is due to TwoStageTransfer's exploitation of the knowledge that some past teammates are more similar to the current teammates than others and weighting data coming from these past teammates accordingly. PLASTIC–Model allows for effective ad hoc teams in the pursuit domain.

7.3 Half Field Offense

The previous section showed that PLASTIC allows ad hoc agents to cooperate with a variety of teammates in the pursuit domain. However, while the pursuit domain requires a number of agents to cooperate, it is still simple compared to many realistic scenarios. Therefore, this section looks into scaling PLASTIC to a more complex domain, namely that of half field offense (HFO), described in Sect. 3.2.3. All past research on HFO has focused on creating full teams that are pre-coordinated, but this section shows that PLASTIC can handle unknown teammates without prior coordination. Given the complexity of HFO, planning using UCT requires many samples and runs into issues with imperfect modeling of the environment. Therefore, we evaluate PLASTIC–Policy in HFO. PLASTIC–Policy is more effective because it avoids the complexity of modeling the domain and teammates. Instead, PLASTIC–Policy directly learns policies for cooperating with previous teammates and then selects between these policies online for the current teammates. PLASTIC–Policy is described in depth in Sect. 5.3.

7.3.1 Grounding the Model

Before we discuss how to learn or act in HFO, it is important to understand how we model the problem. Therefore, this section describes how we model the HFO domain as an MDP.

7.3.1.1 State

A state $s \in S$ describes the current positions, orientations, and velocities of the agents as well as the position and velocity of the ball. In this book, we use the noiseless versions of these values to permit for simpler learning.

7.3.1.2 Actions

In the 2D simulation league, agents act by selecting whether to dash, turn, or kick and specify values such as the power and angle to kick at. Combining these actions to accomplish the desired results is a difficult problem. Therefore, this book builds on the code release by Helios [8]. This code release provides a number of high level actions, such as passing, shooting, or moving to a specified point.

We use 6 high level actions when the agent has the ball:

1. Shoot—shoot the ball at the goal, avoiding any opponents
2. Short dribble—dribble the ball while maintaining control
3. Long dribble—kick ball and chase it
4. Pass_0—pass to teammate 0
5. Pass_1—pass to teammate 1
6. Pass_2—pass to teammate 2

Each action considers a number of possible movements of the ball and evaluates their effectiveness given the locations of the agent's opponents and teammates. Each action therefore represents a number of possible actions that are reduced to discrete actions using the agent2d evaluation function. While using these high level actions restricts the possibilities that the agent can take, it also enables the agent to learn more quickly and prune out ineffective actions, allowing it to select more intelligent actions with fewer samples.

Additionally, the agent can select how it moves when it is away from the ball. As the agent can take a continuous turn action or a continuous dash action every time step, it is helpful to again use a set of high level actions, in this case 7:

1. Stay in the current position
2. Move towards the ball
3. Move towards the opposing goal
4. Move towards the nearest teammate
5. Move away from the nearest teammate
6. Move towards the nearest opponent
7. Move away from the nearest opponent

These actions provide the agent a number of possible actions that adapt to its changing environment, while constraining the number of possible actions.

7.3.1.3 Transition Function

The transition function is defined by a combination of the simulated physics of the domain as well as the actions selected by the other agents. The agent does not directly model this function; instead, it stores samples observed from played games as described in Sect. 7.3.2.

7.3.1.4 Reward Function

The reward function is 1,000 when the offense wins, −1,000 when the defense wins, and −1 per each time step taken in the episode. The value of 1,000 is chosen to be greater than the effects of step rewards over the whole episode, but not so great as to completely outweigh these effects. Other values were tested with similar results.

7.3.2 Methods

PLASTIC–Policy learns policies for cooperating with each previously encountered team. In this book, we use Fitted Q Iteration (FQI), introduced by Ernst et al. [9]. We treat an action as going from when an agent has possession of the ball until the action ends, another agent holds the ball, or the episode has ended. Given that we only control a single agent, the teammates follow their own policies. The agent collects data about its actions and those of its teammates' in the form $\langle s, a, r, s' \rangle$ where the a is our agent's actions. The agent does not directly store the actions of its teammates, instead storing the resulting world states, which include the effects of its teammates' actions. If we controlled all of the agents, we would also consider the action from the teammates' perspectives. The agent observes 100,000 episodes of HFO with each type of teammate. These episodes contain the agent's actions when the agent has the ball as well as when it is away from the ball.

There are many ways to represent the state of a game of half field offense. Ideally, we want a compact representation that allows the agent to learn quickly by generalizing its knowledge about a state to similar states without over-constraining the policy. Therefore, we select 20 features given that there are 3 teammates:

- X position—the agent's x position on the field
- Y position—the agent's y position on the field
- Orientation—the direction that the agent is facing
- Goal opening angle—the size of the largest open angle of the agent to the goal, shown as θ_g in Fig. 7.11
- Teammate i's goal opening angle—the teammate's goal opening angle
- Distance to opponent—distance to the closest opponent

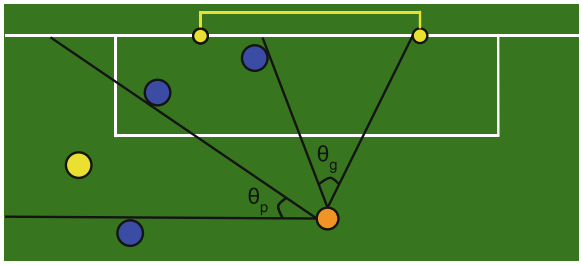

Fig. 7.11 Open angle from ball to the goal avoiding the *blue goalie* and the open angle from the ball to the *yellow teammate*

- Distance from teammate i to opponent—the distance from the teammate to the closest opponent
- Pass opening angle i—the open angle available to pass to the teammate, shown as θ_p in Fig. 7.11

7.3.3 Evaluation Setup

Results are averaged over 1,000 trials, each consisting of a series of games of half field offense. In each trial, the agent is placed on a team randomly selected from the 7 teams described in Sect. 3.2.3.1. Performance is measured by the fraction of the time that the resulting team scores.

In this book, we use two variations on the HFO task: (1) the *limited version* with two offensive players attempting to score on two defenders (including the goalie) and (2) the *full version* with four attackers attempting to score on five defenders. In order to run some existing teams used in the RoboCup competition, it is necessary to field the entire 11 player team for the agents to behave correctly. Therefore, it is necessary to create the entire team and then constrain the additional players to stay away from play, only using the agents needed for half field offense. This approach may alter the behavior of the players used in the HFO, but our initial tests suggested that the resulting teams still perform well on the task. We choose a fixed set of player numbers for the teammates, based on which player numbers tended to play offensive positions in observed play. The defensive players use the behavior created by Helios in the limited version of HFO. In the full HFO, the defense uses the agent2d behavior provided in the code release by Helios [8].

We compare several strategies for selecting from the policies learned by playing with previously encountered teammates. The performance is bounded above by the Correct Policy line, where the agent knows its teammate's behavior type and therefore which policy to use. The lower bound on performance is given by the Random Policy line, where the agent randomly selects which policy to use. The Combined Policy line shows the performance if the agent learns a single policy using the data collected from all possible teammates, representing what an agent might do if treating this as a single angle learning problem instead of an ad hoc teamwork problem.

We then compare two more intelligent methods for selecting models, as described in Sect. 5.3.2. Specifically, our agent must decide which of the 7 policies to follow as it does not know its new teammate's behavior type. The Bandit line represents PLASTIC–Policy that uses an ϵ-greedy bandit algorithm to select policies. Other bandit algorithms were tested as were other values of ϵ, but ϵ-greedy with $\epsilon = 0.1$ linearly decreasing to 0 over the length of the trial outperformed these other methods. The PLASTIC–Policy line shows the performance of our approach, using loss-bounded Bayesian updates to maintain probabilities over which previously learned policy to use. We set $\eta = 0.1$ for updating the probabilities of the models in Eq. 4. We model the noise in predicting actions using a normal distribution. This noise affects the loss function by controlling the probability function $P(\text{actions}|\text{model})$. For differences in distance predictions, we use $\sigma = 4.0$, and, for orientation differences, we use $\sigma = 40°$.

7.3.4 Limited Half Field Offense

Our first set of results are in the limited version of the HFO game which uses 2 offensive players competing against 2 defenders (including the goalie). Therefore, the agent only needs to adapt to a single teammate. This limited version of the problem reduces the number of state features to 8 and the number of actions while holding the ball to 4, while the number of actions away from the ball stays at 7. These evaluations test the hypothesis that PLASTIC–Policy can quickly converge to selecting the best policy, losing only a small amount compared to the correct policy. In addition, we hypothesize that PLASTIC–Policy will converge much faster than the bandit-based approach and will also outperform combining the data from all of the agents to learn a single, combined policy. The results are shown in Fig. 7.12, with the error bars showing the standard error.

The difference between the Correct Policy and Random Policy lines shows that selecting the correct policy to use is important for the agent to adapt to its teammates. The gap between the Correct Policy and Combined Policy shows that knowing the correct teammate is better that grouping all teammates together. While the Bandit line does not show much learning in Fig. 7.12, it does continue learning over time. Its performance converges to scoring 0.418 of the time after approximately 10,000 episodes, though it does score approximately equal to the combined policy (0.382) after 1,750 episodes. Its slow speed is due to the fact that its observations are noisy estimates of the policies' effectiveness and are only received after each game of HFO. In addition, the scoring fractions of the different strategies are fairly close, so determining the best one given the amount of noise is difficult.

On the other hand, the PLASTIC–Policy line shows fast improvement, converging to the performance of the Correct Policy line. This quick adaptation is due to two factors: (1) the better estimations of which policy fits the teammates and (2) the

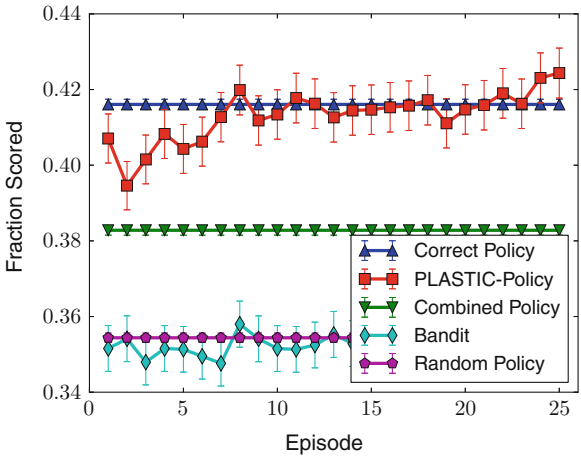

Fig. 7.12 Scoring frequency in the limited half field offense task

frequency of the updates. The estimations of the probabilities are better as they measure how each agent moves, rather than only using a noisy estimate of how the policy performs. The updates are performed after every action rather than after each episode; so updates are much more frequent. These two factors combine to result in fast adaptation to new teammates using PLASTIC–Policy. The differences between the performance of PLASTIC–Policy and Combined Policy and Bandit are statistically significant using a two population binomial test with $p < 0.01$ for all episodes shown in Fig. 7.12. Videos of the performance of PLASTIC–Policy compared to other strategies can be viewed online.[2]

To understand the learning of PLASTIC–Policy, it is useful to look at its beliefs, shown in Fig. 7.13. This graph shows the probability of the correct model of the current teammates as well as the probability that correct model has the highest probability (with ties contributing a probability of $\frac{1}{\#tied}$). While the probability of the correct model takes over 15 episodes to reach above 90 % probability, the correct model becomes the maximal model 90 % of the time after just 5 episodes. This result explains why taking the maximal model gives such good performance in PLASTIC–Policy. Note that choosing the maximal model does not create premature convergence because each action the teammates take allows PLASTIC–Policy to update the probability of those teammates.

In summary, the results in this section show that PLASTIC–Policy is effective for cooperating with unknown teammates on a complex domain with continuous state and continuous actions. PLASTIC–Policy is able to learn policies for cooperating with previous teammates and quickly select from these policies to efficiently cooperate with new teammates.

[2] http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_HFO.

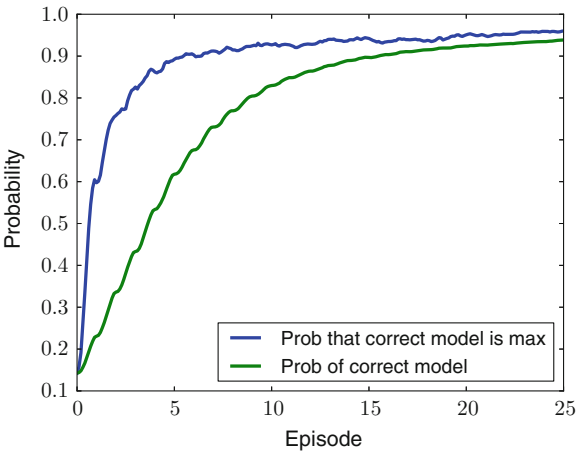

Fig. 7.13 Belief of the probability of the correct model ($P(m^*|s,a)$) and probability of the correct model having the highest probability ($P(p^* = \max p_i|s,a)$) calculated by PLASTIC–Policy in the limited HFO task

7.3.5 Full Half Field Offense

Our second set of results are in the full HFO game with 4 offensive players versus 5 defenders (including the goalie). In this setting, our agent needs to adapt to its three teammates to score against the five defenders. This setting tests the hypothesis that PLASTIC–Policy can learn intelligent policies for cooperating with its three teammates and quickly select between these policies when cooperating with unknown teammates. We expect that PLASTIC–Policy will outperform selecting policies using a bandit-based approach or learning a single policy to cooperate with all teammates. In addition, we hypothesize that PLASTIC–Policy will only lose marginally compared to the gold standard of knowing the best policy before interacting with its teammates.

The results for this setting are shown in Fig. 7.14. As in Sect. 7.3.4, the upper bound on performance is given by Correct Policy and the lower bound is given by Random Policy. The Bandit setting learns slowly, reaching a performance of 0.357 after approximately 20,000 episodes. It outperforms the combined policy (0.350) after 12,000 episodes. Once again, PLASTIC–Policy quickly converges to the correct policy's performance, outperforming the Bandit and Combined lines. These results show that PLASTIC–Policy quickly learns to cooperate with unknown teammates. Using a two population binomial test with $p < 0.05$, PLASTIC–Policy's performance is stastically significantly better than Combined Policy and Bandit from episode 3 on. For visual comparison, videos of ad hoc agents using PLASTIC–Policy and other strategies to cooperate with its teammates can be viewed online.[3]

We again look at PLASTIC–Policy's beliefs over time in Fig. 7.15. In this figure, we can see that PLASTIC–Policy takes several episodes to be convinced that the correct model is the true model of the current teammates because of the noise of the whole team's actions. However, greedily selecting the highest probability model's

[3] http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_HFO.

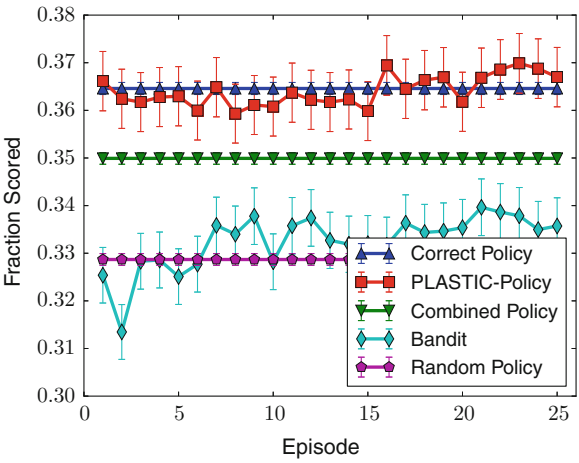

Fig. 7.14 Scoring frequency in the full half field offense task

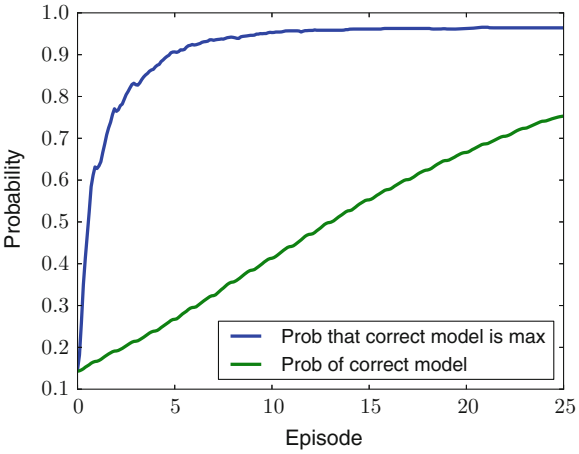

Fig. 7.15 Belief of the probability of the correct model ($P(m^*|s,a)$) and probability of the correct model having the highest probability ($P(p^* = \max p_i|s,a)$) by PLASTIC–Policy in the full HFO task

corresponding policy performs well because the correct model is maximal 90 % of the time after just 5 episodes. This result shows that PLASTIC–Policy learns quickly and can take advantage of its continuing exploration of its teammates despite only selecting what it believes in the current best policy.

In summary, the results in this section demonstrate that PLASTIC–Policy can scale to complex domains requiring coordinating with many teammates, continuous states, and continuous actions. PLASTIC–Policy can efficiently select good policies for cooperating with its current teammates from a set of policies learned for cooperating with past teammates.

7.3.6 Summary

The results in this section show that PLASTIC–Policy is effective for ad hoc team agents in the HFO domain. Our tests evaluate PLASTIC–Policy with teams that competed in the 2013 RoboCup 2D Simulation League and are complex, being developed over several years. Despite this complexity, PLASTIC–Policy is able to learn policies about each type of teammate, and the results show that these policies are specialized to the teammate type. Therefore, PLASTIC–Policy's approach of maintaining the probabilities of each teammate type and selecting the best policy significantly outperforms the other approaches. This section shows that PLASTIC can scale to complex domains.

7.4 Chapter Summary

This chapter presents the empirical analysis of PLASTIC in the bandit domain, the pursuit domain, and half field offense in the 2D RoboCup domain. The results reported here show that both PLASTIC–Model and PLASTIC–Policy enable ad hoc team agents to cooperate with a variety of teammates. Our tests focus on whether PLASTIC can cooperate with *externally-created* teammates, which are not pre-designed for ad hoc teamwork.

An overview of the experiments can be seen in Table 7.1. The results in Sect. 7.1 show that PLASTIC can plan how to effectively communicate with its teammates in the bandit domain. These results also show that PLASTIC can operate effectively with parameterized hand-coded models for HandCodedKnowledge. Section 7.2 shows that PLASTIC can learn models of past teammates and use these models to quickly adapt to new teammates. In addition, the results in Sect. 7.2.7 show that TwoStage-Transfer significantly outperforms other transfer learning algorithms for learning models of new teammates because it takes advantage of the fact that some past teammates are more similar to the current teammates than others. Finally, Sect. 7.3 describes the results in the HFO domain. These results show that PLASTIC can scale to complex domains, where teams of developers work for years to develop smart agents. This chapter shows that PLASTIC can reason about communication, select from a set of parameterized hand-coded models for HandCodedKnowledge, learn models of its past teammates, use transfer learning to improve the models it learns, and perform well in complex domains.

References

1. Barrett, Samuel, Noa Agmon, Noam Hazon, Sarit Kraus, and Peter Stone. 2014. Communicating with unknown teammates. In *Proceedings of the twenty-first European conference on artificial intelligence*, Aug 2014.
2. Barrett, Samuel, and Peter Stone. 2014. Cooperating with unknown teammates in robot soccer. In *AAAI workshop on multiagent interaction without prior coordination* (*MIPC 2014*), July 2014.
3. Barrett, Samuel, Peter Stone, and Sarit Kraus. 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proceedings of the tenth international conference on autonomous agents and multiagent systems* (*AAMAS*), May 2011.
4. Barrett, Samuel, Peter Stone, Sarit Kraus, and Avi Rosenfeld. 2013. Teamwork with limited knowledge of teammates. In *Proceedings of the twenty-seventh conference on artificial intelligence* (*AAAI*), July 2013.
5. Barrett, Samuel, and Peter Stone. 2015. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Proceedings of the twenty-ninth conference on artificial intelligence* (*AAAI*), Jan 2015.
6. Silver, David, and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems 23* (*NIPS*).
7. Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11: 10–18.
8. Akiyama, Hidehisa. 2010. Agent2d base code release. http://sourceforge.jp/projects/rctools.
9. Ernst, Damien, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* (*JMLR*) 6: 503–556.

Chapter 8
Discussion and Conclusion

Past research on multiagent teams assumes that the agents share common communication and coordination protocols, and research on other multiagent systems focus on the case where the agents' goals are diametrically opposed and other agents are treated as opponents. This book instead looks at a multiagent system where agents are brought together to form a new team without prior coordination. This setting is known as *ad hoc teamwork* and addresses the case where agents need to adapt to teammates that they do not know. We believe that this problem is important for deploying robots into real world scenarios because they may encounter robots from other companies or research laboratories that may not share a communication or coordination protocol. For agents to act intelligently in this setting, they need to learn and adapt to their teammates quickly. We believe that this problem is of growing importance due to the increasing durability, increasing capabilities, and decreasing cost of intelligent robots.

Specifically, we believe that to enable ad hoc agents to be viable for real world scenarios, their algorithms must be robust to their teammates' behaviors, robust to diverse tasks, and adapt quickly. Therefore, this book presents an algorithm, PLASTIC, that addresses all three of these requirements. PLASTIC reuses knowledge learned from past teammates and combines this knowledge with any advice provided by domain experts. This approach allows PLASTIC to quickly adapt to new teammates on the fly. We show that PLASTIC performs well on three disparate domains with a variety of teammates and differing amounts of knowledge about its teammates. In addition, we introduce a transfer learning algorithm, TwoStageTransfer, that improves the speed of adaptation, outperforming existing transfer learning algorithms. Furthermore, we analyze the ad hoc teamwork problem in the bandit domain that includes communication and show that it is theoretically tractable as it takes only polynomial time to calculate an ε-optimal behavior. Finally, we identify three dimensions that we believe describe ad hoc teamwork problems in an informative way. We hypothesize that these dimensions will allow researchers to identify which algorithms to use on new problems. In this book, we use these dimensions to analyze existing research and identify areas for future research.

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4_8

This chapter summarizes the book and identifies avenues for future research. We begin by summarizing the PLASTIC algorithm and the rest of the book in Sect. 8.1. Section 8.2 presents the contributions of this book. Then, Sect. 8.3 discusses the applicability of PLASTIC as well as its limitations, and Sect. 8.4 presents directions for future work that address these limitations. Finally, Sect. 8.5 provides some concluding remarks.

8.1 Summary

This book introduces the PLASTIC algorithm for cooperating with unknown teammates in ad hoc teamwork scenarios. PLASTIC exploits any prior knowledge it has learned about previous teammates in addition to any expert-provided knowledge to speed up its adaptation. PLASTIC is described in detail in Chap. 5. Our tests show that reusing this knowledge allows PLASTIC to quickly adapt to new teammates. We show that two versions of PLASTIC are effective: PLASTIC–Model, a model-based approach, and PLASTIC–Policy, a policy-based approach. We discuss which algorithm is expected to be more effective on new domains in Sect. 8.3. In summary, PLASTIC–Policy is more effective when the domain or teammates are difficult to model and plan about, and PLASTIC–Model is preferred in other scenarios due to being able to plan over a distribution of beliefs about the teammates' behaviors.

In Chap. 7, we empirically test PLASTIC–Model and PLASTIC–Policy on three very different domains while varying the amount of information that is available about the current teammates. These domains provide a variety of interesting problems including *communicating* with unknown teammates, requiring coordinated actions, and having continuous state and action spaces. We consider a wide variety of *externally-created* teammates that are not designed for ad hoc teamwork. These results show the effectiveness of PLASTIC; PLASTIC quickly adapts to new teammates and significantly outperforms existing approaches.

In addition, we analyze the computational complexity of applying PLASTIC–Model on the bandit domain that includes *communication*. These theoretical analyses are presented in Chap. 6, and they show that the bandit domain is computationally tractable, i.e. ε-optimally solvable in polynomial time.

Furthermore, we introduce a transfer learning algorithm, TwoStageTransfer, and show that it helps PLASTIC–Model adapt more quickly to new teammates. TwoStageTransfer allows the agent to transfer knowledge from many past teammates while exploiting the idea that some past teammates are more similar to the current teammates than others. TwoStageTransfer is a general transfer learning algorithm, and we empirically test it for learning models of new teammates that are used by PLASTIC–Model on the pursuit domain. These results show that TwoStageTransfer is computationally efficient and outperforms existing transfer learning algorithms. The full TwoStageTransfer algorithm is presented in Sect. 5.2.4 and results with it can be found in Sect. 7.2.7.

After describing the general ad hoc teamwork problem and the evaluation framework used to analyze ad hoc team agents in the beginning of Chap. 2, we go on to introduce three dimensions in Sect. 2.3 that we believe best describe ad hoc teamwork problems. These dimensions describe the ad hoc agent's knowledge of the team, its knowledge about the environment, and the how reactive its teammates are to its actions. We believe that domains with similar values along these dimensions can be solved by similar algorithms, and we apply these dimensions to analyze the problems explored in this book in Sect. 3.2. In addition, we use these dimensions to analyze related ad hoc teamwork research in Sect. 4.4.4. We also discuss other prior related multiagent research in the rest of Chap. 4.

8.2 Contributions

In summary, this book provides the following six major contributions to the field:

1. *PLASTIC*: This book introduces the PLASTIC algorithm and instantiates it in a model-based approach called PLASTIC–Model and a policy-based approach called PLASTIC–Policy. These algorithms are the first algorithms for ad hoc teamwork that allow ad hoc agents to learn about previous teammates and reuse this knowledge to cooperate with new, unknown teammates on a variety of domains. In addition, these algorithms allow domain experts to provide information about potential teammates to speed up adaptation.
2. *Theoretical Analysis*: We provide theoretical analysis of PLASTIC–Model in the multi-armed bandit domain described in Sect. 3.2.1. This analysis proves that ε-optimally selecting between a parameterized set of models of teammates' behaviors and planning reactions to these teammates takes polynomial computation even when the distributions of the arms are unknown. Note that this analysis includes communication with known messages, including the case where the ad hoc agent is uncertain about how its teammates will react to messages.
3. *Reasoning about Communication*: Past research in ad hoc teamwork focuses either on the case where there is no communication or the case where there are completely shared protocols for coordination and communication. However, this book considers the case where messages' meanings are known, but the other agents' reactions to these messages is unknown. This book proves that reasoning about this type of communication using PLASTIC–Model is computationally tractable on the bandit domain described in Sect. 3.2.1. In addition, the book shows that PLASTIC–Model empirically outperforms other approaches on the bandit domain, using this communication in an efficient manner.
4. *TwoStageTransfer*: In this book, we introduce TwoStageTransfer, a new transfer learning algorithm. TwoStageTransfer improves over existing transfer learning algorithms because it efficiently transfers information from many different source data sets, while using the information of which source set data came from. This approach allows TwoStageTransfer to consider how similar each source data set

is to the target data set and transfer information from these sets accordingly. TwoStageTransfer is a general transfer learning algorithm, and we empirically test it in the area of ad hoc teamwork. We test whether TwoStageTransfer can transfer knowledge about previous teammates to quickly build new models of the current teammates.

5. *Empirical Evaluation*: We empirically test PLASTIC on the three domains presented in Sect. 3.2, namely the bandit domain, the pursuit domain, and the half field offense task in the 2D simulated soccer domain. These domains are significantly different, showing that PLASTIC can perform well in domains that provide communication, require coordinated multiagent actions, and have continuous state and action spaces. We show that PLASTIC outperforms other approaches. Combining information learned from previous teammates with expert-provided information allows PLASTIC to quickly adapt to new teammates, using both the PLASTIC–Model and PLASTIC–Policy versions of the algorithm.
6. *Taxonomy of Ad Hoc Teamwork*: We present what we believe are the three most important dimensions for describing ad hoc teamwork problems. We believe that domains with similar values along these dimensions can be solved using similar approaches, while domains with very different values may be best solved using different approaches. This book analyzes where the three domains we use fall along these dimensions. Furthermore, we analyze the related ad hoc teamwork research to identify areas for future research, which we discuss further in Sect. 8.4.

8.3 Discussion

The empirical evaluations in Chap. 7 show that PLASTIC is effective in a variety domains. However, there are limitations to where it can be effectively applied. One such limitation is that if there is a complete, shared communication protocol that allows agents to schedule their tasks and negotiate, PLASTIC is not the best approach. In these settings, standard multiagent team coordination algorithms (e.g. STEAM [1] and GPGP [2]) will serve better because they are designed to handle this scenario. In Sect. 7.1, we show that PLASTIC performs well when there is limited communication, but with full communication there are more applicable approaches.

PLASTIC–Model is only performs well in situations in which it can effectively model the domain. In complex domains, errors in the models can lead to issues with using planning algorithms to calculate the best responding behavior. This issue may be addressed by using a more robust planning algorithm, but in preliminary tests in the HFO domain, PLASTIC–Model performed poorly. However, this poor performance may also be due to the computational costs of PLASTIC–Model. Given that the implementation of PLASTIC–Model used in this book relies on Upper Confidence bounds for Trees (UCT) for planning, the computational costs may be high. Running UCT requires running many forwards simulations, and, if these simulations are expensive to calculate, it is expensive to run enough simulations to calculate which

actions perform best. However, PLASTIC–Policy performs well on these more complex domains.

Another constraint of PLASTIC is that it relies on either knowledge about past teammates or expert knowledge about potential teammates. Relying on this initial knowledge also means that PLASTIC may not perform well with teammates that differ greatly from previous teammates. In this case, the best approach might be to throw away any knowledge about previous teammates and learn about the problem using existing learning algorithms such as Q-learning [3], Fitted Q Iteration [4], or policy search [5]. Albrecht and Ramamoorthy [6] look at the performance of an algorithm that is similar to PLASTIC and prove constraints about the performance of their algorithm given errors in their teammate models. We believe that in most cases, there will be similarities between previous teammates and the current teammates, so reusing this information via PLASTIC will be effective. However, there are scenarios in which this will not be the case, for example if there are two very different approaches to solving the task and only agents using one approach have been encountered previously.

Fortunately, we believe that PLASTIC does apply in most scenarios. PLASTIC allows an ad hoc agent to quickly adapt to new teammates by reusing information about previous teammates or expert knowledge. We believe that this information will be available in many scenarios. PLASTIC will perform especially well when there are a limited number of useful behaviors that teammates may follow. When it has complete prior knowledge about potential teammates, PLASTIC can calculate the optimal behavior rather than just trying to fit into the team and copy its teammates' behaviors. When this prior knowledge is limited but good, PLASTIC can still perform effectively.

Another important consideration in PLASTIC is whether to use a model-based approach (PLASTIC–Model) or a policy-based approach (PLASTIC–Policy). PLASTIC–Model is effective when online planning algorithms like Upper Confidence bounds for Trees (UCT) [7] perform well. Using PLASTIC–Model allows an agent to calculate the best response to its current beliefs over its teammates behavior. On the other hand, PLASTIC–Policy only allows the agent to use the most likely policy because combining a weighted set of policies into a single policy does not have a clear online solution. It is possible to pre-calculate an effective behavior for a specific set of beliefs, but it is too expensive to pre-calculate policies for all possible beliefs. Therefore, we believe that it is desirable to use PLASTIC–Model where possible.

Applying PLASTIC–Model is not effective when the domain is complex to model or very noisy. Complex models take a significant amount of computation time, which limits the effectiveness of online planners such as UCT. Similarly, having noise in the effects of actions, observations, or teammates' behaviors leads to more complexity in planning. Online planners need to consider the range of possible outcomes to determine the best behavior. In addition, imperfect models can lead planners to calculate behaviors that do not work well in practice. In these scenarios, we expect PLASTIC–Policy to be more effective because it directly learns policies for coop-

erating with teammates in the domain and does not rely on building a model of the world and its teammates. Testing this hypothesis remains an area for future research.

The following section describes ways to extend PLASTIC, ensuring good performance in even more scenarios. These extensions apply to both PLASTIC–Model and PLASTIC–Policy.

8.4 Future Work

The research in this book leads to many interesting directions for future research. This section discusses five of these directions. The first is to investigate ad hoc teamwork in complex robotic tasks, discussed in Sect. 8.4.1. Analyzing the related ad hoc team research in Sect. 4.4.4 using the dimensions introduced in Sect. 2.3 leads us to observe that past ad hoc team research focuses on the case where the domain is known. Therefore, Sect. 8.4.2 discusses an important area for future research: considering ad hoc team agents that simultaneously learn about the domain and their teammates. Section 8.4.3 explains how to extend ad hoc teamwork research towards cooperating with both human and artificial teammates interchangeably. Then, Sect. 8.4.4 explores approaches to handling ad hoc teamwork with ambiguous communication, where the language itself must be learned. Finally, we discuss how ad hoc teamwork motivates more advances in transfer learning in Sect. 8.4.5.

8.4.1 Robotic Tasks

The most immediate area for future work on PLASTIC is trying it on larger, more complex domains. While half field offense in the simulated 2D RoboCup domain is complex, it still abstracts away many issues that arise on robots. Dealing with robots requires dealing with a large amount of noise in both perception and actuation. Furthermore, the action space of robots is higher than that of the 2D simulated robots.

Given the complexities of these domains, we expect that using a policy-based approach, such as PLASTIC–Policy, will work better than a model-based approach. However, instead of learning policies with algorithms such as fitted Q iteration as we did in the HFO domain, it may be more effective to use policy search. Research on robotics has shown that policy search techniques can perform well, scaling to complex domains [5]. We expect that similar approaches can determine intelligent policies for interacting with teammates, though the difficulties of these problems may require high level abstractions such as those used in Sect. 7.3.

Given these policies, it is still complex to select between them. While bandit approaches can be used to select between policies (as described in Sect. 5.3.2), these noisy domains will make determining the mean performance of the policies difficult. Instead, we still recommend using a Bayesian-based approach, such as the polynomial weights approach described in Sect. 5.3.2. However, building a model

of the teammates' behaviors is difficult, so using an approximate, nearest neighbors algorithm to represent the expected next states may be more effective as in Sect. 7.3.3.

Unfortunately, due to noisiness of actions and perceptions, the probability updates will also be significantly noisier. This issue will slow the convergence to the best policy. In order to compensate for this issue, it may be necessary to take actions to specifically differentiate which type of teammate that the ad hoc agent is dealing with. This idea corresponds to reasoning about the value of information [8]. For example, Dearden et al. [9] investigated the value of information for controlling exploration in reinforcement learning. However, their methods were computationally intensive and could only be applied to small domains. On the other hand, Ross et al.'s work [10] on POMDPs provide a more efficient approach to tracking beliefs about the environment through particle filtering. Reasoning about the value of information should allow the ad hoc agent to more quickly understand its teammates' behaviors.

In summary, future research into ad hoc teamwork should include work on scaling ad hoc teamwork to complex robotic domains. We expect that policy-based approaches, like PLASTIC–Policy, will perform well in these settings, especially when using policy search algorithms to build effective policies for cooperating with teammates and reasoning about the value of information to decide which policy to use with new teammates.

8.4.2 Learning About the Environment

Our analysis of the dimensions of the related ad hoc teamwork research in Sect. 4.4.4 indicates that most research has focused on the case where the ad hoc agent knows the full dynamics of the domain it is in. This book includes a small exception to that in the theoretical and empirical analysis in the bandit domain in Sects. 6.5 and 7.1.5. However, much more research on handling unknown environments is necessary for ad hoc agents to be ready to handle the changing, unknown environments encountered in the real world. We expect that models of the environment will not be good enough to directly plan on, so the ad hoc agent will have to at least perform some learning to adapt their models or possibly learn new ones.

To simultaneously learn about unknown tasks and unknown teammates, an ad hoc team agent will need to balance the trade-off between exploiting its current knowledge, exploring the dynamics of the task, and exploring the behavior of its teammates. In many ways, this situation is similar to only trying to learn about the teammates. If the ad hoc team agent knows the task and its teammates, the problem can be viewed as a Markov Decision Process (MDP).

On the other hand, if the ad hoc agent knows the task, but not its teammates, the problem can be viewed as a Partially Observable Markov Decision Process (POMDP) where the unobserved variable is behavior of its teammates. Rather than directly observing its teammates' behaviors, the ad hoc agent must reason about these behaviors from the actions it observed. In effect, the ad hoc agent is trying to select the

correct MDP from a set of possible MDPs, where the difference in the MDPs is the behavior of its teammates.

When the ad hoc team agent also does not know the dynamics of the task, the set of possible MDPs increases, expanding along another dimension, but the problem remains fundamentally the same. Therefore, similar methods should allow the ad hoc agent to simultaneously reason about exploring both the task and its teammates, but they must be able to scale to larger problems. The addition of another dimension will greatly increase the number of possible worlds that the ad hoc agent may be inhabiting. Thus, the biggest challenge of this potential contribution is to create algorithms that scale to large POMDPs. It is hopeful that work on scaling the ad hoc team algorithms discussed above to more complex domains will also aid in progress towards reasoning about a larger space of possible worlds. The theoretical analysis in Chap. 6 showed that some versions of the ad hoc team problem can be solved in polynomial time, but if the ad hoc agent needs to simultaneously learn about the domain, the problem becomes much more complex. In general, ε-optimally solving POMDPs takes exponential time in terms of the number of actions and observations. However, approximate methods for planning in POMDPs can achieve good results [11–14].

In addition, there may be ways to limit the complexity of the problem. If the domain is drawn from a limited set, the ad hoc agent may determine which possible world it is in using a Bayesian-based method, similar to the one from Sect. 5.3.2 used to determine with which teammate the ad hoc agent is cooperating. Some research has looked into how to select which MDP an agent is in [15], but combining this with determining the teammates' behavior simultaneously is an interesting problem that is necessary for ad hoc team agents to be ready for deployment into the real world.

8.4.3 Human Interactions

In this book, we consider ad hoc team agents that cooperate with a variety of autonomous agents. However, another type of teammate that agents should be robust to is humans. Ideally, in the future, robots and other agents will be able to quickly adapt to human teammates to accomplish shared tasks. Robots should be able to cooperate with new human teammates as they do with other artificial agents. There is current research into human-agent interactions such as Peled et al.'s work [16]. Peled et al. use machine learning techniques to determine which social factors affect the human players' behaviors and combine these predictions with a decision-theoretic approach to negotiate with humans. However, these approaches commonly only consider dealing with humans, rather than also considering other artificial agents. We would like agents to be able to cooperate with both humans and artificial agents interchangeably, cooperating with both whenever they are present.

The main additional difficulty of cooperating with humans compared to other agents is the humans' relative lack of predictability. Autonomous agents are likely to select the same actions or choose from the same distribution of actions when the

repeatedly visit the same state. Humans do not necessarily follow this assumption; instead, they change their behaviors as the learn, explore, or get bored. In addition, in many situations it is possible for humans to supply a wider range of behaviors than those explored in this book. Each human is unique, and therefore may exhibit a unique behavior.

We expect humans' unpredictability to mainly be a problem for learning models of human teammates, as it is difficult to collect many trials of humans following the same behavior. Instead, it is likely that they will present a spread of possible behaviors during the learning process, making any learned models very noisy. Furthermore, humans are often impatient when dealing with automated systems, which places higher importance of ad hoc agents adapting quickly to their human teammates. In summary, there is no fundamental difference in interacting with humans instead of autonomous agents, but we expect that learning models or policies for cooperation will be more difficult. In addition, we expect that learning quickly will be even more important.

In this book, we propose the approach of learning about past teammates and reusing this information to learn more quickly with new teammates in the from of PLASTIC. We present two instantiations of PLASTIC: PLASTIC–Model that learns models of previous teammates and PLASTIC–Policy which learns policies to cooperate with previous teammates. We expect that this general approach will apply well to dealing with humans despite the wide range of behaviors that humans can exhibit. This expectation is supported by research showing that a small number of behaviors captures the majority of human behaviors in certain tasks [17]. Similarly, research on bandit problems suggests that only a limited number of strategies are viable in social settings [18]. Therefore, learning a small number of teammate models and adapting them to new teammates should allow ad hoc agents to cooperate with a wide variety of humans.

In order to use this approach for cooperating with humans, it is necessary to identify which models are most representative of the spread of possible behaviors that humans might exhibit. In this book, we learn a model for each previous type of teammates and then select from these models when encountering a new teammate. However, human behaviors will have small differences, so the number of possible models that fully represent these past human teammates may quickly grow too large to select from effectively. Therefore, it may be necessary to employ a clustering approach to discover a small number of behaviors that represent all observed behaviors with minimal errors.

One promising approach for solving this problem is that of Mahmud et al. [19]. Mahmud et al. present algorithms to cluster Markov Decision Processes (MDPs), representing these MDPs with a smaller set of MDPs. Their approach relies on looking at the optimal policies that have been learned for each MDP and looking at the cost of reusing these policies on the other MDPs. This analysis results in a cost function that can then be used as an input into a clustering algorithm based on the Metropolis-Hastings algorithm. This approach could be directly applied to PLASTIC–Policy to calculate a smaller subset of policies for cooperating with humans. However, their approach then relies on using a bandit algorithm to update which policy to use in a

new setting, which our results in Sect. 7.3 show is much slower than the Bayesian approach adopted by PLASTIC–Policy. Therefore, it would be advantageous to build a rough model of the teammates in the cluster to allow PLASTIC–Policy to use Bayesian updates to quickly determine which policy is best for cooperating with the current teammates. How to build this combined model remains an open question. In addition, other approaches for clustering policies may prove to be more effective than the one proposed by Mahmud et al. [19]. Alternatively, clustering teammates in the model space may prove to be more tractable, enabling PLASTIC–Model to scale well with humans.

Whatever approaches are eventually adopted, it is also important to consider how to test these approaches. Running large scale human experiments is time consuming. In addition, automatically perceiving human actions at a high level is difficult, although research in activity recognition is making significant progress (for example, see Chen and Grauman's work [20]). Therefore, one place to start is by using human-controlled agents. Rather than directly interacting with humans, ad hoc agents can interact with other agents that are being remotely controlled. This approach greatly reduces the perception problem. In addition, this approach allows large scale tests to be run in simulation, using online services to find human participants, such as Amazon Mechanical Turk.[1]

We suggest using remote-controlled agents over peer designed agents (PDAs) [21] because using PDAs reduces the unpredictability and adaptations of the agents. We expect that methods that are similar to PLASTIC will perform well with humans, but these methods may need to be adapted. Testing this hypothesis remains future work. Future research should then expand these approaches to real human tests by handling the greater uncertainty of the humans' actions.

8.4.4 Learning to Communicate

In Sect. 3.2.1, we describe an ad hoc teamwork scenario that involves communication. In this scenario, the teammates' responses to messages may be unknown, but the meaning of the messages is clear. However, this assumption is not always true; ad hoc agents should also be able to learn about ambiguous messages. They should be able to learn new languages and learn the meanings of unknown messages.

Past work has looked at methods for evolving a common language over time. One early investigation of learning about communication was performed by Levin [22]. Levin demonstrates that a simulated population can converge to a single scheme for coding and decoding messages. Further work by Kirby explored how artificial life approaches could be applied to learning languages and the meaning of messages [23]. In addition, Rawal et al. [24] look at evolving communication in the pursuit domain. The authors look at a version of the pursuit domain where predators can send real valued signals to each other. The results show that without any initial meaning to

[1] http://www.mturk.com.

these signals, the predators are able to evolve a messaging code that improves the performance of the team. Tuci et al. [25] also consider evolving simple communication for a group of robots equipped with different sensors. These papers show that it is possible to evolve meaningful languages without providing an initial language to the agents. Similar methods could allow agents on ad hoc teams to learn a language to communicate with each other.

Another promising line of work looks at how to interpret ambiguous messages [26, 27]. In these works, Grizou et al. considers the case where one party knows the meaning of the messages, and the goal is for the other agent to learn what these messages mean. In addition, they show that agents can simultaneously determine what task they should be considering. This type of reasoning would allow ad hoc agents to learn the languages of their teammates, if these agents already have a language. Importantly, these approaches could also allow agents to quickly determine the meaning of any human feedback when interacting with people.

8.4.5 Improvements in Transfer Learning

To reuse information from past teammates when cooperating with new teammates, ad hoc agents can employ transfer learning algorithms. In this book, we introduce the TwoStageTransfer algorithm for transferring knowledge from many source data sets (past teammates) to build models of the target data set (new teammate). However, we believe that improvements can be made on this approach. TwoStageTransfer is an improvement over existing transfer learning algorithms in that it considers how similar each source data set is to the target data and weights data coming from these sets accordingly. However, in ad hoc teamwork scenarios, these past teammates may be more similar to the current ones in specific parts of the state space. Therefore, it would be helpful to also consider the weightings of different parts of the state space separately, though this does increase the complexity of the problem.

One approach for solving this problem is to adopt an approach similar to TwoStageTransfer. TwoStageTransfer determines the weighting of each source data set using cross validation. Unfortunately, when considering the many partitions of a source data set, calculating a model and then cross validating it may become too computationally intensive. Therefore, a more computationally efficient approach is to treat this problem as a hierarchical Bayesian model, where a source data set has some base similarity distribution to the target data, partitions of this set have similarities drawn from the source data's similarity distribution, and further partitions draw their similarities from the partitions above. This approach allows for fast and simple inference of the similarities of different parts of the state space. However, how to choose these partitions remains an open problem. It is not clear whether this approach will outperform TwoStageTransfer, given that TwoStageTransfer uses a more empirical approach of determining the weighting of each source data set using cross validation, but the approach is promising.

Another approach to applying transfer learning to ad hoc team problems is to transfer policies rather than models. In this book, we directly reuse policies using PLASTIC–Policy, but it is possible to improve on this approach. As above, it is possible in certain areas of the state space, a past teammate is more similar to the current teammate than others. Therefore, it may be possible to combine several different policies to cooperate with a new teammate by using different policies in different parts of the state space. One promising approach for performing this type of transfer is to reuse data of the form $\langle s, a, r, s' \rangle$ from previous domains [28]. Reusing this information allows the agent to learn a policy more quickly on the target domain. Another approach to transferring instances from previous domains is the TIMBREL algorithm introduced by Taylor et al. [29]. TIMBREL transfers saved instances of the transition and reward function that will be used immediately by the agent, ignoring states that are for from the current state. Alternatively, recent research has discovered methods for automatically mapping transfer between tasks [30]. A promising approach is to extend this approach to consider transferring from different parts of the state space rather than considering different mappings. How to perform this transfer is an interesting open problem.

8.4.6 Summary

This section describes five areas of future research that are motivated by this book. One extension of this work is to apply PLASTIC to complex robotic domains. Another area for future research is to enable ad hoc agents to efficiently learn about its environment while cooperating with unknown teammates, which Sect. 4.4.4 identifies as an open problem using the dimensions introduced in Sect. 2.3. Next, expanding ad hoc teamwork to cooperating with noisy humans in addition to artificial agents is an exciting problem. In addition, enabling ad hoc agents to learn about ambiguous messages and learn new languages will allow them to act effectively in many more situations. Finally, more advanced transfer learning methods can speed up learning and improve the behaviors learned for ad hoc team agents cooperating with unknown teammates.

8.5 Conclusion

This book presents the PLASTIC algorithm, which is the first to allow agents to quickly adapt to new teammates by reusing knowledge about previous teammates. PLASTIC fulfills the three desiderata for an ad hoc team agent algorithm: it is robust to a variety of teammates, it is effective on several different domains, and it allows fast adaptation to its teammates. Therefore, PLASTIC is applicable to many complex ad hoc team problems, allowing agents to quickly learn and adapt to their teammates.

We demonstrate the effectiveness of PLASTIC on three domains with a variety of teammates and prior knowledge. Thus, this book represents an important step towards intelligently handling ad hoc team problems in the real world.

References

1. Tambe, Milind. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* (*JAIR*) 7: 81–124.
2. Decker, Keith S., and Victor R. Lesser. 1995. Designing a family of coordination algorithms. In *International conference on multi-agent systems* (*ICMAS*), pp 73–80, June 1995.
3. Watkins, Christopher John Cornish Hellaby. 1989. Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge, UK, May 1989.
4. Ernst, Damien, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. In *Journal of machine learning research* (*JMLR*), pp 503–556, 2005.
5. Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters. 2013. A survey on policy search for robotics. *Foundations and Trends in Robotics* 2(1–2): 1–142.
6. Albrecht, S.V., and S. Ramamoorthy. 2014. On convergence and optimality of best-response learning with policy types in multiagent systems. In *Proceedings of the 30th conference on uncertainty in artificial intelligence* (*UAI*), Quebec City, Canada, July 2014.
7. Kocsis, Levente, and Csaba Szepesvari. 2006. Bandit based Monte-Carlo planning. In *Prooceedings of the seventeenth European conference on machine learning* (*ECML*).
8. Howard, R.A. 1966. Information value theory. *IEEE Transactions on Systems Science and Cybernetics* 2(1): 22–26.
9. Dearden, Richard, Nir Friedman, and David Andre. 1999. Model-based Bayesian exploration. In *Proceedings of the 15th conference on uncertainty in artificial intelligence* (*UAI*), pp 150–15, 1999.
10. Ross, S., B. Chaib-draa, and J. Pineau. 2008. Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *Proceedings of the IEEE international conference on robotics and automation* (*ICRA*), pp 2845–2851, May 2008.
11. Kurniawati, Hanna, David Hsu, and Wee Sun Lee. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of robotics: Science and systems*.
12. Pineau, Joelle, Geoff Gordon, and Sebastian Thrun. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *The 18th international joint conference on artificial intelligence* (*IJCAI*), pp 1025–1030, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
13. Silver, David, and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23* (*NIPS*).
14. Smith, Trey, and Reid Simmons. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th conference on uncertainty in artificial intelligence* (*UAI*), pp 520–527, AUAI Press, Arlington, Virginia, USA.
15. Brunskill, Emma. 2012. Bayes-optimal reinforcement learning for discrete uncertainty domains (extended abstract). In *Proceedings of the eleventh international conference on autonomous agents and multiagent systems* (*AAMAS*).
16. Peled, Noam, Yaakov(Kobi) Gal, and Sarit Kraus. 2014. A study of computational and human strategies in revelation games. *Autonomous Agents and Multi-Agent Systems* (*JAAMAS*) 29: 1–25.
17. Rosenfeld, Avi, Inon Zuckerman, Amos Azaria, and Sarit Kraus. 2012. Combining psychological models with machine learning to better predict people's decisions. *Synthese* 189: 81–93.
18. Mayo-Wilson, Conor, Kevin Zollman, and David Danks. 2012. Wisdom of crowds versus groupthink: Learning in groups and in isolation. *International Journal of Game Theory* 42: 695–723.

19. Mahmud, M.H., Majd Hawasly, Benjamin Rosman, and Subramanian Ramamoorthy. 2013. Clustering Markov decision processes for continual transfer. Technical report, School of Informatics, The University of Edinburgh, United Kingdom.
20. Chen, Chao-Yeh, and K. Grauman. 2012. Efficient activity detection with max-subgraph search. In *Proceedings of the conference on computer vision and pattern recognition* (*CVPR*), pp 1274–1281, June 2012.
21. Lin, Raz, Sarit Kraus, Yinon Oshrat, and Ya'akov(Kobi) Gal. 2010. Facilitating the evaluation of automated negotiators using peer designed agents. In *Proceedings of the twenty-fourth conference on artificial intelligence* (*AAAI*).
22. Levin, Michael. 1995. The evolution of understanding: A genetic algorithm model of the evolution of communication. *Biosystems* 36(3): 167–178.
23. Kirby, Simon. 2002. Natural language from artificial life. *Artificial Life* 8(2): 185–215.
24. Rawal, Aditya, Padmini Rajagopalan, Risto Miikkulainen, and Kay Holekamp. 2012. Evolution of a communication code in cooperative tasks. In *Artificial life* (*13th international conference on the synthesis and simulation of living systems*), East Lansing, Michigan, USA.
25. Tuci, Elio, Christos Ampatzis, Federico Vicentini, and Marco Dorigo. 2006. Operational aspects of the evolved signalling behaviour in a group of cooperating and communicating robots. In *Symbol Grounding and Beyond*, vol. 4211, ed. Paul Vogt, Yuuya Sugita, Elio Tuci, and Chrystopher Nehaniv, 113–127., Lecture notes in computer science Berlin: Springer.
26. Grizou, Jonathan, Manuel Lopes, and Pierre-Yves Oudeyer. 2013. Robot learning simultaneously a task and how to interpret human instructions. In *Joint IEEE international conference on development and learning and on epigenetic robotics* (*ICDL-EpiRob*), Osaka, Japan.
27. Grizou, Jonathan, Iñaki Iturrate, Luis Montesano, Pierre-Yves Oudeyer, and Manuel Lopes. 2014. Interactive learning from unlabeled instructions. In *Proceedings of the 30th conference on uncertainty in artificial intelligence* (*UAI*).
28. Lazaric, Alessandro, Marcello Restelli, and Andrea Bonarini. 2008. Transfer of samples in batch reinforcement learning. In *Proceedings of the twenty-fifth international conference on machine learning* (*ICML*), pp 544–551, ACM, New York, NY, USA.
29. Taylor, Matthew E., Nicholas K. Jong, and Peter Stone. 2008. Transferring instances for model-based reinforcement learning. In *Machine learning and knowledge discovery in databases*, *volume 5212 of lecture notes in artificial intelligence*, pp 488–505, Sept 2008.
30. Ammar, Haitham Bou, Decebal Constantin Mocanu, Matthew E. Taylor, Kurt Driessens, Karl Tuyls, and Gerhard Weiss. 2013. Automatically mapped transfer between reinforcement learning tasks via three-way restricted boltzmann machines. In *Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases* (*ECML PKDD*), Sept 2013.

Appendix A
Hand-Coded Pursuit Teammates

While Sect. 3.2.2.1 briefly introduces the hand-coded teammates used in the pursuit domain, it leaves out many of the details of the implementations. This appendix addresses this gap, giving more information about the hand-coded teammate types.

To more accurately describe these hand-coded predators, some additional notation is helpful. Assume that a predator is at position (x, y) and is trying to move to a destination (x', y') on a world of size (w, h).

$$\begin{aligned} \Delta_x &= (x' - x) \bmod w & \Delta_y &= (y' - y) \bmod h \\ \dim_{\min} &= \operatorname{argmin}(\Delta_x, \Delta_y) & \dim_{\max} &= \operatorname{argmax}(\Delta_x, \Delta_y) \\ & & m_i &= \text{argmin moves } \Delta_i \end{aligned}$$

Thus, m_i is the move that minimizes the difference to the destination for dimension i, and $\overline{m_i}$ is the move in the opposite direction. The stochastic agents use the softmax activation function, which assigns probabilities to a set of values, favoring the higher values. The temperature, τ, controls the amount of this bias, with values closer to 0 resulting in higher probabilities of the maximum value. If $v(i)$ is the value of option i, the probability of option a is

$$p(a) = \frac{\exp(v(a)/\tau)}{\sum_{i=1}^{n} \exp(v(i)/\tau)}$$

To clarify the predators' behaviors, examples of their action selection on the cases shown in Fig. A.1 are discussed, looking at the actions taken by the starred agent. The letters in the figure indicate the destination of the agent after taking one step. Note that none of the hand-coded predators ever choose to stay still, so that action is not labeled.

Greedy Predator (GR)

The greedy predator selects the nearest unoccupied cell neighboring the prey, and tries to move towards it while avoiding immediate obstacles. It follows the succeeding rules in order.

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4

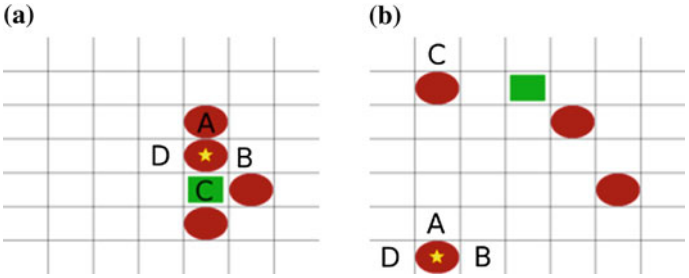

Fig. A.1 World configurations that differentiate the teammates' behaviors. **a** Configuration 1. **b** Configuration 2

- If already neighboring the prey, try to move onto the prey so that if it moves, the predator will follow.
- Choose the nearest unoccupied cell neighboring the prey as the destination.
- Let $d = \dim_{\max}$. If m_d is not blocked, take it.
- Let $d = \dim_{\min}$. If m_d is not blocked, take it.
- Otherwise, move randomly.

For example, using the configurations shown in Fig. A.1 and taking actions as the starred agent, if the starred agent were a Greedy predator, it chooses the move taking it to cell C in configuration 1, and B in configuration 2. On average, a team of all Greedy predators captures a randomly moving prey in 7.74 steps on a 5×5 world.

Teammate-aware Predator (TA)

The teammate-aware predator considers its teammates' distances from the prey when selecting its destination and uses A* path planning (an optimal heuristic search algorithm) [1] to avoid other agents, treating them as static obstacles. In contrast to the greedy predator, a teammate-aware predator that is already neighboring the prey may move towards another neighboring cell to give its spot to a farther away teammate. It is implemented as follows.

- Calculate the distance from each predator to each cell neighboring the prey.
- Order the predators based on worst shortest distance to a cell neighboring the prey.
- In order, the predators are assigned the unchosen destination that is closest to them (without communication), breaking ties by a mutually known ordering of the predators.
- If the predator is already at the destination, try to move onto the prey so that if it moves, the predator will follow.
- Otherwise, use A* path planning to select a path, treating other agents as static obstacles.

For the configurations shown in Fig. A.1, a Teammate-aware predator in the position of the starred predator chooses the move taking it to cell D in configuration 1, and C in configuration 2 (note that since the world is a torus, this is a single move). A team of Teammate-aware predators captures the prey in 7.41 steps on a 5×5 world.

Greedy Probabilistic Predator (GP)

The greedy probabilistic predator moves towards the nearest cell neighboring the prey, but does not always take a direct path there. The predator favors minimizing dim_{max} and prefers m_{dim} over $\overline{m_{\text{dim}}}$.

- If already neighboring the prey, try to move onto the prey so that if it moves, the predator will follow.
- Choose the nearest unoccupied cell neighboring the prey as the destination.
- Given a destination, choose a dimension, d, to minimize using the softmax function with temperature 0.5 using the distance as v.
- Choose either m_d or $\overline{m_d}$ using the softmax function with temperature -0.5, using the distance after the move as v, but penalizing moves that are currently blocked.

On configuration 1 from Fig. A.1, the predator is deterministic, choosing the action taking it to position C. On configuration 2, it selects a distribution of actions, specifically the moves taking it to cells A, B, C, and D with probabilities 0.000, 0.879, 0.119, and 0.002. On a 5×5 world, a team of Greedy Probabilistic predators captures the prey in 12.88 steps.

Probabilistic Destinations Predator (PD)

The probabilistic destinations predator attempts to tighten a circle around the prey. It favors destinations that are both nearer to the prey and to itself, but may choose farther destinations to prevent getting stuck on other predators and dealing with a moving prey.

- If already neighboring the prey, try to move onto the prey so that if it moves, the predator will follow.
- Select a desired distance from the prey using the softmax function with temperature -1 using the distance as v.
- Select a destination at the chosen distance using the softmax function with temperature -1 weighted by the distance of the destination to the predator's current position.
- Let $d = \text{dim}_{\text{max}}$, and select m_d.
- If the destination or the next position is occupied, repeat.

For the configurations in Fig. A.1, a Probabilistic Destinations predator would select the move ending in C in configuration 1. On configuration 2, it would select actions taking it to cells A, B, C, and D with probabilities 0.007, 0.596, 0.388, and 0.009. A team of predators following the Probabilistic Destinations behavior capture in 9.19 steps on a 5×5 world.

Reference

1. Hart, P.E., N.J. Nilsson, and B. Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2): 100–107.

Index

S. Barrett, *Making Friends on the Fly: Advances in Ad Hoc Teamwork*,
Studies in Computational Intelligence 603,
DOI 10.1007/978-3-319-18069-4