UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Ad Hoc Teamwork in Realistic Environments

João Manuel Godinho Ribeiro

Supervisor: Doctor Francisco António Chaves Saraiva de Melo
Co-Supervisor: Doctor José Alberto Rodrigues Pereira Sardinha

Thesis approved in public session to obtain the PhD Degree in
Computer Science and Engineering

Jury final classification: Pass with Distinction

2025

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Ad Hoc Teamwork in Realistic Environments

João Manuel Godinho Ribeiro

Supervisor: Doctor Francisco António Chaves Saraiva de Melo
Co-Supervisor: Doctor José Alberto Rodrigues Pereira Sardinha

Thesis approved in public session to obtain the PhD Degree in
Computer Science and Engineering

Jury final classification: Pass with Distinction

Jury

Chairperson: Doctor José Carlos Alves Pereira Monteiro, Instituto Superior Técnico, Universidade de Lisboa, Portugal
Members of the Committee:

Doctor Matthijs Theodor Jan Spaan, Department of Intelligent Systems, Delft University of Technology, The Netherlands
Doctor Rui Filipe Fernandes Prada, Instituto Superior Técnico, Universidade de Lisboa
Doctor Francisco António Chaves Saraiva de Melo, Instituto Superior Técnico, Universidade de Lisboa
Doctor Reuth Mirsky, Computer Science Department, School of Engineering, Tufts University, USA

Funding Institution

FCT: Fundação para a Ciência e a Tecnologia

2025

Understanding the world is not an excuse to accept it.

Abstract

This thesis studies the problem of *ad hoc* teamwork, where autonomous agents are deployed in unknown scenarios in which they join a team, without any pre-coordination or pre-communication. Specifically, it explores the problem of *ad hoc* teamwork in realistic environment, where states may be partially observable, teammates' actions unobservable, and teams composed of humans and robots.

This thesis asks the main research question:

> *How can an autonomous agent assist unknown teammates in performing tasks under realistic conditions, without the ability to pre-coordinate or pre-communicate with the team?*

From this question, this thesis identifies and studies four sub-problems:

- How can an agent assist teammates in performing a task it doesn't know, without having to learn from scratch how to collaborate with them and how to perform the task itself?
- How can an agent assist its teammates in performing a task it doesn't know, without being able to observe their actions or having a full perception of the state of the environment?
- How can an agent assist its teammates performing a task it doesn't know in arbitrarily large and partially observable environments?
- How can an agent assist robots and/or human teammates in real environments?

The key contributions of this thesis, resulting from the study of the aforementioned problems, are respectively,

- TEAMSTER, a model-based reinforcement learning algorithm which allows continuous adaptation to new teams and tasks in the context of ad hoc teamwork.
- The BOPA and ATPO algorithms, which use Bayesian inference to identify teammates, do not require explicit knowledge of their actions or the complete state of the environment.
- The ATLPO algorithm, which generalizes prior ideas to arbitrarily large partially observable environments, using deep learning techniques.
- The HOTSPOT architecture, which allows the deployment of ad hoc agents in robots operating in real environments with hybrid teams composed of robots and humans.

Keywords: Ad Hoc Teamwork; Multi-Agent Systems; Reinforcement Learning; Partially Observability; Deep Learning

Resumo

Esta tese estuda o problema de trabalho em equipas ad hoc, onde agentes autónomos são colocados em cenários desconhecidos nos quais deverão formar uma equipa sem qualquer pré-coordenação ou pré-comunicação. Especificamente, explora o problema de trabalho em equipas ad hoc em ambientes realistas, onde os ambientes podem ser parcialmente observáveis, as ações dos colegas não-observáveis, e as equipas compostas por humanos e robôs.

Coloca como pergunta de investigação central:

> *Como pode um agente autónomo ajudar colegas de equipa desconhecidos a executar tarefas em condições realistas, sem a capacidade de se pré-coordenar ou pré-comunicar com a equipa?*

A partir desta questão, coloca quatro sub-problemas:

- Como pode um agente auxiliar colegas de equipa a desempenhar uma tarefa que desconhece, sem ter de aprender de raiz a colaborar os mesmos e a desempenhar a tarefa em si?
- Como pode um agente auxiliar colegas de equipa a desempenhar uma tarefa que desconhece, não conseguindo observar as ações dos mesmos nem tendo perceção completa do estado do ambiente?
- Como pode um agente auxiliar colegas de equipa a desempenhar uma tarefa que desconhece em ambientes arbitrariamente grandes?
- Como pode um agente auxiliar colegas de equipa, robôs e/ou humanos, em ambientes reais?

Do estudo dos problemas acima, são as contribuições desta tese:

- O algoritmo de aprendizagem por reforço "model-based" TEAMSTER, que permite a adaptação continua a novas equipas e novas tarefas.
- Os algoritmos BOPA e ATPO, que utilizam inferência Bayesiana para identificar os colegas de equipa, não requerendo conhecimento explícito das suas ações ou do estado completo do ambiente.
- O algoritmo ATLPO, que generaliza as ideias desenvolvidas anteriormente, para ambientes parcialmente observáveis arbitrariamente grandes, utilizando técnicas de aprendizagem profunda.
- A arquitetura HOTSPOT, que permite a implementação de agentes ad hoc em robôs que operam em ambientes reais com equipas híbridas compostas por outros robôs e humanos.

Palavras-Chave: Trabalho de Equipa Ad Hoc; Sistemas Multi-Agente; Aprendizagem Por Reforço; Observabilidade Parcial; Aprendizagem Profunda

Acknowledgements

Earning the title of Doctor of Philosophy is neither easy nor straightforward. Learning how to properly conduct scientific research comes with a few hard-earned lessons. Getting well acquainted with persistence, as in many other noble pursuits, is often the first one. Then, you learn an unexpected yet perhaps more powerful lesson — if left unchecked, persistence alone will often work against you. Not without reason are infamous the challenging conditions PhD students find themselves in during their final years. Wrapping up on research projects, writing and submitting papers as well as incorporating them into a thesis, actively applying to vacancies to secure a future role — none of which would be possible without pragmatism. This equilibrium between persistence and pragmatism, when combined with a deep passion for the field, is sure to take you far. But to truly succeed in pushing forward the boundaries of human knowledge while making it the most pleasant and memorable experience possible, nothing would be possible without surrounding yourself with the right people. Just so happens I was fortunate to have them by my side.

My supervisors, Francisco and Alberto, who first saw in me the potential to grow into a computer scientist and researcher. No words can describe how grateful I am for all the friendship, kindness and patience. Thank you for all the support, your availability and for our nearly religious weekly meetings, without which we would not have come this far. Thank you for all the lessons when dealing with hardship and rejection, and for your prescient suggestions when everything seemed to come to a halt. You truly exceeded my expectations and standards of what great mentors are, and your advice is still very much present as I start this next stage of my life, starting to mentor my own students. I am certain that if our bond was able to grow so much in the face of a quarantine, a research exchange and a relocation to Rio de Janeiro, then nothing will be able to break it. I greatly look forward to continue working together, breaking new boundaries for collaboration.

My colleagues and friends, not only from GAIPS but also from the Algorithmics group. Thank you for all the discussions and growing opportunities. Thank you for all the support, all the lunches and all the beers. A special thanks to Max, Leo, Laurens, as well as Henrique, Ana, Pedro and Jacopo, for all the jam sessions and for allowing music to become one of the fondest memories of this PhD.

My parents, Isabel and Manuel João, to whom I owe everything. I thank you for all the kindness, love and support. Thank you for all the stability, for providing a safe haven without which pursuing a PhD would have never been possible in the first place. Thank you for your values, which I will always carry, and thank you for teaching me that the standard paths often followed by most people are not always the right ones. Thank you for hearing about my hardships faced throughout this academic journey and for the golden council. Most importantly, thank you for being the best role models I could have hoped for.

My beautiful girlfriend Margarida, always by my side and who inspires me to become the best possible version of myself. This victory is also your victory. Not a single day goes by without me wondering where I would be right now if it were not for you. Thank you for taking care of me when I needed recovery and for all the fun moments when I needed to get away from it all. You are the best partner I could hope for. You have no idea how much it meant to me all for you to fly all the way across the globe to watch me present some niche subject of AI to an even more niche audience. I am one-hundred-percent sure you would be able to give a talk on Ad Hoc Teamwork by now. Your genuine interest and respect for Artificial Intelligence, one of my greatest passions, which just so happens to also be my full-time job, are both

blessings that bring me more joy than you could possibly imagine. Thank you for all your love, which I will forever carry in my heart. Here's to the next stage of our lives.

My closest friends, who remained despite the distance. To Henrique, Inês and Rita. To Rafael, Fernando, João and Miguel and Ana, as well as André. Thank you for being there and thank you for listening. A special thank you to Ali, Lina, Jelle, Patrícia and Sofia, for showing me that moving to a new country does not need to be daunting. Thank you for all the kindness, support and good times. I am blessed to have you all in my life.

Finally, my dear family, for always being there for me. To my sister Joana, my aunt Lena and uncle José, my cousins André, Cristina, Pedro and Emanuel. To Fátima, Edgar, Francisca and Joaquim. Thank you for all the love and kindness, and for more than once showing me that a PhD is worth fighting for.

Thank you all for everything.

Contents

Abstract vii

Resumo ix

Acknowledgements xi

1 Introduction 1
1.1 Research Problem . . . 2
1.1.1 How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero? . . . 3
1.1.2 How can an autonomous agent assist unknown teammates in performing unknown tasks without being able to observe their actions, the full state of the environment or both? . . . 3
1.1.3 How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains? . . . 4
1.1.4 How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios? . . . 4
1.2 Contributions . . . 4
1.3 Outline . . . 5

2 Background 7
2.1 Markov Decision Problems . . . 7
2.1.1 Return, Policies, Value Functions and Action-Value Functions . 8
2.1.2 Solving an MDP . . . 8
2.1.3 Planning in MDPs . . . 9
Selection . . . 10
Expansion . . . 10
Simulation . . . 10
Backup . . . 10
2.1.4 Multi-Agent Markov Decision Problems . . . 11
Reducing MMDPs to MDPs . . . 11
2.1.5 Partially Observable Markov Decision Problems . . . 11
Solving a POMDP . . . 12
2.2 Reinforcement learning . . . 13
2.2.1 Model-Based Reinforcement Learning . . . 13
2.2.2 Model-Free Reinforcement Learning . . . 14
2.2.3 Non-Tabular Methods, Function Approximation Techniques and Deep Reinforcement Learning . . . 14

3 Related Work **17**
3.1 *Ad Hoc* Teamwork . . . 17
3.1.1 Problem Formulation . . . 17
3.1.2 Literature Review . . . 18
3.1.3 Related Areas of Research . . . 22
Multi-Agent Reinforcement Learning . . . 22
Zero-Shot Coordination . . . 23
Few-Shot Teamwork . . . 23
Dealing with Humans as Teammates . . . 23
Dealing with Partial Observability & Interactive POMDPs . . . 24
3.2 Discussion & Framing of this Thesis . . . 25

4 Model-Based Reinforcement Learning for Ad Hoc Teamwork **29**
4.0.1 Teammate identification and team tracking . . . 31
Team tracking at runtime . . . 32
4.0.2 Task identification with the learned environment model . . . 33
The environment model . . . 33
4.0.3 Planning and execution . . . 35
4.1 Evaluation . . . 35
4.1.1 Environments and Available Teams . . . 35
4.1.2 Baselines . . . 37
4.1.3 Models . . . 40
4.1.4 Experimental methodology . . . 41
4.1.5 Additional Experiments . . . 43
4.1.6 Results . . . 44
Standard Ad Hoc Evaluation . . . 44
Larger World Adaptation Experiment . . . 49
Performance Convergence Experiment . . . 49
4.2 Conclusion . . . 50

5 Bayesian Online Prediction for Ad Hoc Teamwork **53**
5.1 Bayesian Online Prediction for Ad Hoc Teamwork (BOPA) . . . 53
5.1.1 Preliminaries . . . 54
5.1.2 Algorithm . . . 54
5.1.3 Evaluation . . . 55
Evaluation procedure . . . 56
Metrics . . . 56
Results . . . 56
5.2 Ad Hoc Teamwork under Partial Observability (ATPO) . . . 58
5.2.1 Preliminaries . . . 58
5.2.2 Algorithm . . . 58
5.2.3 Loss bounds for ATPO . . . 60
5.2.4 Evaluation . . . 62
5.2.5 Domain Descriptions . . . 62
Gridworld . . . 63
Pursuit . . . 64
Abandoned Power Plant . . . 65
NTU, ISR, MIT, PENTAGON and CIT . . . 66
Overcooked . . . 67
5.2.6 Hyperparameters for the Perseus Algorithm . . . 68
5.2.7 Results . . . 69

5.2.8 Increasing ATPO's Library . . . 71
5.3 Conclusion . . . 71

6 Ad Hoc Teamwork in Large Partially Observable Domains 73
6.1 Ad hoc Teamwork in Large Partially Observability domains (ATLPO) . 73
6.1.1 Preliminaries . . . 74
6.1.2 Algorithm . . . 74
6.1.3 Choice of Policy Library . . . 75
6.1.4 ATLPO-MF . . . 75
6.1.5 ATLPO-MB . . . 76
6.2 Evaluation . . . 76
6.2.1 Domains . . . 77
Level-Based Foraging . . . 77
Predator-Prey . . . 78
6.2.2 Tasks & Teams . . . 79
Tasks . . . 80
Teams . . . 80
6.2.3 Policy Implementations . . . 80
Encoder Network q . . . 81
Decoder Network d . . . 81
Transition and Reward Networks $\hat{P}$, $\hat{R}$ . . . 82
Actor and Critic Networks $\hat{\pi}$, $\hat{V}$ . . . 83
Bayesian Classifier $\hat{p}$ . . . 83
6.2.4 Baselines . . . 83
6.2.5 Evaluation Procedure . . . 84
Pre-Training Procedure . . . 84
Ad Hoc Evaluation Procedure . . . 84
6.3 Results . . . 84
Task Identification . . . 85
Team Identification . . . 85
Task & Team Identification . . . 88
Task Identification . . . 89
Team Identification . . . 90
Task & Team Identification . . . 90
6.4 Conclusion . . . 91

7 An Ad Hoc Teamwork Platform for Mixed Human-Robot Teams 93
7.1 The HOTSPOT Framework . . . 95
7.2 Conclusion . . . 111

8 Conclusion 113
8.1 Future Work . . . 114
8.1.1 Continual Multi-Task Learning For Ad Hoc Teamwork . . . 115
8.1.2 Bayesian Meta-Learning For Ad Hoc Teamwork . . . 115
8.1.3 Offline Reinforcement Learning for Ad Hoc Teamwork . . . 115

List of Figures

1.1 A depiction of the *ad hoc* teamwork fallen cyclist example introduced by Stone et al. [2010b]. *"Imagine that you are in a foreign country where you do not speak the language, walking alone through a park. You see somebody fall off of his bicycle and injure himself badly; there are a few other people in the area, and all of you rush to help the victim. There are several things that need to be done. Somebody should call an ambulance, someone should check that the victim is still breathing, and someone should try to find a nearby doctor or policeman. However, none of you know one another, and thus you do not know who has a mobile phone, who is trained in first aid, who can run fast, and so forth. Furthermore, not all of you speak the same language. Nonetheless, it is essential that you quickly coordinate towards your common goal of maximising the victim's chances of timely treatment and survival. This scenario is an example of what we call an ad hoc team setting." [Stone et al., 2010b].* Image adapted from `https://www.theglobeandmail.com` 2

2.1 Flowchart for a Markov Decision Problem. In a time step t, an agent executes an action A_t on the environment and is rewarded with R_{t+1} while the environment transitions into a next state X_{t+1}. 7
2.2 Illustration of the four steps of an MCTS iteration. Figure from Sutton and Barto [2018]. 9
2.3 Flowchart for an Multi-Agent Markov Decision Problem. In a time step t, each agent executes an action A_t^n on the environment and the team is rewarded with R_{t+1} while the environment transitions into a next state X_{t+1}. 11
2.4 Flowchart for a Partially Observable Markov Decision Problem. In a time step t, the agent executes an action A_t^n on the environment, is rewarded with R_{t+1} and obtains a partial observation Z_{t+1} of the next state of the environment X_{t+1}. 12

3.1 Interaction between an ad hoc agent α and teammates $-\alpha$ described by an MMDP M. 18
3.2 Ad hoc teamwork as a problem of planning. The ad hoc agent is assumed to know the task's dynamics and goal, represented by a model M, and the policy of the team $\pi^{-\alpha}$. This information, alongside the state of the environment X_t, can be fed to a planner in order to compute the best response A_t^α for the ad hoc agent. 18
3.3 Overview architecture for the PLASTIC-Model algorithm [Barrett and Stone, 2015]. The ad hoc agent α is given a library of possible team policies $\Pi^{-\alpha}$ which it uses to perform Bayesian updates (Eq.3.1) to a prior distribution p. Using the most likely team policy alongside an assumed known model of the task, M, the ad hoc agent uses a MCTS planner in order to compute a best-response A_t^α. 19

3.4 Ad Hoc Teamwork broken down into three sub-problems [Melo and Sardinha, 2016]. The ad hoc agent has the explicit goal of (i) identifying the task being performed by the team (dynamics and goal modelled by M), (ii) identifying the policy of the team π^{α} and (iii) planning using the task and team identification in order to compute a best response A_t^{α}. 20
3.5 Overview architecture for the PLASTIC-Policy algorithm [Barrett et al., 2017]. The ad hoc agent α is given (i) a library of possible team policies $\Pi^{-\alpha}$, which it uses to perform Bayesian updates (Eq.3.1) to a prior distribution p, and (ii) a library of pre-trained policies for the ad hoc agent, Π^{α}. Each policy $\pi_k^{\alpha} \in \Pi^{\alpha}$ was previously trained via reinforcement learning for team $\pi_k^{-\alpha}$. Using the most likely team policy $\pi_k^{-\alpha}$, the ad hoc agent samples an action from the associated pre-trained policy π_k^{α} in order to obtain a best-response A_t^{α}. Whenever the ad hoc agent is interacting with a team, it also trains two new policies, π_{K+1}^{α} and $\pi_{K+1}^{-\alpha}$. π_{K+1}^{α}, trained via reinforcement learning using rewards R_{t+1}, is then stored in Π^{α} by the end of the interaction, while $\pi_{K+1}^{-\alpha}$, trained via supervised learning using the actions of the team $A_t^{-\alpha}$, is then stored in $\Pi^{-\alpha}$. 21
3.6 Framing for this thesis. In order to answer the research question of *how can an autonomous agent assist unknown teammates in performing unknown tasks under more realistic conditions, without being able to pre-coordinate or pre-communication with the existing team?*, we break down our thesis in four different sub-problems. We start in chapter 4 by introducing TEAMSTER, a model-based approach for ad hoc teamwork capable of being more sample efficient than PLASTIC-Policy Barrett et al. [2017]. We then drop the assumptions that the ad hoc agent has access to the actions of the teammates, introducing BOPA, an approach capable of performing ad hoc teamwork relying only on the state of the environment, and ATPO, an approach capable of performing ad hoc teamwork relying only on partial observations of the environment (both in chapter 5). We then expand ATPO for arbitrarily large, partially observable domains, introducing ATLPO, an approach for ad hoc teamwork which employs state-of-the-art Deep Learning techniques to identify the most likely team and task, as well as compute a best-response. We present two variants of ATLPO, ATLPO-MF, which sets up its policy library using state-of-the-art techniques in Model-Free reinforcement learning, and ATLPO-MB, which sets up its policy library using state-of-the-art techniques in Model-Based reinforcement learning. Finally, in chapter 7, we introduce and evaluate a framework for the deployment of ad hoc approaches to real-world scenarios where mixed teams of humans and other robots operate. 27

4.1 TEAMSTER ad hoc agent's architecture. 30
4.2 The team model. A parametrised model which can be optimised to accurately predict the teammates' policies 32
4.3 The environment model $\mathcal{E}$. We assume that $\mathcal{E}$ contains a parametrised representation for the MMDP transition probabilities (denoted as $\hat{\mathbf{P}}$) and reward (denoted as $\hat{r}$). These representations are then optimised to accurately predict the transitions observed from the environment. . 34

4.4 Pursuit domain. (a) The world is a 5×5 toroidal world—i.e., upon exiting one side of the environment, an agent immediately re-enters in the opposite side. Four predators (represented as four ghosts) must capture a prey (here represented as Pacman). (b) Example of a capture configuration. Since the world is toroidal, the orange ghost position is "equivalent" to standing on the left side of Pacman. 36
4.5 CMU-Gridworld domains [Melo and Veloso, 2009, Hu et al., 2015] (NTU, ISR and Pentagon). Agents must navigate in a grid world in order to each reach an exit while taking into account collisions. 37
4.6 Different neural network architectures used in the Pursuit domain. Input layers are represented as $M \times 1$ feature vectors. Subsequent layers are labelled with the type of activation and (#inputs $\times$ #outputs). 39
4.7 Pre-training plots for the four learning ad hoc agents, on each team on the Pursuit domain. Shaded areas correspond to the confidence interval of 95%. 44
4.8 Pre-training plots for the four learning ad hoc agents, on each team on the NTU domain. Shaded areas correspond to the confidence interval of 95%. 45
4.9 Pre-training plots for the four learning ad hoc agents, on each team on the ISR domain. Shaded areas correspond to the confidence interval of 95%. 46
4.10 Pre-training plots for the four learning ad hoc agents, on each team on the Pentagon domain. Shaded areas correspond to the confidence interval of 95%. 47
4.11 Average accumulated reward for the different approaches on each team on a 10×10 world, in the Pursuit domain. Shaded areas correspond to the confidence interval of 95%. 50
4.12 Asymptotic performance of each agent in an additional 'Mixed' team of the Pursuit domain. Shaded areas correspond to the confidence interval of 95%. 51

5.1 Interaction between an ad hoc agent α and teammates $-\alpha$, described by an MMDP M. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1} 54
5.2 Interaction between an ad hoc agent α and teammates $-\alpha$, described by an MDP M^α that reduces M according to $\pi^{-\alpha}$ using Equation 2.2. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}. 55
5.3 Probability of correct task averaged across the whole episode, $1/T \sum_t p_t(r)$ (left) and probability of correct task, at the last step of the episode, $p_T(r)$ (right). The error bars correspond to the variability in the agent's estimate. 56
5.4 Interaction between an ad hoc agent α and teammates $-\alpha$, described by a POMDP $\hat{M}^\alpha$ with an underlying MDP M^α which reduces an MMDP M according to policy $\pi^{-\alpha}$ using Equation 2.2. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$, the reward signal R_{t+1} and the state of the environment X_t. 59
5.5 The gridworld domain. Two agents must each navigate to two goal cells. 64
5.6 The pursuit domain. Two agents must capture a moving prey. 65

5.7 The ntu, isr, mit, pentagon and cit domains. Two agents must navigate in close quarters in order to each reach an exit (while taking into account collisions). 67
5.8 The overcooked domain. Two agents, a cook and an assistant, are required to deliver soups as fast as possible. 68
5.9 Varying the total number of possible POMDPs from the gridworld domain in ATPO's library. Results correspond to the relative performance, normalised between the best possible performance (100%, obtained by the Value Iteration agent) and worst possible performance (0%, obtained by the Random Agent). 71

6.1 Interaction between an ad hoc agent α and teammates $-\alpha$, described by a POMDP $\hat{M}^{\alpha}$ with an underlying MDP M^{α} which reduces an MMDP M according to policy $\pi^{-\alpha}$ using Equation 2.2. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$, the reward signal R_{t+1} and the state of the environment X_t. 74
6.2 Actor-Critic architecture used for each policy in ATLPO-MF's library. An Encoder q, Actor $\hat{\pi}$ and Critic $\hat{V}$ are trained end-to-end using using proximal policy optimisation [Schulman et al., 2017]. 75
6.3 Architecture used for each policy in ATLPO-MB's library. An Encoder q, Transition Predictor $\hat{P}$, Reward Predictor $\hat{R}$ and Decoder d are trained end-to-end, computing latent state representations used to train the Actor $\hat{\pi}$ and Critic $\hat{V}$ independently as a standard fully observable reinforcement learning problem. 76
6.4 Observations for the Level-Based Foraging domain for two example states. A total of $C = (N-1) + M + 2$ channels, each a matrix of shape $(f \times f)$ representing a specific piece of information, with $f = 5$ throughout our experiments. A total of $N-1$ channels indicate the relative positions of each teammate, $N = 2$ above, A channels indicate the relative positions of each apple, $A = 2$ above, one channel indicates the levels of agents and apples within the field-of-view and another channel indicates the boundaries of the world. Each apple and teammate are given their own individual channel, since strategies in this domain sometimes use their indexes for resolving stalemates and without an individual channel there would be no way to distinguish different apples and teammates. 79
6.5 Implementation each policy's architecture from ATLPO-MF's library. . 80
6.6 Implementation each policy's architecture from ATLPO-MB's library. . 81
6.7 Encoder Network q used to extract latent states $\hat{X}_t$ from observations Z_t. Observations $Z_t \in \mathbb{R}^{(C \times 5 \times 5)}$ are passed through two 2D Convolutional layers, one LSTM layer and an MLP, computing a latent state $\hat{X}_t \in \mathbb{R}^{128}$. 82
6.8 Decoder network d used to reconstruct observations Z_t from latent states $\hat{X}_T$. Latent states $\hat{X}_t$ are passed through a single Linear layer, reducing it to 64 features and afterwards through two 2D Transposed Convolutional Layers, reconstructing the observation $\hat{Z}_t \in \mathbb{R}^{(C \times 5 \times 5)}$. . 82

6.9 Recurrent Bayesian classifier used to identify the most likely task and team combination from observation-action pairs. Observations $Z_t \in \mathbb{R}^{(C\times5\times5)}$ are first passed through two 2D Convolutional Layers, obtaining a flattened latent vector $Z'_t \in \mathbb{R}^{64}$, concatenated with the one-hot representation $\mathbf{e}_{A^{\alpha}_t}$ of the action executed by the ad hoc agent α. The concatenated vector $(Z'_t \oplus \mathbf{e}_{A^{\alpha}_t}) \in \mathbb{R}^{64+\mathbf{A}}$ is then passed through an LSTM and MLP layer, outputting the logits used for classification which can be used to compute a distribution $p(m_k)$ using an additional Softmax. 83
6.10 Average steps to solve an episode on the task identification set. The six agents are evaluated for sixteen trials on each of the four tasks on each of the two domains 86
6.11 Average steps to solve an episode on the team identification set. The six agents are evaluated for sixteen trials with each of the three team on each of the two domains 87
6.12 Average steps to solve an episode on the task and team identification set. The six agents are evaluated for sixteen trials on each of the seven task/team combinations on each of the two domains 89
6.13 Belief progressions for ATLPO-MF and ATLPO-MB in both domains. In each timestep, the Recurrent Bayesian Classifier $\hat{p}$ outputs a probability distribution over all tasks, with each entry representing the belief over its respective task. The belief for the correct task is shown in green, while the average sum of the incorrect ones is shown in red. . 90
6.14 Belief progressions for ATLPO-MF and ATLPO-MB in both domains. In each timestep, the Recurrent Bayesian Classifier $\hat{p}$ outputs a probability distribution over all teams, with each entry representing the belief over its respective team. The belief for the correct team is shown in green, while the average sum of the incorrect ones is shown in red. . 91
6.15 Belief progressions for ATLPO-MF and ATLPO-MB in both domains. In each timestep, the Recurrent Bayesian Classifier $\hat{p}$ outputs a probability distribution over all team-task combinations, with each entry representing the belief over its respective team-task combination. The belief for the correct team-task combination is shown in green, while the average sum of the incorrect ones is shown in red. 92

7.1 **HOTSPOT Diagram.** Diagram depicting the interaction between the different modules in HOTSPOT. 95
7.2 **HOTSPOT Decision Module Diagram.** Diagram depicting the decision module of HOTSPOT. 96
7.3 **HOTSPOT Communication Module Diagram.** The communication module, comprising both a *sensing* pipeline that converts speech to state information, and an *actuation* pipeline, converting communicating actions to text to be then spoken by the robot. 99
7.4 **Lab Room Setup.** Lab room used to simulate the Toxic Waste domain (left) and respective layout (right). Each area is represented as a node in a topological map. 101
7.5 **Toxic Waste Materials.** The three balls representing the toxic waste materials. 101
7.6 **Task Configurations.** The two task configurations (i.e., locations of the three balls representing the toxic waste materials). 101
7.7 **Confusion Matrix.** Communication module's pipeline confusion matrix. 103

7.8 **Average number of steps required to solve and identify the target task was analyzed across different scenarios, considering the use of the communication module.** All simulated trials started from the same initial states as those in the live trials. Error bars correspond to a confidence interval with the confidence of 95% ($\alpha = 0.05\%$), calculated over all trials, encompassing 9 real-world trials and 32 trials in the simulated environment). 107
7.9 **Beliefs Entropy.** Entropy of the beliefs at each time step, averaged over all nine trials. 107
7.10 **NLP Modules Accuracies.** The accuracies for the three NLP modules - Speech Recognition (58.62%), NER (77.77%), and Node Identification (74.07%). 108
7.11 **Disabled Communication Experiment.** Average steps to complete a task for different approaches in the disabled communication experiment. Error bars correspond to a confidence interval with confidence of 95% ($\alpha = 0.05$), calculated over a total of 32 independent trials. . . . 110
7.12 **Predator-Prey Domain Results.** Average number of steps to complete a task in the Predator-Prey domain. Error bars correspond to a confidence interval confidence of 95% ($\alpha = 0.05$), calculated over all trials. 111

List of Tables

4.1 Hyperparameters used for the UCT planner (shared by TEAMSTER and PLASTIC-Model). . . . 42
4.2 Architecture used for TEAMSTER's environment model and the team models (same for TEAMSTER, PLASTIC-Model and PLASTIC-Policy), for all environments. . . . 42
4.3 Architecture used for PLASTIC-Policy's DQNs. . . . 42
4.4 Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the Pursuit domain. . . . 46
4.5 Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the NTU domain. . . . 47
4.6 Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the ISR domain. . . . 48
4.7 Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the Pentagon domain. . . . 48

5.1 Average number of steps required for task completion. . . . 57
5.2 Multi-agent domains used for the ad hoc evaluation. π_1 represents whether the domain allows for varying the teammate's policy, r represents whether the domain allows for varying the target task/teammate combination, $|\mathcal{X}|, |\mathcal{A}|$ and $|\mathcal{Z}|$ represent the total number of states, actions and observations per POMDP from each domain (respectively), ϵ represents the amount of noise used in computing the transition and observation probabilities for each POMDP, h represents the number of steps in the trajectory horizon during which each POMDP is ran as an environment and, finally, $|\mathcal{M}|$ represents the total number of different setup POMDPs for each domain (which also corresponds to the size of our approach's model library). . . . 63
5.3 Times to setup and solve POMDPs from each domain. Setup times represent the total time it takes to create the data structures of the POMDP and solve times represent the total time it takes the Perseus algorithm to obtain a policy. . . . 63
5.4 Hyperparameters used for the Perseus [Spaan and Vlassis, 2005] algorithm. The discount factor was also used for the Value Iteration algorithm. . . . 68
5.5 Average accumulated reward for the six agents in the eleven domains over 32 trials (adding to a total of 1152 trials). BOPA requires a common state-space, so it cannot be run in the abandoned power plant scenario. 69

5.6 ATPO's results from Table 5.5 normalised between the best possible performance (100%, obtained by the Value Iteration agent) and worst possible performance (0%, obtained by the Random Agent). 70

6.1 Average steps it took each agent to solve the episodes when placed in each team of each domain, in the Team Identification problem set. . . . 88

To my parents, Isabel and Manuel João,
and to my girlfriend, Margarida.

Chapter 1

Introduction

The development of *autonomous agents* — algorithms capable of autonomous decision-making without prompting for human control — has always been one of the core goals of Artificial Intelligence (AI) [Russell, 2010]. Over the past two decades, new approaches enabled AI agents to not only learn a multitude of complex tasks, but also to outperform humans in some of them [Mnih et al., 2015]. However, when presented with a task on-the-fly, these approaches lack one of the core qualities of human intelligence — the ability to adapt only using prior experiences without having to learn everything from zero. Furthermore, in tasks where other teammates might be already operating, pre-coordinating or pre-communicating with them might not be possible. This very research question — *how can autonomous agents assist unknown teammates in performing unknown tasks on-the-fly, without any pre-coordination or pre-communication* — sparked an entirely new area of research — the problem of *ad hoc teamwork*. Introduced by Stone et al. [2010b], the authors motivate the problem of *ad hoc* teamwork with an analogy of being in a foreign country where we do not speak the local language and finding a group of people assisting a fallen cyclist, with multiple tasks needed to be performed. In such scenario, one would also have to identify who is already doing what, without being able to communicate with them (figure 1.1).

The key characteristic of *ad hoc* teamwork, and what distinguishes it from traditional problems such as multi-agent reinforcement learning, is that agents are not evaluated according to how well they perform tasks after explicitly pre-training for them. They are instead evaluated according to how well they perform tasks on-the-fly, after being placed in a multi-agent environment with pre-established teams, implying they have to make use of their prior experiences, e.g. by identifying tasks or teammates they explicitly pre-trained for in the past, and use this knowledge on-the-fly.

With the price nowadays for robotic technology going down and the capabilities bestowed upon such devices by AI research increasing, autonomous robots will also become ordinary devices found at homes, public institutions, healthcare institutions, and others. In order for this vision to become a reality, AI-equipped devices, particularly robots, must be able not only to co-exist with one another but also cooperate with humans. In such scenarios, robots should be able to seamlessly integrate existing teams, which is the fundamental challenge of *ad hoc* teamwork.

Even though the setting of *ad hoc* teamwork has greatly fallen under the attention of the multi-agent systems community the limitation with most approaches is that they consider relatively strong assumptions, unrealistic for real-world scenarios in which robots operate with other humans, such as full observability of the environment and real-time access to the actions their teammates are performing.

Given that the assumptions made by current work prevent the approaches from being directly applied to real-world scenarios comprised of both robots and human

FIGURE 1.1: A depiction of the *ad hoc* teamwork fallen cyclist example introduced by Stone et al. [2010b]. *"Imagine that you are in a foreign country where you do not speak the language, walking alone through a park. You see somebody fall off of his bicycle and injure himself badly; there are a few other people in the area, and all of you rush to help the victim. There are several things that need to be done. Somebody should call an ambulance, someone should check that the victim is still breathing, and someone should try to find a nearby doctor or policeman. However, none of you know one another, and thus you do not know who has a mobile phone, who is trained in first aid, who can run fast, and so forth. Furthermore, not all of you speak the same language. Nonetheless, it is essential that you quickly coordinate towards your common goal of maximising the victim's chances of timely treatment and survival. This scenario is an example of what we call an ad hoc team setting." [Stone et al., 2010b].* Image adapted from `https://www.theglobeandmail.com`

teammates, this thesis fills the gap in the literature by explicitly dropping these assumptions, namely: (i) the *ad hoc* agent is unable to perfectly observe the environment, being limited by a set of sensors, (ii) the *ad hoc* agent is unable to observe the actions of the team and (iii) teams may not only contain legacy robots but also human teammates.

1.1 Research Problem

This thesis addresses the following research question:

> *How can an autonomous agent assist unknown teammates in performing unknown tasks under more realistic conditions, without being able to pre-coordinate or pre-communicate with the existing team?*

We break down this research question into four sub-problems:

- How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?
- How can an autonomous agent assist unknown teammates in performing unknown tasks without being able to observe their actions, the full state of the environment or both?

- How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?
- How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?

1.1.1 How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?

Unlike in traditional settings such as reinforcement learning, which evaluates how well an agent is able to perform a task after being given time to learn it via trial-and-error, the setting of *ad hoc* teamwork evaluates an agent according to how well it is able perform a task on-the-fly after being deployed into an unknown team. Reinforcement learning approaches are often employed within the setting of *ad hoc* teamwork. For instance, a common approach, introduced by Barrett et al. [2017] is to set up a library of pre-trained policies, i.e., mappings between states/observations and actions to execute, each trained in a purely reinforcement learning setting, and identifying on-the-fly which one should be used to act upon an unknown scenario, depending on the teammate identified. However, if the teammates or task they are performing have never been encountered before, a completely new reinforcement learning policy has to be learned from scratch, not only having a potential catastrophic impact on the team's performance, but also rendering *ad hoc* teamwork as a task of pure reinforcement learning.

To mitigate this issue, we start by addressing in chapter 4 our first research sub-problem — *How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?* To more efficiently adapt to never before seen teammates and tasks, we introduce a novel model-based approach for *ad hoc* teamwork which exploits common knowledge by continuously learning an updating a world model. We show that our approach TEAMSTER, jointly published at the Artificial Intelligence Journal [Ribeiro et al., 2023b] and in the Proceedings of the 38th AAAI Conference on Artificial Intelligence [Ribeiro et al., 2024b], can more efficiently adapt to teammates and tasks on-the-fly, when these teammates and tasks have never before been encountered.

1.1.2 How can an autonomous agent assist unknown teammates in performing unknown tasks without being able to observe their actions, the full state of the environment or both?

While in computer simulated scenarios *ad hoc* agents may have access to information such as the states of the environment and the actions other teammates are executing in each time step, in real-world such pieces of information are seldom available. In real-world scenarios, agents, usually robots, are able to collect observations through a set of usually imperfect sensors. Furthermore, other agents, whether robotic or human, may not broadcast in real-time what actions they are performing or have just performed. Literature in *ad hoc* teamwork, however, often assumes these two pieces of information — (i) the state of the environment and (ii) the actions of the teammates — to be freely available to the *ad hoc* agent in every time step. In chapter 5, we drop these assumptions and address our second research sub-problem — *How can an autonomous agent assist unknown teammates in performing unknown tasks without being able to observe their actions, the full state of the environment or both?* We present two

novel approaches that rely on online Bayesian inference to identify the most likely teammates and tasks from a given set. The first approach, named BOPA, published at the Proceedings of the 20th EPIA Conference on Artificial Intelligence [Ribeiro et al., 2021] and winner of the conference's Best Paper Award, drops the assumption that the actions of the teammates are available to the *ad hoc* agent and identifies the most likely teammates and task being performed relying instead on the transitions of the environment. Given that BOPA still assumes the environment to be fully observable, we then present a second approach, named ATPO, published at the Proceedings of the 26th European Conference on Artificial Intelligence [Ribeiro et al., 2023a], that drops this assumption and identifies the most likely teammates and tasks being performed, relying only on partial observations.

1.1.3 How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?

Another common characteristic of more realistic scenarios is the complexity of their observation spaces. Agents are often provided raw input such as images from a camera feed, or audio from a microphone, which require some degree of feature engineering to extract a feature vector best describing each observation. In other fields such as reinforcement learning, a common way to deal with such challenges is to employ function-approximation techniques such as the ones from the field of Deep Learning [LeCun et al., 2015]. These techniques, famous for their automatic ability to extract reliable feature representations, also known as latent feature vectors, provide a solution for this problem of high-dimensional observation spaces. In chapter 6, we expand upon ATPO, addressing our third research sub-problem — *How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?* We present a novel approach for *ad hoc* teamwork under partial observability capable of identifying the most likely teammates and tasks being performed, relying this time on complex high-dimension observations.

1.1.4 How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?

Simulated environments, where most *ad hoc* agent approaches are evaluated in, even though they provide a reliable way to empirically evaluate agents, often lack a lot of the subtleties from real-world environments, such as having other humans as teammates. Humans are unpredictable, and modelling their behaviour in simulated environments may not always be straightforward. To mitigate this issue, we address in chapter 7 our fourth and final research sub-problem — *How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?* This seminal work showcases how results achieved in simulated environments may be effectively replicated in real-world ones. This contribution [Ribeiro et al., 2024a] is published at the Public Library of open Science (PLoS ONE) Journal.

1.2 Contributions

To summarise, this thesis contributes to the literature in *ad hoc* teamwork by proposing and evaluating:

- TEAMSTER (chapter 4), a model-based reinforcement learning approach for *ad hoc* teamwork which allows an *ad hoc* agent to continuously adapt to new teammates and tasks by efficiently reusing environmental knowledge. This contribution is jointly published at the Artificial Intelligence Journal [Ribeiro et al., 2023b] and in the Proceedings of the 38th AAAI Conference on Artificial Intelligence [Ribeiro et al., 2024b].

- BOPA and ATPO (chapter 5), two Bayesian online approaches for *ad hoc* teamwork capable of identifying and assisting unknown teammates in performing unknown tasks without needing to observe their actions. BOPA, published at the Proceedings of the 20th EPIA Conference on Artificial Intelligence [Ribeiro et al., 2021] and winner of the conference's Best Paper Award, performs this inference by relying on the state of the environment and ATPO, published at the Proceedings of the 26th European Conference on Artificial Intelligence [Ribeiro et al., 2023a], expands upon BOPA by relying on partial observations instead.

- ATLPO (chapter 6), an approach capable of identifying and assisting unknown teammates in performing unknown tasks in arbitrarily large partially observable domains without needing to observe their actions. Relying on state-of-the-art deep learning techniques to identify both the teammates and the tasks being performed, ATLPO relies on a library of state-of-the-art reinforcement learning policies to assist the team.

- The HOTSPOT framework (chapter 7), an *ad hoc* teamwork platform for mixed human-robot teams that enables the deployment of *ad hoc* approaches into real-world scenarios. This contribution is published at the Public Library of open Science (PLoS ONE) Journal [Ribeiro et al., 2024a].

1.3 Outline

The remainder of this thesis is organised as it follows:

- Chapter 2 lays the notation used throughout the remainder of this thesis and provides an in-depth discussion of the background concepts required for the understanding of the posterior chapters.

- Chapter 3 discusses the present literature in *ad hoc* teamwork and its closely related areas, while framing this thesis and its contributions within the landscape.

- Chapter 4 presents and discusses TEAMSTER, a novel approach for *ad hoc* teamwork which addresses our first research sub-problem.

- Chapter 5 presents and discusses BOPA and ATPO, two novel approaches for *ad hoc* teamwork which addresses our second research sub-problem.

- Chapter 6 presents and discusses ATLPO, a novel approach for *ad hoc* teamwork which addresses our third research sub-problem.

- Chapter 7 presents and discusses the HOTSPOT platform, a novel approach for *ad hoc* teamwork which addresses our fourth and final research sub-problem.

- Chapter 8 summarises this thesis, its main contributions and discusses further research directions.

Chapter 2

Background

2.1 Markov Decision Problems

Markov Decision Problems (MDPs) [Puterman, 2005] model the interaction between an agent and an environment which can be divided it into discrete time steps (figure 2.1).

In each time step t the environment has an internal state X_t which uniquely describes it. The agent, provided or observing state X_t is allowed to execute an action A_t. The effect of this action transitions the environment into a next time step $t+1$ and next state X_{t+1}. In an MDP the set of all possible states is known as the *state space* $\mathcal{X}$ and the set of all possible actions known as *action space*) $\mathcal{A}$. Given a state X_t and an action A_t, a transition into a next state X_{t+1} takes into account the environments *transition probabilities* $P(y \mid x, a) \in [0,1]$, i.e.

$$P(y \mid x, a) \triangleq \mathbb{P}\left[X_{t+1} = y \mid X_t = x, A_t = a\right].$$

In the next time step $t+1$, the agent's action is evaluated immediately according to a reward function *reward function* $r(x,a)$ receiving an immediate reward R_{t+1}, i.e.

$$r(x,a) \doteq \mathbb{E}\left[R_{t+1} \mid X_t = x, A_t = a\right].$$

Both the transition probabilities and the reward function follow the *Markov Property*, i.e., for any history $H_t = \{X_0, A_0, R_1, X_1, A_1, ..., X_t\}$,

$$\mathbb{P}\left[X_{t+1} = P(y \mid x,a) \mid H_t = h\right] = \mathbb{P}\left[X_{t+1} = P(y \mid x,a) \mid X_t = x, A_t = a\right]$$

and

$$\mathbb{P}\left[R_{t+1} = r(x,a) \mid H_t = h\right] = \mathbb{P}\left[R_{t+1} = r(x,a) \mid X_t = x, A_t = a\right]$$

An MDP may therefore be represented as a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, P, r, \gamma, \mu)$, where the additional parameter $\gamma \in (0,1]$ allows for specifying how much immediate rewards are valued over non-immediate rewards, a parameter known in the literature

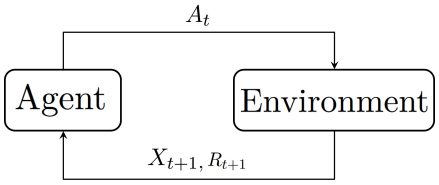


FIGURE 2.1: Flowchart for a Markov Decision Problem. In a time step t, an agent executes an action A_t on the environment and is rewarded with R_{t+1} while the environment transitions into a next state X_{t+1}.

as the *discount factor* and the additional parameter $\mu = \Delta(\mathcal{X})$ represents an initial distribution over the state space $\mathcal{X}$.

2.1.1 Return, Policies, Value Functions and Action-Value Functions

When starting a new interaction in a state $X_t \sim \mu$, the goal for the agent is to select its action in a way that maximises its accumulation of future discounted rewards, a metric known in the literature as the *return* G_t,

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

This decision making process is usually represented by a decision function commonly referred to as a *policy* $\pi(x)$. For each state $x \in \mathcal{X}$, a policy $\pi(x) : \mathcal{X} \to \Delta(\mathcal{A})$ maps x into a distribution over all possible actions $\Delta(\mathcal{A})$, from which an action to execute can be sampled. Since the goal is to select actions in a way that maximises the expected return $\mathbb{E}_\pi[G_t]$, policies are evaluated according to this exact metric. For a given policy $\pi(x)$ an evaluation function commonly referred to as a *value function* $v^\pi(x)$ can be computed as

$$\begin{aligned} v^\pi(x) &= \mathbb{E}_\pi[G_t \mid X_t = x] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid X_t = x\right] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid x) r(x,a) + \gamma \sum_{y \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(a \mid x) P(y \mid x, a) v^\pi(y). \end{aligned} \tag{2.1}$$

from which it can be also derived a function that evaluates the policy if a specific first action a is to be executed on x, often referred to as the *action-value function* or *q-value function* $q^\pi(x,a)$,

$$\begin{aligned} q^\pi(x,a) &= \mathbb{E}_\pi[G_t \mid X_t = x, A_t = a] = \mathbb{E}_\pi\left[r(x,a) + \sum_{k=1}^{\infty} \gamma^k R_{t+1+k} \mid X_t = x, A_t = a\right] \\ &= r(x,a) + \gamma \sum_{y \in \mathcal{X}} \sum_{a' \in \mathcal{A}} P(y \mid x, a) \pi(a' \mid y) q^\pi(y, a') \end{aligned}$$

All methods for solving an MDP, i.e., computing the best possible policy $\pi^*(x)$ rely on evaluating the policy via its value or action-value functions $v^\pi(x)$ and $q^\pi(x,a)$

2.1.2 Solving an MDP

Solving an MDP consists on computing a policy π^* such that, for all states $x \in \mathcal{X}$ and other policies π, $v^{\pi^*}(x) \geq v^\pi(x)$. This solution, known as the MDPs' *optimal policy* has a value function defined as

$$\begin{aligned} v^*(x) &= \max_\pi v^\pi(x) = \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid X_t = x\right] \\ &= \max_{a \in \mathcal{A}} \left[r(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y \mid x, a) v^*(y)\right] \end{aligned}$$

which can be computed using dynamic programming through algorithms such as *value iteration* [Puterman, 2005, Sutton and Barto, 1998].

From the optimal value function v^*, it is also useful to derive the *optimal action-value* function $q^* = q^{\pi^*}$, with

$$q^*(x,a) = \max_\pi q^\pi(x,a) = \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} G_t \mid X_t = x, A_t = a\right]$$
$$= r(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y \mid x,a) \max_{a' \in \mathcal{A}} q^*(y,a')$$

Finally, the optimal policy π^* is represented as an uniform distribution over all actions a^* that maximise the optimal action-value function q^*, i.e.

$$\pi^*(a|x) = \frac{\mathbb{I}[a \in \mathcal{A}^*(x)]}{|\mathcal{A}^*(x)|}$$

where

$$\mathcal{A}^*(x) = \operatorname*{argmax}_{a \in \mathcal{A}} q^*(x,a), \forall x \in \mathcal{X}$$

2.1.3 Planning in MDPs

Given the stochastic nature of the transitions in MDPs, a very common planner used for decision-making in MDPs is the Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006].

MCTS is a planner which resorts to Monte Carlo iterations in order to compute an estimate for the action-values $q(x,a)$ of a given state x. The more iterations are ran, the better the final estimate of $q(x,a)$. Before the first iteration, an estimator $\hat{Q}_0(x,a) = \mathbf{0}$ is initialised. Each iteration i then updates $\hat{Q}_i(x,a)$ with four base steps: (i) Selection, (ii) Expansion, (iii) Simulation and (iii) Backup. Figure 2.2 illustrates the process of a single iteration.

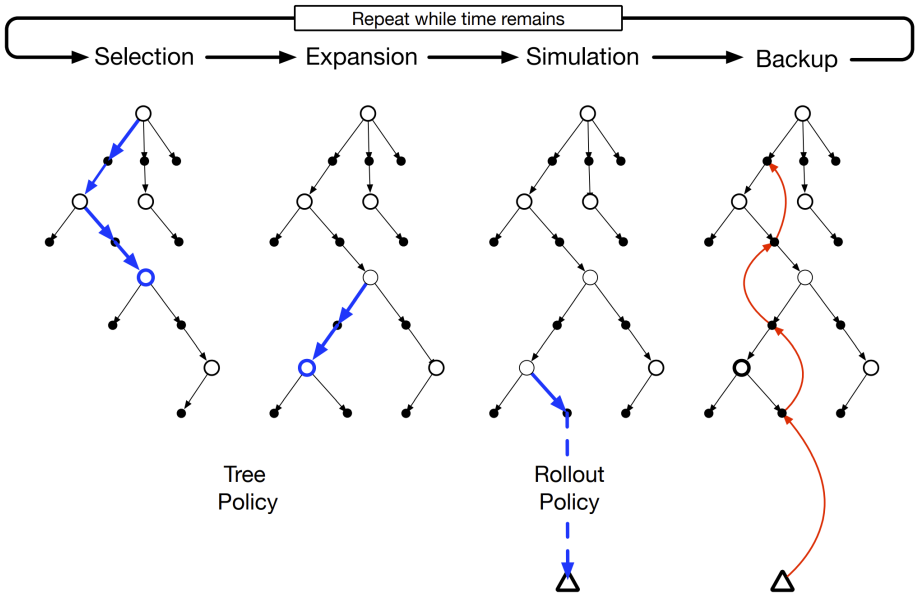


FIGURE 2.2: Illustration of the four steps of an MCTS iteration. Figure from Sutton and Barto [2018].

Selection

In the first step of iteration i, the MCTS planner selects an action node of a state x to be expanded. An action node n contains not only an associated state $n(x)$ and action $n(a)$ but also a counter N_i^n which keeps track of how many times the node has been selected for expansion. All states in the tree, showcased in white in figure 2.2, are initialised with $\mathcal{A}$ unexpanded action nodes (one per each action in $\mathcal{A}$, showcased in black in figure 2.2). From all action nodes n in the tree $\mathcal{T}$, one is selected in iteration i according to the following upper confidence bound

$$n_i = \underset{n \in \mathcal{N}}{\operatorname{argmax}} \hat{Q}_i(n(x), n(a)) + \epsilon \sqrt{\frac{2 \ln i}{N_i^n}}$$

where ϵ is an exploration factor (usually $\sqrt{2}$ for rewards $R \in [0, 1]$).

After selecting node n_i, with its corresponding action, $a_i = n_i(a)$, and its corresponding state $x_i = n_i(x)$, the MCTS planner moves on to the second step — Expansion.

Expansion

After selecting the action a_i, the planner expands the current node by sampling a next state y_i, using the model's transition probabilities $P(y \mid x, a_i)$. The planner then estimates the action values for y_i by performing the next step, Simulation.

Simulation

Starting in y_i in a depth $d = 0$, until a maximum depth $d = D$ is reached, the MCTS planner now resorts to a *rollout policy* π, recursively selection actions a_i^d. In a given intermediary depth d, state y_i^d is expanded using $a_i^d \sim \pi$ (where states y_i^{d+1} are sampled using $P(y \mid y_i^d, a_i^d)$). In each depth d, the reward $R_d = R(y_i^{d-1}, a_i^{d-1})$ is stored. When the maximum depth D is reached, the final reward $R_D = R(y_i^{D-1}, a_i^{D-1})$ is then propagated upwards using the final Backup step.

Backup

In the final step of one MCTS iteration, the reward obtained by the Simulation step, R_D, is propagated upwards by computing $q_D = R_D$. Then, for each depth $d = D - 1, \ldots, 0$, values q_d are computed using

$$q_d = R_d + \gamma q_{d+1}$$

Finally, having q_0 computed, the action-value estimate for the current iteration i, $\hat{Q}_i(x_i, a_i)$, is updated using

$$\hat{Q}_i(x_i, a_i) = \frac{q_0 - \hat{Q}_{i-1}(x_i, a_i)}{N_i^{n_i}}$$

where $\hat{Q}_{i-1}(x_i, a_i)$ is the estimate from the previous iteration and $N_i^{n_i}$ is the number of times node n_i has been visited until iteration i.

The process can be repeated for any arbitrary number of iterations, with $\hat{Q}_i(x, a) \approx q(x, a)$ as $i \to \infty$.

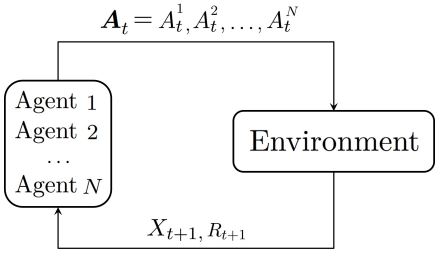


FIGURE 2.3: Flowchart for an Multi-Agent Markov Decision Problem. In a time step t, each agent executes an action A_t^n on the environment and the team is rewarded with R_{t+1} while the environment transitions into a next state X_{t+1}.

2.1.4 Multi-Agent Markov Decision Problems

Multiagent MDPs (MMDPs) are an extension of MDPs to multiagent settings. An MMDP can be described as a tuple $\mathcal{M} = (N, \mathcal{X}, \mathcal{A} = \prod_{n=1}^{N} \mathcal{A}^n, P, r, \gamma)$ where N is the number of agents, $\mathcal{X}$ is the state space, $\mathcal{A}^n$ is the individual action space for agent n, P are the transition probabilities, r is the expected reward function, and γ is the discount factor. Fig. 2.3 showcases the flowchart for an MMDP.

We write $\mathcal{A}$ to denote the set of all joint actions, corresponding to the Cartesian product of all individual action spaces $\mathcal{A}^n$, i.e., $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times \ldots \times \mathcal{A}^N$. We also denote an element of $\mathcal{A}^n$ as a^n and an element of $\mathcal{A}$ as a tuple $\boldsymbol{a} = (a^1, \ldots, a^N)$, with $a^n \in \mathcal{A}^n$. We write $\boldsymbol{a}^{-n}$ to denote a reduced joint action, i.e., a tuple $\boldsymbol{a}^{-n} = (a^1, \ldots, a^{n-1}, a^{n+1}, \ldots, a^N)$, and thus $\mathcal{A}^{-n}$ is the set of all reduced joint actions. We adopt, for policies, a similar notation. Specifically, we write π^n to denote an individual policy for agent n, $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^N)$ to denote a joint policy, and $\boldsymbol{\pi}^{-n} = (\pi^1, \ldots, \pi^{n-1}, \pi^{n+1}, \ldots, \pi^N)$ to denote a reduced joint policy.

Reducing MMDPs to MDPs

A MMDP is just an MDP in which the action selection process is distributed across N agents, and can be solved by computing $\boldsymbol{\pi}^*$ from v^* as in standard MDPs. If we control agent n and assume that we know the policies of the other agents, $\boldsymbol{\pi}^{-n}$, we can reduce an MMDP $\mathcal{M} = (N, \mathcal{X}, \mathcal{A} = \prod_{n=1}^{N} \mathcal{A}^n, P, r, \gamma)$ for agent n to an MDP $\mathcal{M}^n = (\mathcal{X}, \mathcal{A}^n, P^n, r^n, \gamma)$, where

$$\begin{aligned} P^n(y \mid x, a^n) &\triangleq \mathbb{P}\left[X_{t+1} = y \mid X_t = x, A_t^n = a^n, \boldsymbol{A}_t^{-n} \sim \boldsymbol{\pi}^{-n}\right] \\ &= \sum_{a^{-n} \in \mathcal{A}^{-n}} P(y \mid x, (a^n, a^{-n})) \pi^{-n}(a^{-n} \mid x) \end{aligned} \tag{2.2}$$

which can then be solved to obtain the optimal policy $\pi^{*,n}$.

2.1.5 Partially Observable Markov Decision Problems

Partially observable MDPs, or POMDPs, are an extension of MDPs to partially observable settings. A POMDP can be described as a tuple $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, P, O, r, \gamma)$, where $\mathcal{X}$, $\mathcal{A}$, P, r, and γ, are the same as in MDPs, $\mathcal{Z}$ is the *observation space* and O are the *observation probabilities*. Fig. 2.4 showcases the flowchart for a Partially Observable Markov Decision Problem.

When the agent executes an action A_t in state X_t, the environment transitions into a next state X_{t+1} according to P, and the agent, instead of observing X_{t+1}, obtains

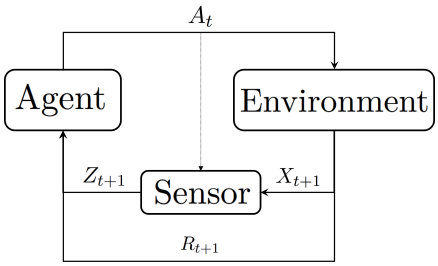


FIGURE 2.4: Flowchart for a Partially Observable Markov Decision Problem. In a time step t, the agent executes an action A_t^n on the environment, is rewarded with R_{t+1} and obtains a partial observation Z_{t+1} of the next state of the environment X_{t+1}.

instead a partial observation Z_{t+1} sampled from the observation probabilities O, with

$$O(z \mid y, a) \triangleq \mathbb{P}\left[Z_{t+1} = z \mid X_{t+1} = y, A_t = a\right].$$

In a time step t, the agent has a belief $b_t(x)$ over each possible state of the environment. $b_t(x)$ represents the probability that $X_t = x$ given the current history, and can be defined as

$$b_t(x) \triangleq \mathbb{P}\left[X_t = x \mid X_0 \sim \boldsymbol{b}_0, A_0 = a_0, Z_1 = z_1, A_1 = a_1 \ldots, Z_t = z_t\right],$$

where $\boldsymbol{b}_0$ is the initial state distribution. Given the action A_t and the observation Z_{t+1}, we can update the belief b_t to incorporate the new information yielding

$$\begin{aligned} b_{t+1}(y) &= \left[\mathrm{Bel}(b_t, a_t, z_{t+1})\right]_y \\ &\triangleq \frac{1}{\rho_{t+1}} \sum_{x \in \mathcal{X}} b_t(x) P(y \mid x, a_t) O(z_{t+1} \mid y, a_t), \end{aligned} \tag{2.3}$$

where ρ_{t+1} is a normalisation constant. Every finite POMDP admits an equivalent *belief-MDP* where b_t being the state of this new MDP at time step t. A policy in a POMDP can thus be seen as mapping π from beliefs to distributions over actions, and we define

$$v^\pi(b) \triangleq \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid b_0 = b\right]. \tag{2.4}$$

As in MDPs, the value function associated with an optimal policy is denoted as v^* and can be computed using, for example, point-based approaches [Pineau et al., 2006]. From v^*, the optimal Q-function can now be computed as

$$q^*(b, a) = \sum_{x \in \mathcal{X}} b(x) \left[r(x, a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} P(y \mid x, a) O(z \mid y, a) v^*(\mathrm{Bel}(b, a, z))\right],$$

yielding as optimal any policy π^* such that $\pi^*(a \mid b) > 0$ only if $a \in \operatorname{argmax}_{a \in \mathcal{A}} q^*(b, a)$.

Solving a POMDP

Similarly to an MDP, the value function of a POMDP verifies

$$v^*(b) = \max_{a \in \mathcal{A}} \left[\sum_{x \in \mathcal{X}} R(x, a) b(x) + \gamma \sum_{y \in \mathcal{X}} \sum_{z \in \mathcal{Z}} P(y \mid x, a) O(z \mid y, a) v^*(\mathrm{Bel}(b, z, a))\right] \tag{2.5}$$

From the value function $v^* : \mathcal{X} \to \mathbb{R}$, we can extract a policy $\pi : \Delta(\mathcal{X}) \to \Delta(\mathcal{A})$.

Given that the value function of a POMDP is piecewise linear and convex (PWLC), such structure can be exploited and we may represent $v^*(b)$ using a finite set of vectors (often called α-vectors), each with an associated optimal action. In other words, given a pre-computed set of α-vectors $\Gamma = \{\alpha_0, \alpha_1, ..., \alpha_n\}$, $v(b)$ can be given by

$$v(b) = \max_{\alpha \in \Gamma} b \cdot \alpha \tag{2.6}$$

We can consider two main classes of solvers capable of computing Γ for a given POMDP. Exact methods (e.g. Incremental Pruning, Witness [Cassandra et al., 2013, Littman, 1994]), for instance, are able to compute a set Γ which represents the true optimal value function of the POMDP.

However, given that POMDPs are PSPACE-complete [Papadimitriou and Tsitsiklis, 1987], such methods do not scale well with the total number of states, actions and observations. The state-of-the-art in POMDP solvers has therefore shifted more towards Point-based methods (e.g. Perseus [Spaan and Vlassis, 2005]), which approximates Γ by computing α-vectors in a given set of beliefs (or points).

In the case of Perseus algorithm [Spaan and Vlassis, 2005] (used by this thesis), a set of beliefs $\hat{\mathcal{B}}$ is randomly sampled from the POMDP. Then, in each iteration k, Perseus computed a new set of α vectors Γ^k for sample $\hat{\mathcal{B}}$ and ignores beliefs $b \in \hat{\mathcal{B}}$ which have their value dominated by others, i.e, $v^k(b) < v^k(b'), b' \in \hat{\mathcal{B}}$ are ignored. Whenever all points have been ignored, Perseus returns the current set of α vectors Γ.

2.2 Reinforcement learning

Exact methods for solving MDPs, MMDPs or POMDPs, such as value iteration [Littman, 2001], require access to the model itself, in particular its transition probabilities $P(y \mid x, a)$ and reward function $r(x, a)$. Agents, however, might not always be provided this information. Being only given the possibility of interacting with the environment, a family of methods known as *Reinforcement Learning* computes policies by instead relying on trial-and-error. In each interaction time step t, the agent has access to the current state of the environment, or partial observation depending on whether the environment is fully or partially observable, and is able to execute actions. After executing an action and receiving a reward signal, the agent can use this time step data sample $(X_t, A_t, R_{t+1}, X_{t+1})$ to update its current policy. In this section we discuss two main classes of reinforcement learning methods — (i) *model-based reinforcement learning* and *model-free reinforcement learning*.

2.2.1 Model-Based Reinforcement Learning

Model-based reinforcement learning methods, as the names suggests learn the underlying model of the environment, i.e. transition probabilities $P(y \mid x, a)$, reward function $r(x, a)$ and observation probabilities $O(z \mid y, a)$ in order to use traditional planning methods to compute the optimal policy π^*.

For instance, when assuming full observability the environment is in a state X_t when the agent executes an action A_t and the environment transitions into state X_{t+1} according to some transition probabilities unknown to the agent, a maximum likelihood estimator can be updated from the data sample $s_t = (X_t, A_t, X_{t+1}, R_{t+1})$

$$\hat{P}^t(y \mid X_t, A_t) = \hat{P}^{t-1}(y \mid X_t, A_t) + \frac{1}{N_t(X_t, A_t)} \left[\mathbb{I}[X_{t+1} = y] - \hat{P}^{t-1}(y \mid X_t, A_t) \right] \forall y \in \mathcal{X} \tag{2.7}$$

Similarly, an estimator for the reward function, $\hat{r}(x, a)$, may also be updated from s_t

$$\hat{r}^t(X_t, A_t) = \hat{r}^{t-1}(X_t, A_t) + \frac{1}{N_t(X_t, A_t)} (R_{t+1} - \hat{r}^{t-1}(X_t, A_t)) \tag{2.8}$$

In both cases, N_t is the number of times state-action pair (X_t, A_t) has been visited. Given the estimators for both the transition probabilities, $\hat{P}$ and reward function $\hat{r}$, an estimator for the optimal action-values may also be updated in every time step

$$\hat{Q}^t(X_t, A_t) = \hat{r}^t(X_t, A_t) + \gamma \sum_{y \in \mathcal{X}} \hat{P}(y \mid X_t, A_t) \max_{a \in \mathcal{A}} \hat{Q}^{t-1}(y, a) \tag{2.9}$$

2.2.2 Model-Free Reinforcement Learning

Model-free reinforcement learning methods try to estimate the optimal action-value function directly, without learning the model (i.e., without learning the transition probabilities or reward function).

The most notable way of estimating the action-value function is by setting up and updating an estimator from samples $s_t = (X_t, A_t, X_{t+1}, R_{t+1})$ in each time step (an algorithm called *q-learning* [Watkins, 1989]):

$$\hat{Q}^t(X_t, A_t) = \hat{Q}^{t-1}(X_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} \hat{Q}^{t-1}(X_{t+1}, a) - \hat{Q}^{t-1}(X_t, A_t) \right] \tag{2.10}$$

where $\alpha \in (0, 1]$ is the step size of the updates.

2.2.3 Non-Tabular Methods, Function Approximation Techniques and Deep Reinforcement Learning

When dealing with arbitrarily large or continuous state spaces, learning the action-value function for all state-action pairs becomes unfeasible. Function approximation methods tackle this limitation by learning abstract representations of the states/observations when given sets of features $\phi(x)/\phi(z)$. Usually implemented using feedforward networks, hidden layers in these models can be trained to extract abstract (or *latent*) feature representations of the states/observations, implicitly allowing for the learning of similarities between states/observations with different features $\phi(x)/\phi(z)$.

Two examples of deep model-free reinforcement learning approaches are deep q-networks [Mnih et al., 2015] and deep policy gradients [Schulman et al., 2017]. Parametrised models in these algorithms take as input the aforementioned sets of features $\phi(x)/\phi(z)$ and are usually trained end-to-end to approximate a particular function — the action-value function $\hat{q}(x, a; \theta)$ in the case of deep q-networks and a policy function $\hat{\pi}(a \mid x; \theta)$ in the case of deep policy gradients.

Similarly, model-based methods can also benefit from function approximation methods, usually employing them not only to approximate the transition and reward functions $\hat{P}(y \mid x, a; \theta)$, $\hat{R}(x, a; \theta)$ of the environment but also, in the case of partially

observable domains, learn good state representations. Two examples are the MuZero [Schrittwieser et al., 2020b] and Dreamer [Hafner et al., 2019b,a, 2020, 2023a] algorithms. Both MuZero and DreamerV3 [Hafner et al., 2023a] train an encoder network $q(z;\theta)$ to learn a latent state representation $\hat{X}_t$. MuZero, on one hand, trains the encoder alongside a following transition function $\hat{P}(y \mid x, a;\theta)$ and reward function $\hat{R}(x, a;\theta)$ all end-to-end. Then, given a a set of features describing an observation Z_t, an initial latent state $\hat{X}_t$ is extracted and $\hat{P}(y \mid x, a;\theta)$, $\hat{R}(x, a;\theta)$ alongside a planner such as UCT (section 2.1.3) to compute an action. DreamerV3, on the other hand, first trains the encoder $q(z;\theta)$ to learn a latent state representation $\hat{X}_t$ by training an additional decoder $d(\hat{x};\theta)$ to reconstruct the original observation $\hat{Z}_t$. Latent states $\hat{X}_t$ are then used to train a standard model-free reinforcement learning algorithm as if the environment was fully observable.

Chapter 3

Related Work

This Chapter discusses the current state-of-the-art in the setting of ad hoc teamwork and frames the contributions of this thesis within the available literature.

3.1 *Ad Hoc* Teamwork

Introduced by Stone et al. [2010b], the multi-agent systems setting of ad hoc teamwork showcases a scenario in which an autonomous agent is deployed for a specific purpose into an existing team of agents. Stone et al. define the setting of ad hoc teamwork as a setting where an ad hoc agent is expected to assist an unknown team of agents in performing their cooperative task, without any pre-coordination or explicit communication protocols available. While virtually all available works so far on ad hoc teamwork have focused on developing agents capable of assisting existing team without pre-coordination or pre-communication, most of them diverge in terms of other key assumptions made — namely, whether the ad hoc agent knows the task being performed beforehand, the behaviour (or policies) of the teammates, whether there are available reward signals from the environment and whether the ad hoc agent can observe the actions of the teammates.

3.1.1 Problem Formulation

Throughout this thesis, we formulate the problem of ad hoc teamwork as a multi-agent system where a single *ad hoc* agent, denoted as α, interacts with a team of $N-1$ teammates. We reduce the N agent system to a 2 agent system by modelling the teammates as a single "meta agent", named $-\alpha$.

When α is deployed into the environment where $-\alpha$ operate, we assume an underlying MMDP $M = (2, \mathcal{X}, \{\mathcal{A}^{\alpha}, \mathcal{A}^{-\alpha}\}, P, r, \gamma)$ and $-\alpha$ to follow some policy $\pi^{-\alpha}$. Both the model M and policy $\pi^{-\alpha}$ are unknown, *a priori*, to the ad hoc agent α. Figure 3.1 illustrates the interaction between the ad hoc agent and the teammates with the environment.

Using this framework, we define the problem of ad hoc agent as following the policy π^{α} which maximises the return interacting with the teammates in their environment without knowing M and $\pi^{-\alpha}$ beforehand, i.e.

$$\pi^{*\alpha} = \max_{\pi^{\alpha}} \quad \mathbb{E}\left[G_t \mid A_t^{\alpha} \sim \pi^{\alpha}, A_t^{-\alpha} \sim \pi^{-\alpha}\right]$$

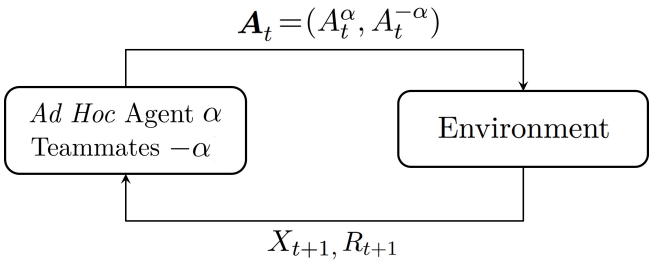


FIGURE 3.1: Interaction between an ad hoc agent α and teammates $-\alpha$ described by an MMDP M.

3.1.2 Literature Review

Earlier works started by assuming the ad hoc agent knew both the task (or model M) and team's policy $\pi^{-\alpha}$ were known to the ad hoc agent α. With this information, planners using both M and $\pi^{-\alpha}$ could directly obtain the best A_t^{α} for a state X_t [Stone and Kraus, 2010]. Stone et al. [2009] showcased an example using these assumptions, by introducing a finite-horizon decision problem where the ad hoc agent α plans its actions A_t^{α} to lead a single teammate $-\alpha$ (where $\pi^{-\alpha}$ is not only optimal but also known to the ad hoc agent α) to achieve a common goal. The authors later on extended their previous work to discounted infinite horizon problems Barrett and Stone [2011], while Agmon and Stone [2012] modified the scenario to consider N teammates instead of a single one.

In another work, Stone et al. [2010c] considered a situation where the ad hoc agent α was paired with a bounded-memory teammate $-\alpha$ following an ϵ-greedy policy. Barrett et al. [2014] recently explored the same setting, but now considering communication between the agents (where the authors showed that describing the planning problem by using decision-theoretic models leads to more efficient solutions). All aforementioned works, however, assume that the ad hoc agent has full access to (i) the task's dynamics and goal (model M) and (ii) the teammates' policies $\pi^{-\alpha}$, which can be argued that reduces ad hoc teamwork to a problem of planning (Fig. 3.2).

Before the introduction of ad hoc teamwork Stone et al. [2010b], Fern et al. [2007] introduced the framework of *assistance* [Fern et al., 2007], which models scenarios where an agent α_1 (called the assistant) must help a unknown teammate α_2 in solving a given sequential task M under uncertainty. The authors assume a shared task M is known beforehand to α but the teammate's behavior π^{α_2} is not, and must therefore be identified. The setting also assumes that there is only a single teammate which the α may assist instead of a broader team. In their work [Fern et al., 2007], the authors

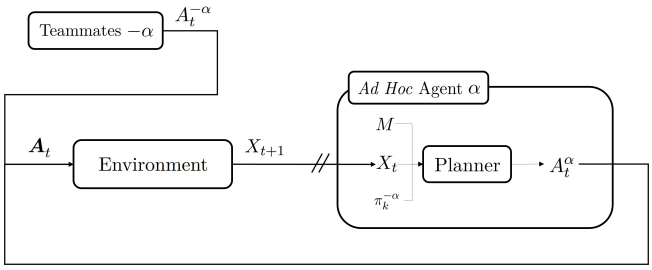


FIGURE 3.2: Ad hoc teamwork as a problem of planning. The ad hoc agent is assumed to know the task's dynamics and goal, represented by a model M, and the policy of the team $\pi^{-\alpha}$. This information, alongside the state of the environment X_t, can be fed to a planner in order to compute the best response A_t^{α} for the ad hoc agent.

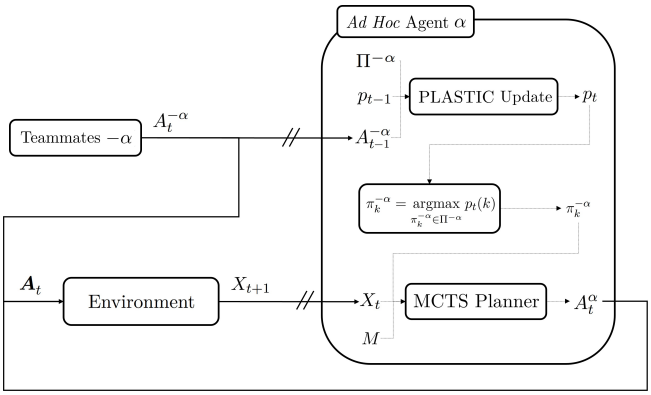


FIGURE 3.3: Overview architecture for the PLASTIC-Model algorithm [Barrett and Stone, 2015]. The ad hoc agent α is given a library of possible team policies $\Pi^{-\alpha}$ which it uses to perform Bayesian updates (Eq.3.1) to a prior distribution p. Using the most likely team policy alongside an assumed known model of the task, M, the ad hoc agent uses a MCTS planner in order to compute a best-response A_t^{α}.

consider that the teammate's actions $A_t^{\alpha_2}$ to be visible to the assistant α but the state of the environment to be partially observable.

More recently, Chakraborty and Stone [2013] start by dropping the assumption that the team's policy $\pi^{-\alpha}$ is known to the ad hoc agent (however still considering that the ad hoc agent knows model M). In their work [Chakraborty and Stone, 2013], the authors assume the teammates follow an unknown Markov policy $\pi^{-\alpha}$ which the ad hoc agent α has the goal of identifying. On a more similar line of work, Barrett and Stone [2015] expand upon Chakraborty and Stone [2013] by introducing an approach capable of identifying the current team policy $\pi^{-\alpha}$ (unknown beforehand to the ad hoc agent) from a handcoded library of possible policies $\Pi^{-\alpha} = \left\{\pi_1^{-\alpha}, \pi_2^{-\alpha}, \ldots, \pi_K^{-\alpha}\right\}$. Their approach, named PLASTIC-Model, then relies in the state of the environment X_t and the actions of the team $A_t^{-\alpha}$ in order to perform Bayesian updates to a prior distribution p over each possible team policy $\pi_k^{-\alpha} \in \Pi^{-\alpha}$ named the PLASTIC Update rule, i.e.

$$p_t(k) = \frac{1}{\rho}(1 - \eta l_k) p_{t-1}(k) \tag{3.1}$$

where ρ is some normalisation constant, $\eta \in (0, 1]$ is an hyperparameter which bounds the maximum loss l_k the update, which is given by

$$l_k = 1 - \pi_k^{-\alpha}(A_{t-1}^{-\alpha} \mid X_{t-1})$$

In the current time step, t, the most likely team policy $\pi_k^{-\alpha}$ given by

$$\pi_k^{-\alpha} = \underset{\pi_k^{-\alpha} \in \Pi^{-\alpha}}{\text{argmax}}\, p_t(k)$$

Finally, $\pi_k^{-\alpha}$ is fed to an MCTS Planner alongside the state of the environment X_t, which outputs the action of the ad hoc agent, A_t^{α}. Figure 3.3 showcases an overview of PLASTIC-Model's architecture.

The limitation with both PLASTIC-Model [Barrett and Stone, 2015] and the approach introduced by Chakraborty and Stone [2013], however, is that both assume the ad hoc agent knows the task's dynamics and goal (i.e., M). Under this assumption, it

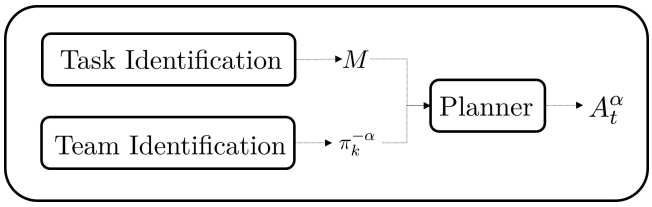


FIGURE 3.4: Ad Hoc Teamwork broken down into three sub-problems [Melo and Sardinha, 2016]. The ad hoc agent has the explicit goal of (i) identifying the task being performed by the team (dynamics and goal modelled by M), (ii) identifying the policy of the team π^{α} and (iii) planning using the task and team identification in order to compute a best response A_t^{α}.

can be argued that ad hoc teamwork problem is closely related to *learning in games*, a setting for which extensive literature exists [Fudenberg and Levine, 2000].

Later on, Melo and Sardinha [2016] formalise ad hoc teamwork by breaking it down into three sub-problems — i) task identification: the ad hoc agent has to identify the task's goals and dynamics (M), ii) team identification: the ad hoc agent has to identify the team's policy $\pi^{-\alpha}$ and iii) planning/execution: using both M and $\pi^{-\alpha}$, the agent α has to plan and execute a best-response A_t^{α} (Figure 3.4).

In their work, the authors address the ad hoc teamwork problem with unknown teammate behaviour, in a setting where the ad hoc agent is paired with a bounded-memory best-responder teammate in an unknown one-shot game. The target task M is assumed to belong to a library of possible tasks $\mathcal{M} = \{M_1, M_2, \ldots, M_K\}$, accessible to the ad hoc agent as prior knowledge. The agent α, however does not know in advance which it is. In this scenario, the ad hoc agent must simultaneously identify the teammate's policy $\pi^{-\alpha}$, while using it to identify their target task $M \in \mathcal{M}$. After identifying the target task, the ad hoc agent is now able to compute its optimal action A_t^{α}. Their approach, however, was tailored for one-shot games, where there are no environment transitions.

Afterwards, Barrett et al. [2017] drop the assumptions that the task's dynamics and goal M and team's policy $\pi^{-\alpha}$ are known to the agent α beforehand.

Similarly to their prior work, PLASTIC-Model [Barrett and Stone, 2015], their new approach, named PLASTIC Policy [Barrett et al., 2017], also relies on the state of the environment X_t and the actions of the team $A_t^{-\alpha}$ in order to perform Bayesian updates (Eq. 3.1) to a prior distribution p representing the beliefs over each possible task/team combination k.

The authors make two key changes to PLASTIC-Model — i) the ad hoc agent α is not assumed to have a model M of the task's dynamics and goal and ii) the ad hoc agent α, instead of having, as prior knowledge, access to a library of possible team policies $\Pi^{-\alpha}$, has now access to two policy libraries, Π^{α} and $\Pi^{-\alpha}$. $\Pi^{-\alpha} = \{\pi_1^{-\alpha}, \pi_2^{-\alpha}, \ldots, \pi_K^{-\alpha}\}$ represents the same library from PLASTIC-Model, which contains the policies for the possible teams, learned via supervised learning during previous interactions instead of being handcoded. Similarly, $\Pi^{\alpha} = \{\pi_1^{\alpha}, \pi_2^{\alpha} \ldots, \pi_K^{\alpha}\}$, trained via reinforcement learning in previous interactions, contains the policies for ad hoc agent α which allow for the best responses to each team $k \in \{1, \ldots, K\}$, when given a current state X_t. In each time step t, both the priors are updated using the PLASTIC Update (Eq. 3.1) and, using the updated priors, and action A_t^{α} is sampled from the policy π_k^{α} associated with the most likely team $\pi_k^{-\alpha}$. Figure 3.5 showcases an overview of PLASTIC-Model's architecture.

PLASTIC-Policy has, however, two main limitations. The first limitation is that PLASTIC Policy implicitly learns the task dynamics and goal (M) when learning the policies Π^{α} during an interaction with a team $-\alpha$. For this reason, PLASTIC-Policy

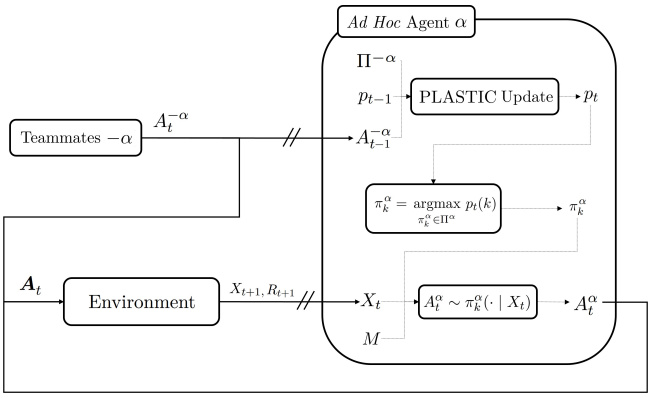


FIGURE 3.5: Overview architecture for the PLASTIC-Policy algorithm [Barrett et al., 2017]. The ad hoc agent α is given (i) a library of possible team policies $\Pi^{-\alpha}$, which it uses to perform Bayesian updates (Eq.3.1) to a prior distribution p, and (ii) a library of pre-trained policies for the ad hoc agent, Π^α. Each policy $\pi_k^\alpha \in \Pi^\alpha$ was previously trained via reinforcement learning for team $\pi_k^{-\alpha}$. Using the most likely team policy $\pi_k^{-\alpha}$, the ad hoc agent samples an action from the associated pre-trained policy π_k^α in order to obtain a best-response A_t^α. Whenever the ad hoc agent is interacting with a team, it also trains two new policies, π_{K+1}^α and $\pi_{K+1}^{-\alpha}$. π_{K+1}^α, trained via reinforcement learning using rewards R_{t+1}, is then stored in Π^α by the end of the interaction, while $\pi_{K+1}^{-\alpha}$, trained via supervised learning using the actions of the team $A_t^{-\alpha}$, is then stored in $\Pi^{-\alpha}$.

has no way of reusing previous task knowledge when interacting with a new team that it never encountered before. Consequently, a new policy π_{K+1}^α must be learned entirely from scratch, reducing ad hoc teamwork to a standard reinforcement learning problem. Tackling this limitation, Chen et al. [2020] introduce an attention-based algorithm for ad hoc teamwork which, when interacting with an unknown team, weighs in the likelihood of each possible known team in order to best model the current one (also relying on the states of the environment X_t and actions of the teammates $A_t^{-\alpha}$). In another line of work [Chen et al., 2019], the authors instead explore how the sub-tasks being performed by the teammates can be identified instead of their policies (alleviating the issue of dealing with unknown teammates). Their approach, ATSIS, identifies the sub-tasks by observing the actions of the teammates $A_t^{-\alpha}$ and partial observations of the environment Z_{t+1}. Later on, Shafipour Yourdshahi et al. [2020] expand upon Chen et al. [2019] and introduce an approach (OEATA) which keeps track of each sub-task being performed by the teammates.

The second limitation is that PLASTIC-Policy is that it assumes the environment to be fully observable (i.e., relying on a state X_t), the actions of the team $A_t^{-\alpha}$ to be visible to the ad hoc agent and an explicit reward signal R_{t+1} available which can be used to adapt, on-the-fly, to never before seen teams. These assumptions limits the application of PLASTIC-Policy to partially observable environments, such as those found in real-world scenarios (where the agent is deployed into robots which are only able to obtain partial observations via some sensors).

On parallel lines of research [Wang et al., 2024a] introduce the problem of N-Agent Ad Hoc Teamwork, which considers scenarios where not one but multiple ad hoc agents are deployed into a team and required to assist the remainder of the team on-the-fly. In their work, the authors formalise the setting and propose a novel approach called POAM (Policy Optimisation with Agent Modelling), framing the problem within the scope of multi-agent reinforcement learning. Similarly, Rahman et al. [2021, 2023] introduce the setting of open ad hoc teamwork, under the assumption that

teammates might join and leave the team dynamically, thus requiring the ad hoc agent to adapt to additional changes in the team. The authors present an approach that resorts to graph neural networks (GNNs) [Wu et al., 2020], and model the team as a graph. In their approach, the nodes in the graph represent the teammates, which may be added or removed dynamically. The authors show that their approach is able to dynamically adapt to changes in the team, as teammates come and go. Rahman et al. [2021, 2023] are then followed by Wang et al. [2024b], who employ a joint Q-learning from cooperative game theory for the problem of open ad hoc teamwork. In our work, we do not consider scenarios with open teams, leaving that possibility for future work. We assume that the ad hoc agent joins a fixed-size team of stationary agents. The authors are then followed by Rahman et al. [2024], Fang et al. [2024] and Hammond et al. [2024] deal with the problem of adapting to teammates when having limited prior team sets. Rahman et al. [2024] introduce an approach which builds a minimum coverage set of teammates which can be used to generate teammate models similar to newly encountered ones, while Fang et al. [2024] present a meta-learning approach to adapt a team model on-the-fly. Finally, Hammond et al. [2024] introduce the concept of symmetry breaking augmentations (SBA) which identifies team models never before encountered if their model/actions are symmetric to ones previously seen.

3.1.3 Related Areas of Research

In this thesis, we consider four different areas of research to be similar to the problem of ad hoc teamwork. These are (i) the problem of zero-shot coordination [Hu et al., 2020], the problem of (ii) few-shot teamwork [Fosong et al., 2022], (iii) Interactive POMDPs (IPOMDPs) [Gmytrasiewicz and Doshi, 2005] and (iv) multi-agent reinforcement learning (MARL) [Zhang et al., 2021]. Even though these areas of research share some assumptions with the setting of ad hoc teamwork, each and every single one of them has key characteristics which prevent them from being labelled as ad hoc teamwork.

Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) [Zhang et al., 2021], as the name suggests, extends the framework of RL to settings involving multiple learning agents. MARL and ad hoc teamwork have different research goals. While MARL studies how to compute policies for multi-agent settings, ad hoc teamwork studies how an ad hoc agent is able to adapt on-the-fly to potentially unexpected teams. Nevertheless, MARL techniques are often employed in ad hoc teamwork scenarios, for example, to compute a policy for handling a specific team when building the ad hoc agent's policy library. One example is the work of Albrecht and Ramamoorthy [2012], in which the authors showcase how five MARL algorithms can be applied to a diverse set of ad hoc problems.

Traditional (single-agent) reinforcement learning approaches can be used to tackle MARL in a decentralised way (e.g. DQN [Mnih et al., 2015]). In decentralised MARL, each agent can be independently trained in an environment, with the other agent's policies modelled as part of its dynamics [Tampuu et al., 2017]. In centralised learning approaches, on the other hand, other agents are explicitly modelled and taken into account during training. One example are Coordination Graphs [Guestrin et al., 2001], in where the optimal joint-action value function is computed for each agent which then communicates with the others by passing messages. In a more recent line of

work, Rashid et al. [2018] introduce QMix, an approach which estimates the action-values of each agent using neural networks and then resorts to a mixing network to combine them and estimating the optimal joint-action values.

Zero-Shot Coordination

From all related problems, the problem of *zero-shot coordination* [Hu et al., 2020, Lupu et al., 2021, Bullard et al., 2021, Treutlein et al., 2021], is the most similar to the problem of ad hoc teamwork. Introduced by Hu et al. [2020], the problem of zero-shot coordination studies how two independent agents, α_1 and α_2, may interact with one another on first-attempt, after being trained independently on a common task.

The setting assumes each agent α_i is trained for a two-agent problem M, where it individually obtained two policies, one for each agent role, $\pi^1_{\alpha_i}$ and $\pi^2_{\alpha_i}$, which when interacting with one another represent an optimal joint-policy $\pi^*_{\alpha_i}$. After this independent training stage, the goal is for agent α_1, without knowing the policies $\pi^1_{\alpha_2}, \pi^2_{\alpha_2}$ computed by α_2, to try to adapt as soon as possible in order to maximise a shared reward over an interaction.

There are, therefore, two key differences between ZSC and ad hoc teamwork: (i) in ZSC, an agent α_1 has to coordinate with a single teammate α_2, while in ad hoc teamwork, an ad hoc agent α has the goal of assisting a team of arbitrary length N and (ii) in ZSC, α_1 knows α_2 has computed an optimal joint-policy $\pi^1_{\alpha_2}, \pi^2_{\alpha_2}$ for the common task, while in ad hoc teamwork, the ad hoc agent α doesn't know anything about its teammates (teammates may even be legacy handcoded agents that don't even have an optimal policy).

Techniques from the setting of ZSC, may, nevertheless, be used in conjunction with techniques for ad hoc teamwork, for example after identifying an optimally trained teammate amongst the teammates and adapting to its behaviour as best as possible.

Few-Shot Teamwork

The problem of few-shot teamwork (FST), recently introduced by Fosong et al. [2022], showcases a similar problem to ad hoc teamwork, however, applied to teams of multiple agents. Instead of focusing on how a single ad hoc agent α may assist a team $-\alpha$ on-the-fly, in FST, two or more teams $-\alpha_1, \ldots, -\alpha_K$, each trained independently on their sub-tasks $M_1, \ldots, M_K$, must come together in order to perform a global shared task M comprised of the multiple sub-tasks. The authors assume the teams, after going through a source stage where they train via MARL how to perform their sub-tasks, are then given an adjustment stage, where they are able to interact with each other in performing the global task (hence the name "few-shot" instead of "zero-shot"). After the adjustment phase is complete, the combination of all teams $\mathrm{A} = \bigcup_{i=1}^{K} -\alpha_i$ is then evaluated on it $M = \bigcup_{i=1}^{K} M_i$ according to some metric.

Dealing with Humans as Teammates

In real-world domains teams may be comprised of both robots and human teammates. Carroll et al. [2019b] show that traditional RL agents (such as those trained via self-play [Silver et al., 2017], even though they are able to perform the task well when paired with instances of themselves, are unable to maintain their performance when paired with human teammates. In their work, the authors demonstrate that approaches relying on model-based planning or model-free algorithms, when trained

using learned human models, are able to significantly improve the performance when paired with human teammates, outperforming agents trained via self-play. Strouse et al. [2021] later introduce a method called Fictitious Co-Play (FCP), capable of achieving higher performances than the baseline in benchmark multi-agent tasks. On a similar line of work, Wang et al. [2020] approach the problem of cooperating with human teammates when there are sub-tasks to be performed in the environment. The authors introduce an approach called Bayesian delegation, which infers the intentions of the teammates via inverse planning, using theory-of-mind. Siu et al. [2021] evaluate how humans, blindly paired with agents as teammates, perform in the game and how well they perceive their teammates in terms of performance, teamwork, interpretability and trust.

In ad hoc teamwork, current approaches such as PLASTIC-Policy [Barrett et al., 2017] consider settings where the teammates' actions are visible to the ad hoc agent. When dealing with human teammates or even other robots in real-world scenarios, observing the actions of the teammates is seldom possible. In this thesis we mitigate this problem introducing the HOTSPOT framework [Ribeiro et al., 2024a], a framework which allows for the deployment and evaluation of ad hoc agents into real-world scenarios involving human-robot collaboration.

Dealing with Partial Observability & Interactive POMDPs

When dealing with scenarios where the agent is given a partial view of the state of the environment, such observations are not enough to predict the next observation or reward. Traditional approaches, such as partially observable Markov Decision Processes (POMDPs) model not only the environments dynamics and agents' observation functions, but also a belief over each possible state. POMDPs can be solved via either exact methods [Cassandra et al., 2013, Littman, 1994] or point-based methods [Spaan and Vlassis, 2005].

Even though these approaches may be used when dealing with multi-agent environments in a decentralised way, a more common approach is to model and solving the environment as a Decentralised POMDP (or Dec-POMDP) [Bernstein et al., 2002]. Dec-POMDPs expand upon multi-agent MDPs (MMDPs) by incorporating the probabilities of each agent making each observation. However, given that Dec-POMDPs are NEXP-Complete [Bernstein et al., 2002], available solvers [Szer and Charpillet, 2006, Oliehoek et al., 2010, Spaan et al., 2011] do not scale to larger problems. According to Kraemer and Banerjee [2016], even though more scalable approaches have been proposed for solving DEC-POMDPs [Seuken and Zilberstein, 2007, Amato et al., 2009, Dibangoye et al., 2009], these approaches rely not only on the assumption that the model is known a priori but the agents are able to plan in a centralised way. In their work, Kraemer and Banerjee [2016] introduce an approach named reinforcement learning as a rehearsal (RLaR), which assumes the agents are, during a training phase, able observe the state of the environment and the actions of the teammates. In a similar line of work, Omidshafiei et al. [2017] introduce the multi-task multi-agent reinforcement learning (MT-MARL) algorithm. MT-MARL assumes there are multiple tasks to be performed and model each one of them as a Dec-POMDP. The authors then follow Hausknecht and Stone [2015] by using Deep Recurrent Q-Networks in order to learn how to perform each DEC-POMDP. Later on, Foerster et al. [2018] expand upon the work of Lowe et al. [2017] by introducing another multi-agent actor-critic method called counterfactual multi-agent (COMA). Their approach trains a centralised critic network (which estimates the optimal action-values for all agents) and then trains decentralised actors in order to obtain each agent's policy. The authors

are then followed by Chen [2019] which introduce a more sample efficient algorithm named Centralised Training and Exploration with Decentralised execution via policy Distillation (CTEDD). Finally, Lyu et al. [2021] introduce a more analytical where in where they analyse the properties of centralised training with decentralised execution approaches. In their work [Lyu et al., 2021], the authors claim that a centralised critic, contrary to previous claims, does not make the learning process more stable.

Finally, Interactive POMDPs (IPOMDPs), introduced by Gmytrasiewicz and Doshi [2005], are approaches which modify POMDPs in order to incorporate the behaviour of possible teammates. The authors [Gmytrasiewicz and Doshi, 2005] consider agent α may interact with a team $-\alpha$ in a partially observable environment modelled as a POMDP $\hat{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, P, O, r, \gamma)$. The authors incorporate the team's behaviour as part of the states, augmenting the state space $\mathcal{X}$ into an interactive state space $\mathcal{IX} = \mathcal{X} \times \mathrm{A}$ (by taking into account N possible team combinations $\mathrm{A} = \{-\alpha_1, -\alpha_2, \ldots, -\alpha_N\}$). The state space is therefore grown exponentially with the number of possible teams N. The resulting IPOMDP can then be solved using regular POMDP solvers (refer to section 2.1.5 of chapter 2). Agent α is also able to keep track of a belief b over the possible augmented states $ix \in \mathcal{IX}$.

Even though IPOMDPs share two common assumptions with our work: (i) the agent α must adapt to an unknown team and (ii) agent α only has access to a partial observation of the environment, IPOMDPs have a key assumption which differentiates them from ad hoc teamwork — IPOMDPs assume all agents $\{\alpha, -\alpha\}$ share the same reward function r (also known beforehand to the agent α), while in ad hoc teamwork, on the other hand, while still assumed the team shares a common goal, teammates may have different reward functions, given that they may be performing sub-tasks [Chen et al., 2019].

3.2 Discussion & Framing of this Thesis

In all existing literature, all approaches either assume the ad hoc agent α is able to observe the state of the environment X_t, the actions of the teammates $A_t^{-\alpha}$ in order to adapt, on-the-fly, to current teammates $-\alpha$. As previously mentioned, these assumptions prevent these approaches from being directly deployed into real-world scenarios which may not only involve robots with imperfect sensors, but also non-optimal teammates such as humans, which do not explicitly communicate their actions $A_t^{-\alpha}$ to the ad hoc agent deployed into a robot.

This thesis fills the gap in the literature by dropping these assumptions and asking and answering the following research question:

> *How can an autonomous agent assist unknown teammates in performing unknown tasks under more realistic conditions, without being able to pre-coordinate or pre-communication with the existing team?*

In our work, we break down the above research question into four sub-problems, each individually addressed in their specific chapter.

- Chapter 4: How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?

- Chapter 5: How can an autonomous agent assist unknown teammates in performing unknown tasks without being able to observe their actions, the full state of the environment or both?
- Chapter 6: How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?
- Chapter 7: How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?

To conclude, we lay a road map for this thesis, targeting each sub-problem sequentially. We start from the current state-of-the-art in ad hoc teamwork (PLASTIC-Policy [Barrett et al., 2017]), answering the first research sub-problem in chapter 4 by introducing and evaluating a novel model-based approach for ad hoc teamwork which more efficiently adapts to different teams and tasks on-the-fly. We then answer the second research sub-problem in chapter 5. We start by dropping the assumptions that the ad hoc agent α is able to observe the actions of the team, $A_t^{-\alpha}$, introducing and evaluating a novel Bayesian online approach which infers the correct team policy $\pi^{-\alpha}$ and task M being performed from observing state transitions alone. In the same chapter, we then further drop the assumption that the state is fully observable, introducing and evaluating a novel Bayesian online approach which infers the correct team policy $\pi^{-\alpha}$ and task M being performed, relying only on partial observations Z_{t+1}. Finally, in chapter 6, we answer the third research sub-problem we extend our approaches to arbitrarily large, partially observable domains, introducing and evaluating a novel approach, with two variants, capable of performing ad hoc teamwork in these conditions. To conclude, figure 3.6 showcases a diagram of the road map for the thesis.

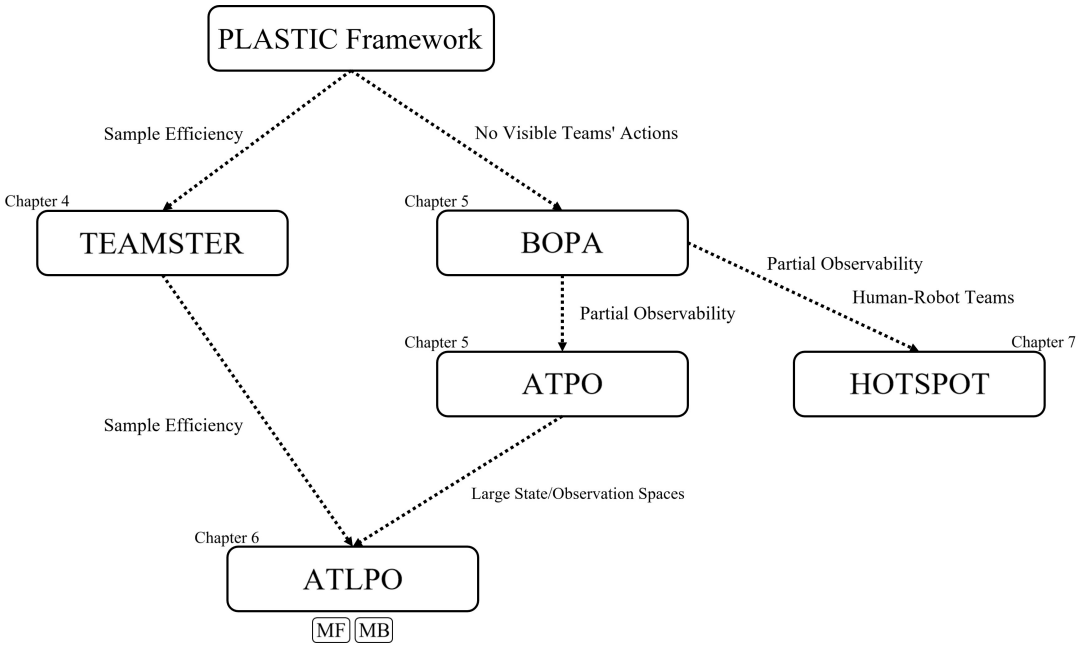


FIGURE 3.6: Framing for this thesis. In order to answer the research question of *how can an autonomous agent assist unknown teammates in performing unknown tasks under more realistic conditions, without being able to pre-coordinate or pre-communication with the existing team?*, we break down our thesis in four different sub-problems. We start in chapter 4 by introducing TEAMSTER, a model-based approach for ad hoc teamwork capable of being more sample efficient than PLASTIC-Policy Barrett et al. [2017]. We then drop the assumptions that the ad hoc agent has access to the actions of the teammates, introducing BOPA, an approach capable of performing ad hoc teamwork relying only on the state of the environment, and ATPO, an approach capable of performing ad hoc teamwork relying only on partial observations of the environment (both in chapter 5). We then expand ATPO for arbitrarily large, partially observable domains, introducing ATLPO, an approach for ad hoc teamwork which employs state-of-the-art Deep Learning techniques to identify the most likely team and task, as well as compute a best-response. We present two variants of ATLPO, ATLPO-MF, which sets up its policy library using state-of-the-art techniques in Model-Free reinforcement learning, and ATLPO-MB, which sets up its policy library using state-of-the-art techniques in Model-Based reinforcement learning. Finally, in chapter 7, we introduce and evaluate a framework for the deployment of ad hoc approaches to real-world scenarios where mixed teams of humans and other robots operate.

Chapter 4

Model-Based Reinforcement Learning for Ad Hoc Teamwork

This chapter addresses our first research sub-problem:

> *How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?*

As discussed on chapter 3, Barrett et al. [2017] introduced two algorithms for ad hoc teamwork, PLASTIC Model Barrett [2014] and PLASTIC Policy Barrett et al. [2017] which assume the ad hoc agent is able to observe the state of the environment, the actions of the teammates and may use a reward signal provided by the environment in order to learn via reinforcement learning. PLASTIC Model also assumes the task's dynamics are fully known beforehand while PLASTIC Policy, on the other hand, relies on a model-free reinforcement learning algorithm to learn a policy for each team within its library. When faced with a new team, both agents then match the teammates' observed behaviour to previously known teams and uses the policy to play against that team.

While on one hand these algorithms provide state-of-the-art performance in ad hoc teamwork, each algorithm suffers from a distinct limitation. PLASTIC Model assumes the task's dynamics are fully known *a priori*. This assumption not only requires a significant amount of expert knowledge in the task's domain but also prevents the ad hoc agent from adapting to slightly different variations of the task. On the other hand, PLASTIC Policy implicitly learns the task dynamics when learning the policy during an interaction with a team. Their model-free design and the assumption that the environment's dynamics and goal never change, however, prevent them from better adapting to new teams never before encountered before. Unfortunately, using a model-free approach, its prior knowledge of team and environment is combined. Therefore, if the agent learned how to play team A in environment X and team B in environment Y and encounters team A in environment Y, it will have to learn everything from scratch, which does not happen with PLASTIC Model.

This chapter addresses these limitations and our first research sub-problem — *how can an autonomous agent more efficiently adapt to different teams and tasks on-the-fly?* — and proposing a novel model-based algorithm for ad hoc teamwork, called TEAMSTER [1], which seeks to capture the advantages of both PLASTIC methods while addressing the corresponding limitations. Much like PLASTIC Model, TEAMSTER considers previous knowledge about the environment and teammate behaviour separately, readily overcoming PLASTIC Policy's inability to do so. At the same time,

[1] TEAMSTER stands for **T**ask and t**EAM**mate identification through model-ba**S**ed reinforcemen**T** l**EaR**ning

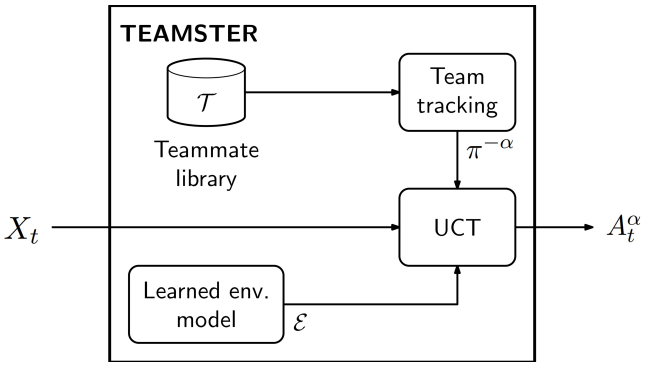


FIGURE 4.1: TEAMSTER ad hoc agent's architecture.

much like PLASTIC Policy, our approach does not rely on a hand-coded model of the environment, enabling the agent to adapt to variations of the task more efficiently.

We expand upon the PLASTIC-Policy under the same set of assumptions: (i) the agent is allowed to use prior knowledge in order to adapt to a new team, (ii) the agent is unable to pre-coordinate or communicate with the team and (iii) the agent is able to observe the state of the environment, the actions of the team and the reward signal from the environment.

We illustrate our proposed algorithm's benefits by performing an empirical evaluation in the Pursuit environment, measuring its performance and comparing with both PLASTIC Policy and PLASTIC Model. Our results confirm that TEAMSTER exhibits several important advantages, such as improved sample efficiency, robustness to changes in the environment and flexibility to accommodate inaccurate environment models. As such, it offers an appealing alternative in real-world settings, where accurate models are seldom available and the agent is required to identify the current task and the existing agents' behaviour.

TEAMSTER is an algorithm for ad hoc teamwork that combines the advantages of PLASTIC-Model and PLASTIC-Policy while circumventing the disadvantages of both. We follow Melo and Sardinha [Melo and Sardinha, 2016], breaking down the problem of ad hoc teamwork into the three sub-problems: (i) teammate identification; (ii) task identification; and (iii) planning and execution.

Figure 4.1 depicts TEAMSTER's architecture, with its three core modules. The first module is the *teammate library and team tracking* module, used for teammate identification. We discuss this model in detail in section 4.0.1. The second module is the *learned environment model*, used for task identification. We discuss this model in detail in section 4.0.2. Finally, the last module consists of a standard online planner, which receives information from the other two modules and, given the current state X_t, computes the agent's action, A_t^α. We discuss this model in detail in section 4.0.3.

We also summarise TEAMSTER's main loop in Algorithm 1, to further clarify the interaction between the different elements in Fig. 4.1. TEAMSTER requires a library of teammates, $\mathcal{T}$, and an environment model, $\mathcal{E}$.[2] When interacting with a team, TEAMSTER does not know whether such team is one in $\mathcal{T}$ or a completely new team, never encountered before. As such, TEAMSTER initialises a new team model, $\hat{\pi}^{-\alpha}$ (line 2), and sets its belief regarding the current team to a uniform distribution over the set $\hat{\mathcal{T}} = \mathcal{T} \cup \{\hat{\pi}^{-\alpha}\}$ (lines 3-4). The new teammate model, $\hat{\pi}^{-\alpha}$, will be updated as TEAMSTER interacts with the current team and, eventually, added to the library $\mathcal{T}$.

[2]The first time TEAMSTER runs, $\mathcal{T}$ is empty and $\mathcal{E}$ is just a "blank slate" model. In that case, TEAMSTER reduces to a model-based RL algorithm.

Algorithm 1 TEAMSTER agent pseudo-code description.

Require: Teammate library, $\mathcal{T}$
Require: Environment model, $\mathcal{E}$
1: $t = 0$
2: initialise team model $\hat{\pi}^{-\alpha}$
3: Set $\hat{\mathcal{T}} = \mathcal{T} \cup \{\hat{\pi}^{-\alpha}\}$
4: initialise distribution p_0
5: **repeat**
6: Observe current state X_t
7: $\pi_t^{-\alpha} = \text{argmax}_{\pi^{-\alpha} \in \hat{\mathcal{T}}}\, p_t(\pi^{-\alpha})$
8: A_t^{α} = Planner(X_t,$\pi_t^{-\alpha}$,$\mathcal{E}$)
9: Execute A_t^{α} and observe $\boldsymbol{A}_t^{-\alpha}$, R_{t+1}, X_{t+1}
10: p_{t+1} = UPDATETEAMBELIEFS(p_t, $\hat{\mathcal{T}}$, X_t, $\boldsymbol{A}_t^{-\alpha}$)
11: $\mathcal{E}$ = UPDATEENVIRONMENTMODEL($\mathcal{E}$, X_t, $\boldsymbol{A}_t$, R_{t+1}, X_{t+1})
12: $\hat{\pi}^{-\alpha}$ = UPDATEONLINETEAMMODEL($\hat{\pi}^{-\alpha}$, X_t, $\boldsymbol{A}_t^{-\alpha}$)
13: $t = t + 1$
14: **until** X_t **is terminal**

The interaction between TEAMSTER and the teammates corresponds to the main cycle in Algorithm 1. At every step t, the agent observes the current state of the environment, X_t (line 6). It then selects the team model that best describes the current team, $\pi_t^{-\alpha}$, according to the agent's belief p_t (line 7). Even though we consider beliefs to be uniformly initialised at the start of new interactions (i.e. $p_0 = \text{Uniform}(\hat{\mathcal{T}})$), any arbitrary distribution may be used as initial prior. TEAMSTER then uses the planner to determine its action, A_t^{α}, given the current state, X_t, the environment model, $\mathcal{E}$, and the current team, $\pi_t^{-\alpha}$ (line 8).[3]

Once the action is selected, TEAMSTER proceeds as standard model-based RL: it executes the action A_t^{α}, observes the teammate actions, $\boldsymbol{A}_t^{-\alpha}$, the resulting state X_{t+1} and reward, R_{t+1}, and uses these to update the belief p_t over teams, the environment model $\mathcal{E}$, and the team model $\hat{\pi}^{-\alpha}$ (lines 9-12).

In the continuation, we provide detailed descriptions of each of the modules in TEAMSTER.

4.0.1 Teammate identification and team tracking

The purpose of the team tracking module is to determine whether the current team corresponds to any of the teams in the library $\mathcal{T}$. As depicted in Fig. 4.2, each team model in $\mathcal{T}$ is nothing but a reduced joint policy: given as input a state $x \in \mathcal{X}$, the team model returns a predicted reduced joint action $\boldsymbol{A}^{-\alpha}$ at state x.

We use parametrised team models in order to represent the teammates' policies. In particular, we assume that the parametrised model for team k has a set of parameters θ_k that are learned/optimised as the ad hoc agent interacts with such team. In other words, the interaction with a team k generates a dataset $\left\{(X_t, \boldsymbol{A}_t^{-\alpha}), t = 0, \ldots, T\right\}$ that TEAMSTER uses to learn the parameters θ_k for the reduced policy $\pi_k^{-\alpha}$ describing team k. We write $\pi_k^{-\alpha}(\boldsymbol{A}^{-\alpha} \mid x; \theta_k)$ to denote the probability of action $\boldsymbol{A}^{-\alpha}$ in state x

[3]While we could use the complete distribution p_t in the planner to more accurately describe the agent's belief regarding the teammates, such option requires the planner to sample significantly larger trees. In our experiments, the gains in performance are negligible compared with the computational cost, hence the formulation adopted.

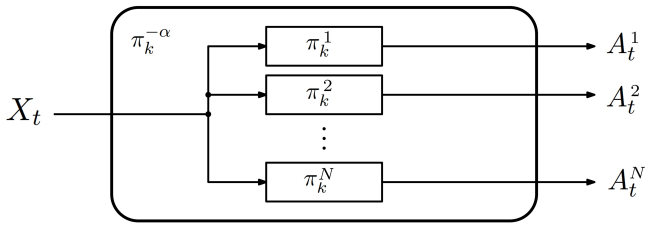


FIGURE 4.2: The team model. A parametrised model which can be optimised to accurately predict the teammates' policies

according to $\pi_k^{-\alpha}$, when parametrised by θ_k, i.e.,

$$\pi_k^{-\alpha}(\boldsymbol{A}^{-\alpha} \mid x; \theta_k) = \mathbb{P}_{\theta_k}\left[\mathbf{A}_t^{-\alpha} = \boldsymbol{A}^{-\alpha} \mid \mathrm{X}_t = x\right],$$

for $x \in \mathcal{X}$ and $\boldsymbol{A}^{-\alpha} \in \mathcal{A}^{-\alpha}$. Since this work only considers finite action settings, $\mathcal{A}^{-\alpha}$ is a finite set and, for each $x \in \mathcal{X}$, $\pi_k^{-\alpha}(\cdot \mid x; \theta_k)$ is a categorical distribution parametrised by θ_k. Such distribution can take the general form

$$\pi_k^{-\alpha}(\boldsymbol{A}^{-\alpha} \mid x; \theta_k) = \frac{\exp\left\{f(x, \boldsymbol{A}^{-\alpha}; \theta_k)\right\}}{\sum_{\boldsymbol{A}^{-\alpha'}} \exp\left\{f(x, \boldsymbol{A}^{-\alpha'}; \theta_k)\right\}},$$

where f is a real-valued function defined over $\mathcal{X} \times \mathcal{A}^{-\alpha}$ and parametrised by θ_k. Our model is agnostic to the particular choice of f, which will be problem-dependent. Concretely, in our experiments, f is represented through a neural network.

Then, given the dataset $\left\{(X_t, \boldsymbol{A}_t^{-\alpha}, t = 0, \ldots, T\right\}$, the parameters θ_k are adjusted to maximise the likelihood of the data. In other words, defining the loss

$$L(\theta_k) = -\sum_{t=0}^{T} \log \pi^{-\alpha}(\boldsymbol{A}_t^{-\alpha} \mid X_t; \theta_k), \tag{4.1}$$

the parameters θ_k can be computed analytically, in simple settings, or using gradient descent, in more complex settings, such as those considered in our experiments.

Team tracking at runtime

TEAMSTER uses the same team model selection approach as the PLASTIC algorithms [Barrett et al., 2017], enabling the ad hoc agent to identify the most likely team at runtime. The model selection approach departs from the library of learned team models, $\mathcal{T} = \left\{\pi^{-\alpha^1}, \ldots, \pi^{-\alpha^K}\right\}$. As discussed above, each model $\pi^{-\alpha^k} \in \mathcal{T}$ has been previously learned, when the ad hoc agent interacted with team k. The agent then maintains a probability distribution p_t over the augmented library $\hat{\mathcal{T}} = \mathcal{T} \cup \{\hat{\pi}^{-\alpha}\}$, where $\hat{\pi}^{-\alpha}$ is the model being learned for the current team. The distribution p_t is initialised as the uniform distribution and updated at each time step according to the ability of the different models to predict the observed actions of the teammates.

To describe the update process for p_t, we first observe that, given a team model $\pi^{-\alpha} \in \hat{\mathcal{T}}$, the probability of executing an action $\boldsymbol{a}^{-\alpha} = (a^1, \ldots, a^N)$ at state is x can be computed as

$$\pi^{-\alpha}(\boldsymbol{a}^{-\alpha} \mid x) \triangleq \mathbb{P}_{\pi}\left[\mathbf{A}_t^{-\alpha} = \boldsymbol{a}^{-\alpha} \mid \mathrm{X}_t = x\right] = \prod_{n=1}^{N} \pi^n(a^n \mid x), \tag{4.2}$$

where π^n represents the individual policy for each agent $n \in \{1, \ldots, N\}$. Since all $N+1$ agents act simultaneously, without knowing in advance which actions the rest of the team will execute, their actions are independent. The probability $\pi^{-\alpha}(\boldsymbol{A}^{-\alpha} \mid x)$ can therefore be obtained by multiplying all the individual probabilities $\pi^n, n = 1, \ldots, N$, as shown in (4.2).

Then, at each time step t, given the state X_t and the teammates' actions $A_t^{-\alpha}$, the loss for each team model $\pi^{-\alpha} \in \hat{\mathcal{T}}$ is given by

$$loss(\pi^{-\alpha}) = 1 - \pi^{-\alpha}(\boldsymbol{A}_t^{-\alpha} \mid X_t),$$

and we set

$$p_{t+1}(\pi^{-\alpha}) = \frac{1}{\rho} p_t(\pi^{-\alpha})(1 - \eta\, loss(\pi^{-\alpha})),$$

where η is a hyper-parameter that bounds the maximum allowed loss, and ρ is a normalisation constant given by:

$$\rho = \sum_{\pi^{-\alpha} \in \hat{\mathcal{T}}} p_t(\pi^{-\alpha})(1 - \eta\, loss(\pi^{-\alpha})).$$

Algorithm 2 summarises the function that updates the probability distribution p_t over $\hat{\mathcal{T}}$, called in line 10 of Algorithm 1.

Algorithm 2 Function to updates the team's beliefs at runtime

```
1: function UpdateTeamBeliefs(p, T, x, A^{-α})
2:     for π^{-α} ∈ T do
3:         Compute π^{-α}(A^{-α} | x) = ∏_{n=1}^{N} π^n(a^n | x)
4:         Compute loss(π^{-α}) = 1 − π^{-α}(A^{-α} | x)
5:         Compute p_new(π^{-α}) = (1/ρ) p(π^{-α})(1 − η loss(π^{-α}))
    return p_new
```

4.0.2 Task identification with the learned environment model

Simultaneously, the agent keeps a model $\mathcal{E}$ of the environment, also learned as the agent interacts with previous teams. We consider, in our discussion, the same environment for all interaction with the previous teams. However, the proposed approach also admits keeping a library of environment models that the agent can select in the same way as the teammates.

The environment model

TEAMSTER, in its interactions with different teams, incrementally learns an environment model, $\mathcal{E}$, comprising estimates $\hat{\mathbf{P}}$ and $\hat{r}$ for the MMDP parameters $\mathbf{P}$ and r (see Fig. 4.3). The environment model is essential for the planner, because it predicts the next state, X_{t+1}, and reward, R_{t+1}, given the current state, X_t, and the team's joint action, $\mathbf{A}_t$.

Much like the team models, we also consider an environment model $\mathcal{E}$ parametrised by a set of parameters θ_E that are learned/optimised using transitions $(x, \boldsymbol{a}, r, x')$ observed by the agent while interacting with the environment. Specifically, we consider that $\theta_E = (\theta_P, \theta_r)$, where θ_P are the parameters describing the transition probabilities $\hat{\mathbf{P}}$ and θ_r are the parameters representing the reward function $\hat{r}$. Then, given a state $x \in \mathcal{X}$ and a joint action $\boldsymbol{a} \in \mathcal{A}$, $\hat{\mathbf{P}}(x, \boldsymbol{a}; \theta_P)$ denotes the distribution over $\mathcal{X}$ prescribed

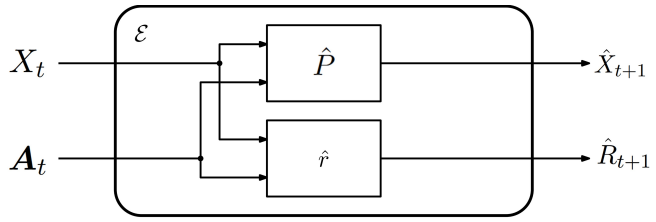


FIGURE 4.3: The environment model $\mathcal{E}$. We assume that $\mathcal{E}$ contains a parametrised representation for the MMDP transition probabilities (denoted as $\hat{\mathbf{P}}$) and reward (denoted as $\hat{r}$). These representations are then optimised to accurately predict the transitions observed from the environment.

by the estimated transition model with parameters θ_P, and $\hat{r}(x, \boldsymbol{a}; \theta_r)$ describes the reward prescribed by the estimated reward function with parameters θ_r.

Recalling that the transition probabilities $\mathbf{P}$ consist of distributions over (next) states conditioned on the current state and action, i.e.,

$$\mathbf{P}(x' \mid x, \boldsymbol{a}) = \mathbb{P}\left[\mathrm{X}_{t+1} = x' \mid \mathrm{X}_t = x, \mathbf{A}_t = \boldsymbol{a}\right],$$

our model, $\hat{\mathbf{P}}$, encodes these transition probabilities using some parametrised family of distributions. For continuous action spaces, a typical parameterisation is to consider that the distribution over possible next states is a Gaussian distribution with mean $\mu(x, \boldsymbol{a})$ and variance $\sigma^2(x, \boldsymbol{a})$. Formally,

$$\hat{\mathbf{P}}(x' \mid x, \boldsymbol{a}; \theta_P) = \mathbf{Normal}(x'; \mu(x, \boldsymbol{a}; \theta_P), \sigma^2(x, \boldsymbol{a}; \theta_P)),$$

where μ and σ are functions defined over $\mathcal{X} \times \mathcal{A}$ and parametrised by θ_P. In more complex settings, we may instead consider a *mixture of Gaussians*, where

$$\hat{\mathbf{P}}(x' \mid x, \boldsymbol{a}; \theta_P) = \sum_{m=1}^{M} \rho_m(x, \boldsymbol{a}; \theta_P) \mathbf{Normal}(x'; \mu_m(x, \boldsymbol{a}; \theta_P), \sigma_m^2(x, \boldsymbol{a}; \theta_P)),$$

where $\rho_m, m = 1, \ldots, M$, are the parameters of the mixture.

In the case of our environments, since $\mathcal{X}$ is a discrete space, the transition probabilities are, once again, given by a categorical distribution over $\mathcal{X}$, i.e.,

$$\hat{\mathbf{P}}(x' \mid x, \boldsymbol{a}) = \frac{\exp\{g(x, \boldsymbol{a}, x'; \theta_P)\}}{\sum_{x''} \exp\{g(x, \boldsymbol{a}, x''; \theta_P)\}},$$

for some real-valued function g defined over $\mathcal{X} \times \mathcal{A} \times \mathcal{X}$ and parametrised by θ_P. g corresponds to a neural network in our experiments. Then, given a dataset of transitions, $\{(x_n, \boldsymbol{a}_n, x'_n), n = 1, \ldots, N\}$, the parameters θ_P are computed to maximise the likelihood of the data, i.e., minimise the loss

$$L(\theta_P) = -\sum_{n=1}^{N} \log \hat{\mathbf{P}}(x'_n \mid x_n, \boldsymbol{a}_n; \theta_P),$$

which can be done using, for example, stochastic gradient descent.

4.0.3 Planning and execution

At each time step t, the agent uses the current probability p_t to select a teammate model $\pi_t^{-\alpha}$ (line 7 of Algorithm 1). With the teammate model $\pi_t^{-\alpha}$ and the environment mode $\mathcal{E}$, the ad hoc agent can now select its next action A_t^{α} using a planning algorithm (line 8 of Algorithm 1).

TEAMSTER, concretely, uses an online planning algorithm (UCT [Kocsis and Szepesvári, 2006]) to plan the action A_t^{α} as a function of the current state of the environment X_t. However, any arbitrary planners, as long as they take as input the current state X_t, the environment model $\mathcal{E}$ and teammate model $\pi^{-\alpha}$ and are able to compute the best action for the ad hoc agent, A_t^{α}, can be used as this component. Offline planners such as value iteration [Littman, 2001], which compute the optimal policy for the ad hoc agent using dynamic programming, can also be used. However, they require the domains to have both discrete and relatively small state spaces. Online planners (such as UCT) are an efficient way of handling large state and action spaces, thus being suitable for complex domains. The UCT algorithm, used in our experiments, builds a search tree using rollouts to estimate the Q-values of the different actions in the current state x. The tree is expanded using the UCB heuristic, designed to efficiently select the most promising nodes to be expanded. To perform such rollouts, UCT requires the ability to simulate the environment—i.e., the environment model $\mathcal{E}$—and the actions of the teammates—i.e., the teammate model $\pi^{-\alpha}$.

After executing the action A_t^{α} recommended by the planner, TEAMSTER observes the actions executed by the teammates, $\boldsymbol{A}_t^{-\alpha}$, the resulting environment state, X_{t+1}, and corresponding reward, R_{t+1} (line 9 of Algorithm 1). The resulting transition,

$$\tau(t) = \left(X_t, (A_t^{\alpha}, \boldsymbol{A}_t^{-\alpha}), R_{t+1}, X_{t+1}\right),$$

can then be used to update the model $\mathcal{E}$ of the environment (line 11 of Algorithm 1) and the current team model $\hat{\pi}^{-\alpha}$ (line 12 of Algorithm 1).

4.1 Evaluation

This section details the experimental setup (domains and teams) (section 4.1.1), the evaluation methodology (section 4.1.4) and the obtained results (section 4.1.6).

4.1.1 Environments and Available Teams

In our experiments, we consider four different benchmarks from the multi-agent systems literature: (i) the Pursuit domain [Barrett, 2014], the NTU domain [Melo et al., 2019, Hu et al., 2015], the ISR domain [Melo et al., 2019, Hu et al., 2015] and the Pentagon domain [Melo et al., 2019, Hu et al., 2015].

Pursuit Our first testbed is the Pursuit domain. In this domain, four predator agents must cooperate to capture a prey. Predators and prey move in a toroidal environment of $M \times M$ cells. For instance, Fig. 4.4(a) shows an example of a 5×5 world. An episode starts with a random configuration, i.e., the four predators and prey start in a random position within the environment. The episode ends when a *capture* occurs, i.e., the four predators surround the prey. Figure 4.4(b) shows an example of a capture configuration.

We model the state as a tuple $(s_0, s_1, s_2, s_3, s_4)$, where s_0 is the position of the ad hoc agent (one of the ghosts in Fig. 4.4(a)), s_1 to s_3 represent the positions of the teammates, and s_4 is the position of the prey (Pacman in Fig. 4.4(a)). In a $M \times M$ grid, each $s_n, n = 0, \ldots, 4$, takes values in the set $\{1, \ldots, M^2\}$.

At each step t, the prey tries to move (randomly) to one of the adjacent cells. The predator agents have available 4 possible actions—move up, down, left and right—each of which moves the agent to the adjacent cell in that direction, if unobstructed. If any two agents (predator or prey) try to move to the same cell, only one of the two succeeds (selected randomly), and the position of the other remains unchanged. We refer to this situation as a *collision*.

We can describe this scenario as an MMDP because the positions of the predators and prey, at time step $t + 1$, depend only on their corresponding positions and actions at time step t.

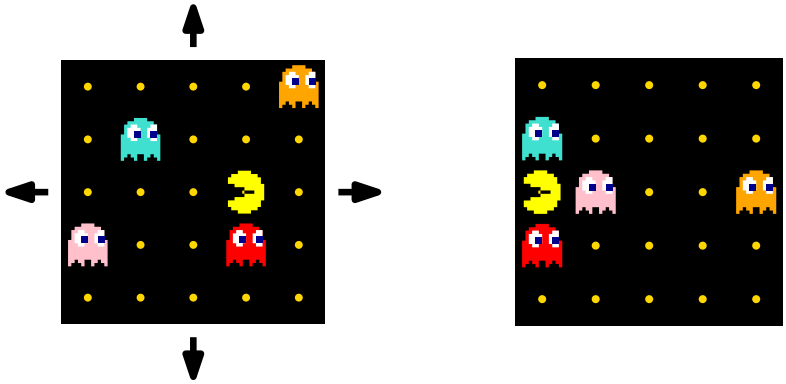
We consider three different teams in the Pursuit domain, each containing $N = 4$ agents:

Teammate Aware Team (τ_1): Agents that run an A* search to find the shortest path to the prey, treating teammates as immovable obstacles;

Greedy Team (τ_2): Agents that move to minimise their individual distance towards the prey, not taking into account collisions with other teammates;

Probabilistic Destinations Team (τ_3): Agents that favour positions closer to the prey while also flavouring positions that allow for circling around the prey, taking the teammates' locations into account.

NTU, Pentagon and ISR The NTU, Pentagon, and ISR domains [Melo and Veloso, 2009, Hu et al., 2015] in Fig. 4.5 model navigation tasks where N agents are spawned in random locations of a facility and are given the goal of exiting it as soon as possible. Teams in NTU contain $N = 4$ agents and in Pentagon and ISR domain contain $N = 3$ agents. All three domains abide by the same base rules, yet have different team sizes and map layouts, some being smaller and easier to navigate and others being larger and more complex to navigate. They have different state spaces, but the states are



(a) (b)

FIGURE 4.4: Pursuit domain. (a) The world is a 5×5 toroidal world—i.e., upon exiting one side of the environment, an agent immediately re-enters in the opposite side. Four predators (represented as four ghosts) must capture a prey (here represented as Pacman). (b) Example of a capture configuration. Since the world is toroidal, the orange ghost position is "equivalent" to standing on the left side of Pacman.

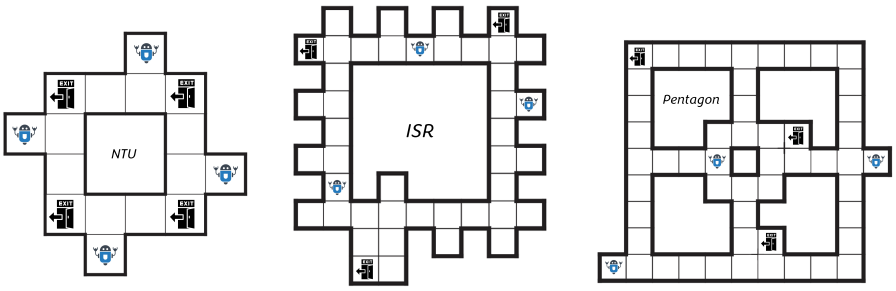


FIGURE 4.5: CMU-Gridworld domains [Melo and Veloso, 2009, Hu et al., 2015] (NTU, ISR and Pentagon). Agents must navigate in a grid world in order to each reach an exit while taking into account collisions.

modelled in the same way. In addition, each domain has a set of exit locations that may spawn in different cells at the beginning of an episode. An episode finishes when all agents have exited the facility.

We model the state as a tuple $(s_0, s_1, ..., s_N)$, where s_0 is the position of the ad hoc agent and $s_i, i \in \{1, ..., N\}$ represents the position of teammate i.

Similar to the Pursuit domain, NTU, ISR and Pentagon also come with three teams:

Altruistic Team (τ_1): Each agent computes the distances of all agents to all exits and selects the combination with the smallest total distance, independently of their closest goal. They then move towards their assigned exit from the selected combination.

Selfish Team (τ_2): Each agent moves towards its closest available exit, not taking into account whether some teammate is closer to that exit.

Weighted-Altruistic Team (τ_3): Similar to the Altruistic team, agents compute the distances to each exit for all teammates, but instead of selecting the combination with the lowest total distance, they randomly select one of the combinations using as its probability the corresponding distance. In other words, the smaller the total distance to an exit combination, the higher the probability of it being chosen.

4.1.2 Baselines

Alongside the original teammates detailed above in section 4.1.1, we evaluate the following six agents in the role of the ad hoc agent, adding to a total of seven baselines:

TEAMSTER (UCT + learned $\mathcal{E}$ + learned $\mathcal{T}$). Our approach.

PLASTIC-Model (UCT + perfect $\mathcal{E}$ + perfect $\mathcal{T}$). PLASTIC-Model has an architecture similar to TEAMSTER. However, there is one key difference: the environment model $\mathcal{E}$ and initial team library $\mathcal{T}$ are exact (provided by the agent designer) and not learned. Like TEAMSTER and PLASTIC-Policy, it also contains an extra team model, which is learned on the fly, enabling the PLASTIC-Model to adapt to new teams. However, unlike TEAMSTER and PLASTIC-Policy, its environment model $\mathcal{E}$ is fixed, i.e., it does not adapt to environment changes (e.g., different transition probabilities). Thus, with a fixed environment, the PLASTIC-Model's performance can be considered an upper bound to TEAMSTER because the latter learns the environment model from experience. It is

also worth stating that even though we consider PLASTIC-Model to represent the best possible behaviour in our domains, the UCT planner is nevertheless approximate, required due to the domains' relatively large state spaces. In our scenarios, methods such as value iteration [Littman, 2001] can be used to plan the optimal actions [Barrett et al., 2011, 2017, Barrett, 2014].

Learns Team (UCT + perfect $\mathcal{E}$ + learned $\mathcal{T}$): Learns Team has an architecture similar to TEAMSTER and PLASTIC-Model. It utilises an UCT planner combined with both an environment model $\mathcal{E}$ and initial team library $\mathcal{T}$. Its environment model $\mathcal{E}$ is exact, like PLASTIC-Model, but the team library $\mathcal{T}$ learned like TEAMSTER and PLASTIC-Policy. Thus, with a fixed environment, the Learns Team's performance can be considered an upper bound to TEAMSTER, because TEAMSTER learns both the team models and the environment model from experience, and a lower bound to PLASTIC-Model, which has both perfect models of the environment and teams.

Learns Environment (UCT + learned $\mathcal{E}$ + perfect $\mathcal{T}$): Learns Environment is an agent converse to Learns Team. Like Learns Team, PLASTIC-Model and TEAMSTER, it utilises an UCT planner combined with both an environment model $\mathcal{E}$ and initial team library $\mathcal{T}$. Its team library $\mathcal{T}$ is exact, like PLASTIC-Model, but its environment model $\mathcal{E}$ learned, like TEAMSTER. Similarly to Learns Team, Learns Environment can be considered an upper bound to TEAMSTER, since it has the perfect models of the teams, while TEAMSTER has to learn them and a lower bound to PLASTIC-Model, which has both perfect models of the environment and teams.

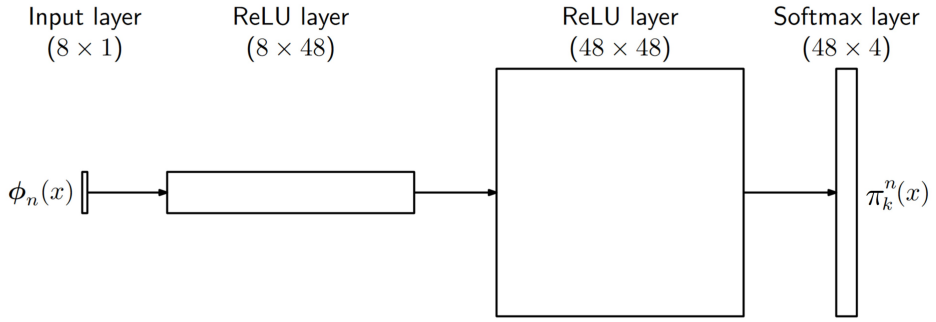
PLASTIC-Policy. PLASTIC-Policy has a policy library $\mathcal{Q} = \{q_1, \ldots, q_K\}$, where each q_k is the Q-function learned when playing team k. These Q-functions are learned during the interaction with the corresponding team and implicitly encode the teammate and environment behaviour. Alongside this policy library, PLASTIC-Policy also has a team library, $\mathcal{T} = \left\{\pi_k^{-\alpha}, k = 1, \ldots, K\right\}$, where each team model $\pi_k^{-\alpha}$ has an associated policy $q_k \in \mathcal{Q}$.

To train each policy q_k, we used a deep Q-network [Mnih et al., 2015], with the architecture depicted in Fig. 4.6(b).

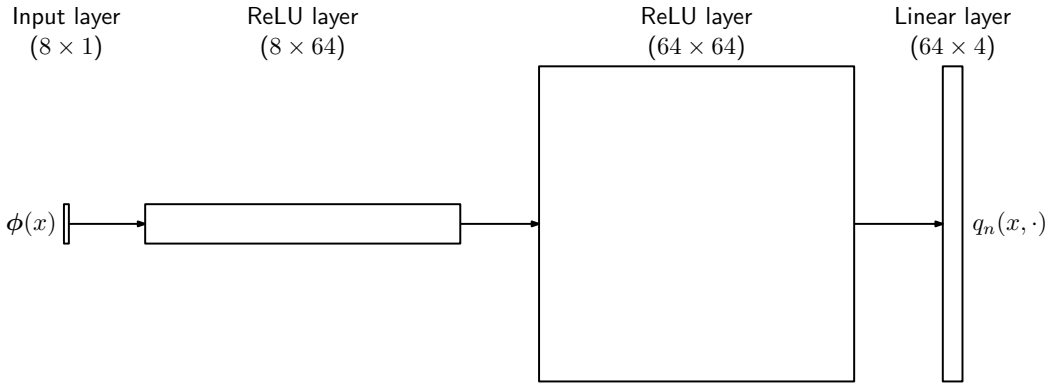
The network's inputs receive a similar set of features describing the state, and the output returns an estimated Q-value for the individual actions of the ad hoc agent. The network consists of two hidden layers with 64 rectified linear units and a linear output layer. As the agent interacts with the environment (and teammates), it stores a set of transitions of the form (x, a, r, y), where r is the reward received by the agent upon executing action a within state x and y is the next state. Such sample transitions are stored in an experience replay buffer with a capacity for $15,000$ samples. We trained the network using the Adam optimiser [Kingma and Ba, 2014] with an initial step size of 0.001 and batches of $T = 32$ samples. We also used the standard DQN loss,

$$L(\theta_q) = \frac{1}{T}\sum_{t=1}^{T}(r_t + \gamma \max_{a \in \mathcal{A}_0} q_{\theta'}(y_t, a) - q_{\theta_q}(x_t, a_t))^2,$$

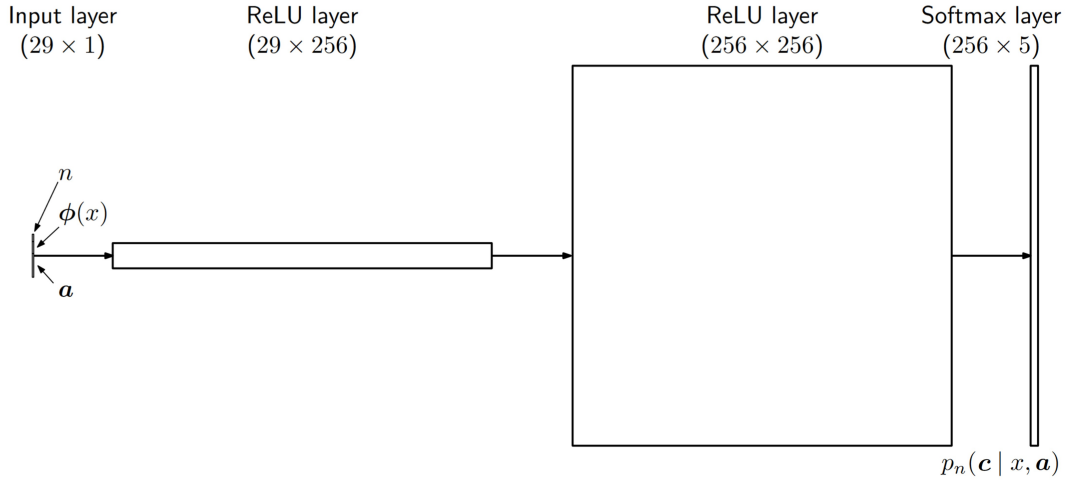
where γ is a discount factor (we used $\gamma = 0.95$), θ' denotes the parameters of the target network (i.e., a copy of the main network), and (x_t, a_t, r_t, y_t) represents the tth sample in a batch [Mnih et al., 2015]. We updated the target network after each batch update.



(a) Network for individual teammate policies, π_k^n.



(b) DQN network for the Pursuit domain, used as part of PLASTIC-Policy.



(c) Transition network for the Pursuit domain, used as part of TEAMSTER's environment model.

FIGURE 4.6: Different neural network architectures used in the Pursuit domain. Input layers are represented as $M \times 1$ feature vectors. Subsequent layers are labelled with the type of activation and (#inputs × #outputs).

Random policy. We also compare the ad hoc agents above with an agent that selects actions randomly. The performance of this agent can be considered as a lower bound to TEAMSTER and the other agents.

We run, for each of the four domains, each of the three teams, and each of the seven agents, 8 trials, adding up to a total of 672 trials. All trials were executed sequentially, without hand-picking the best runs. We present the accumulated episode reward as metric, allowing a comparison of the performances of the different ad hoc agents. Even though we do not report the average decision times, we expect PLASTIC-Policy to be, on average, faster than the UCT planner agents in terms of seconds to compute an action for a given state.

4.1.3 Models

Team models Even though the learned team models used by the ad hoc agents learn the team's policies from collected data, the team policies generating this data are actually handcoded (in other words, the goal of the team models is to learn handcoded policies, which are, in turn, stationary). In our implementation, each team model $\pi_k^{-\alpha}$ is implemented as a set of individual policies $(\pi_k^1, \ldots, \pi_k^N)$, as shown in Fig. 4.6(a). Since all domains contain a relatively large state space, the policies π_k^n correspond to categorical distributions approximated by feed-forward neural networks. The input vector of features $\boldsymbol{\phi}_n(x)$ describes the state as viewed by agent n, and the output is a probability distribution over the individual actions of agent n. The features of the state of agent n are passed through two hidden layers and then passed through a final softmax layer.

To learn the teammate models and build the the library, we train a policy q_k, afterwards used for PLASTIC-Policy, with each team k for a given number of time steps. During the interaction, we collect a dataset $\mathcal{D}_k = \left\{ (X_t, \boldsymbol{A}_{t,k}^{-\alpha}) \right\}$ with observed states and teammate actions, which was then used to train each network π_k^n in a supervised fashion, considering only the nth component of each $\boldsymbol{A}_{t,k}^{-\alpha}$. We trained the network using the Adam optimiser [Kingma and Ba, 2014] with batches of 32 samples and an initial step size of 0.001. We selected the network's hyperparameters by performing an F1-score evaluation on an independent, offline-generated dataset. At each time step t, the algorithm uses backpropagation to compute the gradient of the loss in equation 4.1 with respect to the network parameters, θ_{K+1}, and performs an Adam update.

TEAMSTER's environment model TEAMSTER maintains an environment model $\mathcal{E}$, which is learned from interacting with the environment. This model includes an estimate of the transition probabilities, $\hat{\mathbf{P}}$, and an estimate of the reward function, $\hat{r}$, that are learned in a supervised way as described next. Figure 4.6(c) depicts an overview of the architecture for our implementation.

Learning the transition probabilities. To implement our model $\hat{\mathbf{P}}$ of the transition probabilities, we use a feed-forward neural network designed to take advantage of the particular structure of our scenario. Specifically, since the state space $\mathcal{X}$ grows exponentially with the number of agents in the environment, and given the symmetry of the various agents' motion, we decompose $\hat{\mathbf{P}}(x' \mid x, \boldsymbol{a})$ as

$$\hat{\mathbf{P}}(x' \mid x, \boldsymbol{a}) = \prod_{n=0}^{N} p_n(s_n \mid x, \boldsymbol{a}), \tag{4.3}$$

where we use the fact that each state $x \in \mathcal{X}$ can be written as a tuple $x = (s_0, \ldots, s_N)$, with s_n representing the position of the nth agent in the environment (more specifically, $s_0, \ldots, s_{N-1}$ correspond to the predators and s_N corresponds to the prey). Each p_n thus encodes the transition probabilities for agent n given the full state x and the joint action $\boldsymbol{a}$. The decomposition in (4.3) enables significantly smaller models that are much more sample efficient to learn, at the cost of not being able to learn collision dynamics perfectly. Each p_n is then a categorical distribution over the possible next state of agent n.

The network's input is a concatenation of the following: (i) an integer $n \in \{0, \ldots, N\}$, where $0, \ldots, N-1$ correspond to the predators and N corresponds to the prey, encoded in a one-hot representation; (ii) a set of features $\boldsymbol{\phi}(x)$, representing the distances between the agent n and the other agents (predators and/or prey); and (iii) the actions of the four predator agents, encoded in a one-hot representation. The network's output layer consists of a softmax layer with five outputs, each representing the probability of agent n moving in the four directions or staying in the same cell.

We trained the network by resorting to an experience replay buffer, with data collected online, with a maximum size of 15,000 samples. We used the Adam optimiser [Kingma and Ba, 2014] with a batch size of 32 samples and an initial learning rate of 0.001. At each iteration, the algorithm uses backpropagation to compute the gradient of the loss with respect to the network's parameters, θ_P, and perform an Adam update. All hyperparameters were tuned empirically by performing an F1-score evaluation on an offline-generated dataset.

Learning the reward function. We also model the reward function as a feed-forward neural network. Much like the transition probabilities, we take advantage of our domain knowledge to improve the sample efficiency of learning the reward function by designing the network's architecture accordingly. The reward network's input is a vector of features describing the distance between the prey and each predator agent. The network has a single hidden layer with 64 ReLU units, while the output layer has a single sigmoid unit. The reward is 1 if the predators capture the prey and 0 otherwise, thus verifying UCT's conditions, in which the reward should be a scalar between 0 and 1 [Kocsis and Szepesvári, 2006]. As before, we use an experience replay buffer, with data collected online, with a maximum size of 15,000 samples. In addition, we use the Adam optimiser [Kingma and Ba, 2014] with a batch size of 32 samples and an initial learning rate of 0.001. At each iteration, the algorithm uses backpropagation to compute the gradient with respect to the network's parameters, θ_r, and perform an Adam update. All hyperparameters were tuned empirically by performing an F1-score evaluation on an offline-generated dataset.

The update to parameters is denoted as a call to function UPDATEENVIRONMENTMODEL in line 11 of Algorithm 1. All hyperparameters for the MCTS, Environment Model and DQNs can be found in Tables 4.1,4.2 and 4.3 respectively.

4.1.4 Experimental methodology

In this work, we study if explicitly learning the dynamics of the environment by relying on model-based techniques allow an ad hoc agent to more efficiently adapt to new teams on-the-fly. In order to test whether our proposed approach (TEAMSTER)

TABLE 4.1: Hyperparameters used for the UCT planner (shared by TEAMSTER and PLASTIC-Model).

Environment	Total Iterations	Maximum Rollout Depth	UCB Exploration	Discount Factor
Pursuit	100	5	$\sqrt{2}$	0.95
NTU	100	10	$\sqrt{2}$	0.95
ISR	300	50	$\sqrt{2}$	0.95
Pentagon	300	25	$\sqrt{2}$	0.95

TABLE 4.2: Architecture used for TEAMSTER's environment model and the team models (same for TEAMSTER, PLASTIC-Model and PLASTIC-Policy), for all environments.

Model	Adam Learning Rate	Hidden Layers	Hidden Units per Layer	Activation Function
Transition	0.001	2	256	ReLU
Rewards	0.01	2	64	ReLU
Team	0.001	2	48	ReLU

TABLE 4.3: Architecture used for PLASTIC-Policy's DQNs.

Environment	Adam Learning Rate	Hidden Layers	Hidden Units per Layer	Activation Function	ϵ-Greedy Start ϵ	ϵ-Greedy End ϵ	Initial Random Steps (Buffer Initialisation)	Target Update Frequency (Steps)
Pursuit	0.01	2	64	ReLU	0.50	0.05	0	1
NTU	0.01	2	512	ReLU	0.75	0.05	10000	3
ISR	0.001	2	512	ReLU	0.95	0.05	15000	4
Pentagon	0.001	2	512	ReLU	0.95	0.05	15000	4

is better than PLASTIC-Policy at more efficiently using its experience when moving from team to team, we setup an evaluation procedure where the ad hoc agent is able to learn how to interact with each team sequentially and evaluated in between these interactions on all teams. For each environment and for each ad hoc agent, our evaluation procedure is as it follows:

1. The agent is paired with team τ_i;
2. The agent together with team τ_i interacts with the environment for T timesteps; the ad hoc agent collects experience and learns from it. The timesteps for each domain were chosen by assessing how long it would the policy algorithms used by PLASTIC-Policy to converge to a performance close to that of PLASTIC-Model.
3. Every 10% of the interaction, the learning is paused and the team's performance is evaluated by measuring the accumulated reward. These results allow studying how each agent behaves from a pure reinforcement learning perspective;
4. After the T timesteps, the interaction ends and the ad hoc agent stores the information about team τ_i in its library;
5. The agent is deployed into each of the teams τ_1, τ_2 and τ_3 and evaluated by measuring the team's accumulated reward. This step allows for assessing how experience collected from interacting with a single team can be used to perform with the rest.
6. The process repeats with team τ_{i+1}

To summarise, our evaluation has two main goals:

1. Assessing whether TEAMSTER's model-based design provides performance advantages over PLASTIC-Policy's model-free design, when sequentially learning new teams.

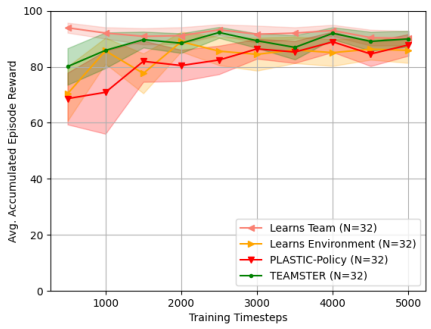
2. Assessing how going from an approach using an UCT planner relying perfect models (PLASTIC-Model) to an approach using an UCT planner relying on learned models (TEAMSTER) affects performance. This is achieved by evaluating PLASTIC-Model, TEAMSTER and the two baselines Learns Team and Learns Environment.

4.1.5 Additional Experiments

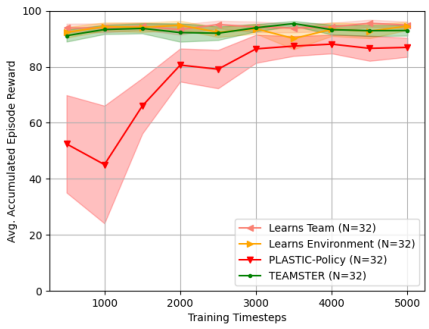
Apart from the main evaluation detailed in section 4.1.4, we conduct two additional experiments on the Pursuit domain with the PLASTIC-Model, PLASTIC-Policy and TEAMSTER agents: (i) an experiment where all agents are allowed to interact with a new team for a three times as many timesteps as in the prior evaluation and (ii) an experiment where all agents interact with a larger version of the environment (10×10):

- *Larger World Adaptation Experiment.* This experiment has the goal of assessing whether TEAMSTER's continuously trained environment model is better able to adapt to a larger environment when compared with PLASTIC-Policy. In this stage we evaluate all agents in an ad hoc teamwork setting where the environment increases in size from 5×5 to 10×10. We repeat the same procedure of the first evaluation. Therefore, this evaluation's primary goal is to show that our approach can adapt to a larger environment and, subsequently, a more complex task (showcasing its scalability). In addition, the three ad hoc agents use the knowledge acquired during the original evaluation. We note, however, that—by construction—only TEAMSTER updates its environment model. This difference between the three agents is one of the differentiating factors between TEAMSTER and the two PLASTIC algorithms: PLASTIC-Model merely uses its environment model. PLASTIC-Policy, on the other hand, does not maintain an explicit environment model: it is "embedded" in the teammate representation, and the agent cannot update the environment model independently of the teammate model.

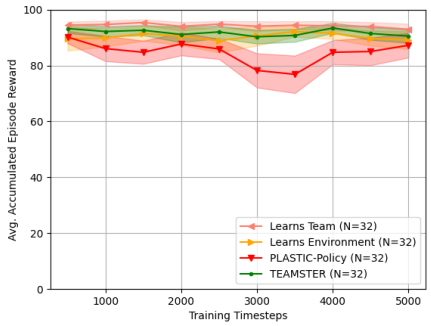
- *Performance Convergence Experiment.* This final experiment has the goal of studying the asymptotic performance of both TEAMSTER and PLASTIC-Policy, and comparing them against the performance of PLASTIC-Model. We aim to answer two research questions: (i) Even though PLASTIC-Model does not rely on function approximation techniques, can TEAMSTER still converge to PLASTIC-Model's performance, given enough time to learn? (ii) Does PLASTIC-Policy eventually converge to TEAMSTER's performance, given enough time and/or data to learn? In order to force PLASTIC-Model to learn the team model as well (effectively reducing PLASTIC-Model to the "Learns Team" agent), we setup a new team for this experiment by mixing the different teammate types from the three previous teams. We refer to this new team as 'Mixed'. Agents are then placed in an environment with the Mixed team, and interact with it for three times as long as with each of the three original teams ($15,000$ timesteps). Every 500 timesteps, we stop the interaction and assess the performance of each agent.



(a) Team τ_1, Pursuit.



(b) Team τ_2, Pursuit.



(c) Team τ_3, Pursuit

FIGURE 4.7: Pre-training plots for the four learning ad hoc agents, on each team on the Pursuit domain. Shaded areas correspond to the confidence interval of 95%.

4.1.6 Results

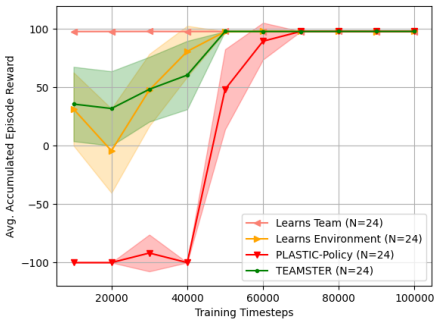
This section presents the results for our evaluation.

Standard Ad Hoc Evaluation

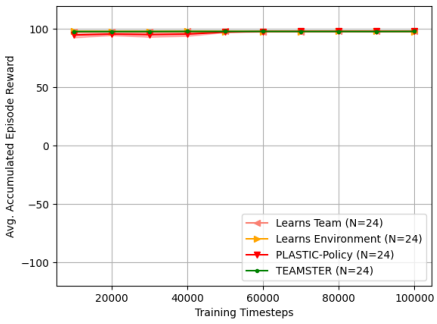
Figures 4.7, 4.8, 4.9 and 4.10 display the pre-training plots for the four learning agents on the four domains (respectively).

The first observation we can make from these results is that in all domains, when the first team is being learned (τ_1), the model-based methods (TEAMSTER, Learns Environment and Learns Team) are quicker to converge on their final performance than the model-free method (PLASTIC-Policy). These results are expected, as when learning the first team with an empty library, all four agents are rendered as pure-reinforcement learning algorithms. It is, therefore, expected for model-based methods to be more sample-efficient than model-free methods.

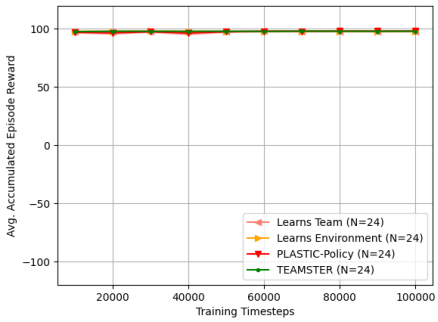
The second observation we can make is that when keeping their prior knowledge and learning how to interact with the second team (τ_2), the model-based methods have the most stable learning curves. PLASTIC-Policy, the model-free method, has in most cases, drops in performance. These drops in performance for PLASTIC-Policy are most visible in the Pursuit and Pentagon domains. In the Pursuit domain, the policy learned for team τ_1 was completely unable to adapt in the beginning of the interaction, meaning that the learned policy for τ_1 is not useful at all for assisting team τ_2. With the Pentagon domain, we can see that the drops in performance occur more towards the middle of the interaction. The most plausible explanation for



(a) Team τ_1, NTU.



(b) Team τ_2, NTU.



(c) Team τ_3, NTU

FIGURE 4.8: Pre-training plots for the four learning ad hoc agents, on each team on the NTU domain. Shaded areas correspond to the confidence interval of 95%.

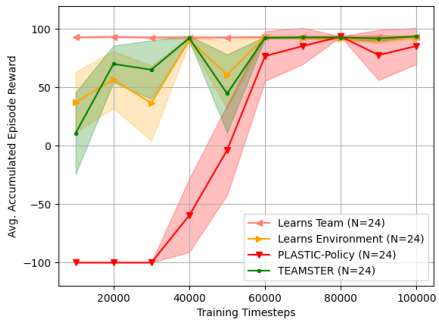
this phenomenon is that at the beginning, the policy learned for team τ_1 was also effective at assisting team τ_2. Afterwards, when PLASTIC-Policy's learning team model becomes effective in identifying the current team, the new policy is then more likely to be selected. This phenomenon is also observed when PLASTIC-Policy learns team τ_3, with smaller drops in performance in the Pursuit domain than in the Pentagon domain.

On the NTU and ISR domains, however, we can see that even before a new policy is learned, the policy learned for the previous team is able to attain a better performance than in the Pursuit and Pentagon domains.

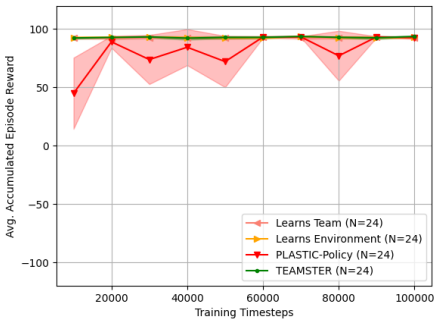
The performances of TEAMSTER and the other two model-based ablations (Learns Team and Learns Environment), suggest that explicitly learning the dynamics of the environment (or having a handcoded model, in the case of the Learns Team agent) brings an advantage when adapting to new teams. We can see in all domains that apart from when learning the first team (τ_1), the performances do not suffer any drops.

The four agents, after interacting with each of the three teams, are also evaluated alongside PLASTIC-Model, the original teammate from the team and a random policy. We report the accumulated episode reward measured in each trial. Tables 4.4, 4.5, 4.6, 4.7 showcase the average accumulated episode reward for all agents, on each domain.

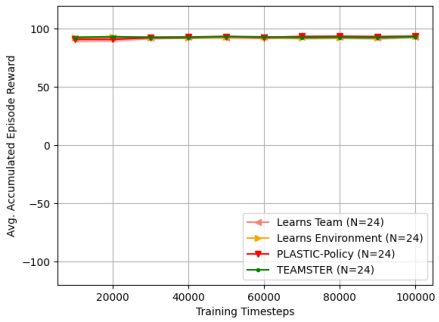
In all four domains, the first observation we can make is that all agents were able to achieve similar performances after training for the team in question. For instance, we can see that PLASTIC-Model has the lowest variance of all ad hoc agents, which



(a) Team τ_1, ISR.



(b) Team τ_2, ISR.



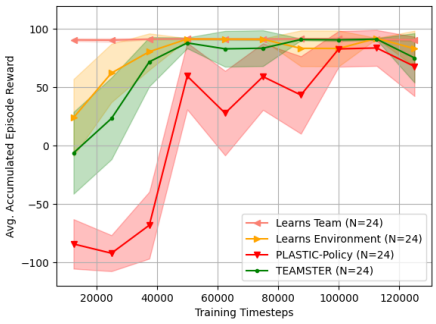
(c) Team τ_3, ISR

FIGURE 4.9: Pre-training plots for the four learning ad hoc agents, on each team on the ISR domain. Shaded areas correspond to the confidence interval of 95%.

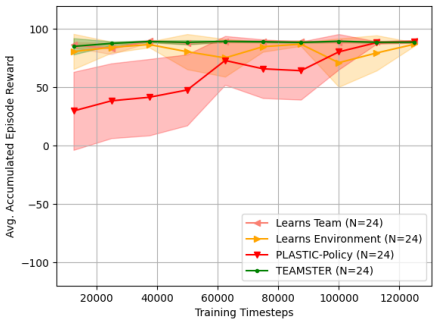
TABLE 4.4: Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the Pursuit domain.

	Eval on τ_1		**Eval on τ_2**		**Eval on τ_3**	
Original Teammate	92.84	(±6.76)	94.28	(±5.35)	94.09	(±4.69)
Random Policy	39.84	(±63.84)	56.88	(±35.47)	45.66	(±62.64)
After Learning τ_1						
PLASTIC-Policy	90.25	(±7.77)	82.06	(±21.49)	79.31	(±19.85)
TEAMSTER	92.69	(±5.45)	91.47	(±4.95)	89.91	(±8.01)
PLASTIC-Model	93.47	(±4.69)	84.28	(±12.86)	88.91	(±9.55)
Learns Environment	85.59	(±16.93)	81.78	(±15.16)	82.94	(±19.45)
Learns Team	92.25	(±7.33)	90.72	(±7.7)	92.72	(±5.67)
After Learning τ_2						
PLASTIC-Policy	86.31	(±13.11)	86.81	(±11.04)	84.19	(±12.6)
TEAMSTER	89.22	(±11.13)	92.94	(±5.7)	90.94	(±7.13)
PLASTIC-Model	94.66	(±2.68)	96.47	(±2.83)	92.53	(±4.98)
Learns Environment	85.62	(±11.59)	93.19	(±4.81)	89.78	(±9.78)
Learns Team	90.25	(±7.79)	95.66	(±3.07)	93.5	(±5.41)
After Learning τ_3						
PLASTIC-Policy	84.81	(±13.28)	86.41	(±8.26)	76.78	(±43.89)
TEAMSTER	91.16	(±8.64)	94.56	(±3.46)	93.06	(±5.63)
PLASTIC-Model	94.5	(±3.83)	94.25	(±4.42)	94.88	(±3.72)
Learns Environment	84.41	(±14.19)	93.53	(±5.16)	87.81	(±12.39)
Learns Team	90.38	(±9.14)	95.75	(±3.64)	94.34	(±4.1)

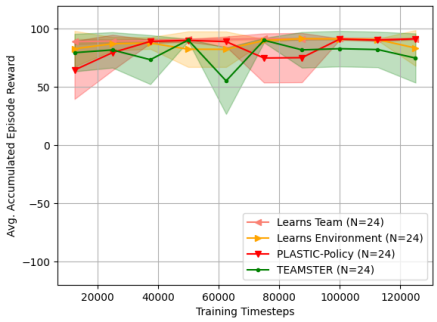
is expected given its perfect models of both the environment and teams. All other four ad hoc agents (PLASTIC-Policy, TEAMSTER, Learns Environment and Learns Team), showcase a higher variance in their evaluation. This is expected, given that different initialisations of their learning model's parameters are expected to yield different final models. It is interesting, however, that this variance is very evident in the results obtained by the Learns Environment agent after learning its second



(a) Team τ_1, Pentagon.



(b) Team τ_2, Pentagon.



(c) Team τ_3, Pentagon

FIGURE 4.10: Pre-training plots for the four learning ad hoc agents, on each team on the Pentagon domain. Shaded areas correspond to the confidence interval of 95%.

TABLE 4.5: Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the NTU domain.

	Eval on τ_1		**Eval on τ_2**		**Eval on τ_3**	
Original Teammate	98.0	(±0.0)	98.0	(±0.0)	97.83	(±0.55)
Random Policy	79.58	(±22.72)	69.0	(±30.37)	69.12	(±30.12)
After Learning τ_1						
PLASTIC-Policy	98.0	(±0.0)	98.0	(±0.0)	97.75	(±0.66)
TEAMSTER	97.96	(±0.2)	98.0	(±0.0)	97.92	(±0.4)
PLASTIC-Model	98.0	(±0.0)	97.96	(±0.2)	97.58	(±0.81)
Learns Environment	98.0	(±0.0)	97.92	(±0.28)	97.96	(±0.2)
Learns Team	98.0	(±0.0)	97.96	(±0.2)	98.0	(±0.0)
After Learning τ_2						
PLASTIC-Policy	98.0	(±0.0)	98.0	(±0.0)	97.92	(±0.4)
TEAMSTER	98.0	(±0.0)	98.0	(±0.0)	97.92	(±0.4)
PLASTIC-Model	97.92	(±0.28)	98.0	(±0.0)	97.92	(±0.4)
Learns Environment	97.83	(±0.37)	97.96	(±0.2)	97.88	(±0.6)
Learns Team	97.92	(±0.4)	98.0	(±0.0)	97.29	(±1.51)
After Learning τ_3						
PLASTIC-Policy	98.0	(±0.0)	98.0	(±0.0)	98.0	(±0.0)
TEAMSTER	98.0	(±0.0)	98.0	(±0.0)	97.88	(±0.44)
PLASTIC-Model	98.0	(±0.0)	97.96	(±0.2)	97.67	(±0.75)
Learns Environment	97.96	(±0.2)	97.96	(±0.2)	97.96	(±0.2)
Learns Team	97.96	(±0.2)	98.0	(±0.0)	97.71	(±0.68)

team (τ_2). Compared to TEAMSTER, which learns both the model of the environment and the models of the teams, the Learns Environment agent only learns the model of the environment. Since this is the only difference between the agents, it follows that it must be the cause for its higher variance. The hand-coded team models always return, for a given state, the same team policy, while the learned team models might sometimes return different policies. This high 'certainty' might be the cause for a

TABLE 4.6: Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the ISR domain.

	Eval on τ_1		**Eval on τ_2**		**Eval on τ_3**	
Original Teammate	94.0	(±0.0)	94.0	(±0.0)	94.0	(±0.0)
Random Policy	60.0	(±28.4)	69.17	(±24.5)	64.04	(±22.52)
After Learning τ_1						
PLASTIC-Policy	74.0	(±38.48)	88.71	(±18.99)	85.75	(±26.16)
TEAMSTER	93.04	(±1.97)	80.33	(±31.05)	93.71	(±0.79)
PLASTIC-Model	92.79	(±1.55)	92.0	(±2.12)	92.71	(±2.88)
Learns Environment	93.46	(±1.29)	92.29	(±2.19)	92.92	(±3.12)
Learns Team	92.96	(±1.88)	92.5	(±3.34)	93.33	(±1.4)
After Learning τ_2						
PLASTIC-Policy	93.21	(±1.91)	93.12	(±1.64)	93.08	(±1.75)
TEAMSTER	92.96	(±1.77)	93.08	(±1.98)	93.38	(±1.65)
PLASTIC-Model	92.71	(±2.15)	93.17	(±1.28)	93.25	(±1.05)
Learns Environment	93.21	(±1.63)	92.83	(±1.91)	93.12	(±1.76)
Learns Team	92.71	(±1.86)	92.08	(±3.17)	93.33	(±1.7)
After Learning τ_3						
PLASTIC-Policy	93.29	(±1.88)	93.21	(±1.63)	93.42	(±0.81)
TEAMSTER	93.42	(±1.35)	92.75	(±1.81)	93.04	(±2.01)
PLASTIC-Model	92.71	(±2.01)	92.62	(±1.98)	93.62	(±0.7)
Learns Environment	93.04	(±1.81)	93.21	(±1.47)	92.62	(±2.67)
Learns Team	93.0	(±1.85)	91.96	(±3.96)	92.17	(±2.53)

TABLE 4.7: Average accumulated episode reward measured in each trial for each of the seven agents on each of the three teams after learning each team on the Pentagon domain.

	Eval on τ_1		**Eval on τ_2**		**Eval on τ_3**	
Original Teammate	91.67	(±0.94)	88.0	(±0.0)	91.83	(±0.9)
Random Policy	66.17	(±29.44)	61.92	(±26.31)	80.79	(±12.8)
After Learning τ_1						
PLASTIC-Policy	80.29	(±30.74)	81.54	(±24.98)	76.54	(±34.69)
TEAMSTER	83.54	(±25.61)	84.29	(±7.07)	91.04	(±2.68)
PLASTIC-Model	91.12	(±1.94)	82.67	(±17.79)	91.67	(±1.34)
Learns Environment	91.54	(±2.38)	77.08	(±26.42)	83.42	(±25.52)
Learns Team	91.83	(±0.99)	83.88	(±5.95)	90.88	(±1.45)
After Learning τ_2						
PLASTIC-Policy	91.5	(±1.87)	89.25	(±1.81)	87.46	(±18.55)
TEAMSTER	86.5	(±18.35)	89.17	(±1.43)	90.96	(±1.65)
PLASTIC-Model	91.04	(±2.51)	83.83	(±12.86)	90.5	(±2.52)
Learns Environment	87.83	(±18.58)	85.79	(±5.47)	89.67	(±5.97)
Learns Team	90.79	(±2.12)	88.71	(±1.34)	90.54	(±1.55)
After Learning τ_3						
PLASTIC-Policy	90.92	(±2.86)	89.04	(±2.19)	91.38	(±1.49)
TEAMSTER	83.79	(±25.15)	82.83	(±9.18)	87.62	(±18.58)
PLASTIC-Model	91.54	(±1.61)	84.08	(±10.15)	91.21	(±1.53)
Learns Environment	84.0	(±25.66)	84.46	(±17.88)	90.83	(±2.03)
Learns Team	91.33	(±1.21)	85.38	(±5.2)	91.04	(±1.62)

lower performance, as the policy used for planning, if misidentified, might misdirect the planning for that team. Our results support this hypothesis. For instance, after learning team τ_1, the Learns Environment model was able to attain a virtually optimal performance in interacting with that same team. However, after learning team τ_2, its performance, even though falling on a closely related interval, dropped on average and increased its variance. Misidentifications of the team are the most likely cause for this drop, as the planner used by the agent may sometimes provide a sub-optimal action. Finally, we can see that PLASTIC-Policy also has a slightly lower average. TEAMSTER, on the other hand, has a slightly higher average. These results therefore support our hypothesis that relying on model-based methods might provide an advantage to ad hoc agents.

The second assessment we can make with these results is in how each agent is able to use information with teams that are not in its known set and leverage what they have already learned. From the results presented in Tables 4.4, 4.5, 4.6 and 4.7, we can observe that in most domains, experience acquired for a single team allows for a near-optimal performance on the other two teams, using the Original Teammate as baseline. This is most evident in the smallest domain (NTU), where the variance was

the lowest. This result is expected, since the dynamics of the environment and goal do not change, only the way the teammates behave. One clear exception, however, is the Pursuit domain. In Table 4.4, we can see an evident drop in performance when agents are evaluated on team τ_2, after learning only how to interact with team τ_1. These results hint that team τ_2 of the Pursuit domain requires a new team model learned. Our results support this hypothesis as two observations can be made. First, PLASTIC-Policy has a clear drop in performance when interacting with team τ_2 after learning a policy for team τ_1. This showcases that a model-free policy learned for team τ_1 is not near-optimal for interacting with team τ_2. Secondly, the agents with the hand-coded team models (PLASTIC-Model and Learns Environment) were actually outperformed by both TEAMSTER and Learns Team, further supporting our previous hypothesis that planning with a hand-coded model might not always provide an advantage when teams are misidentified. As previously discussed, handcoded team models (unlike the learned team models) always return the same team policy for a given state. This 'certainty' in the model, in this specific case, negatively affects planner whenever teams are misidentified. Finally, the last observation we can make, is that all agents, after learning the first two teams, are able to obtain a good performance on team τ_3.

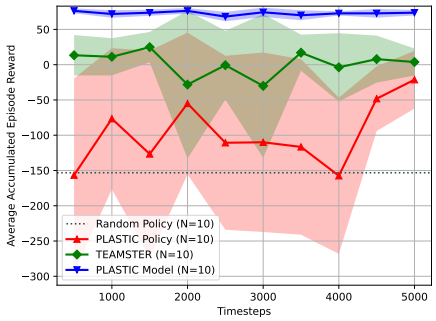
Larger World Adaptation Experiment

In this experiment we increase the size of the Pursuit world from a 5×5 grid into a 10×10 grid, in order to assess if TEAMSTER's environment model is able to better adapt a larger scenario with the same dynamics than PLASTIC-Policy. For this experiment, we evaluate only TEAMSTER, PLASTIC-Policy and PLASTIC-Model with each of the three teams and record the average accumulated episode reward. Figure 4.11 presents our results for the three agents interacting with the three teams, respectively.

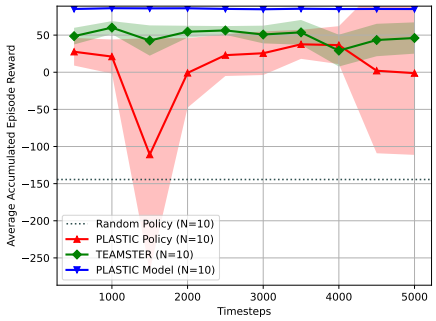
These results suggest three main conclusions. First, all agents had a substantial drop in performance compared with the results observed in the 5×5 world. This result is expected because the environment's dimension is significantly larger, and thus, the predators take more steps to capture the prey. Second, our approach (TEAMSTER) outperforms the PLASTIC-Policy algorithm again. In this larger scenario, however, we can see a performance overlap when interacting with team τ_2, at around time step 4000. We can also observe that PLASTIC-Policy had a higher variance than TEAMSTER, sometimes even performing worst than the Random policy. This phenomenon hints that TEAMSTER's environment model might provide a more stable solution when adapting to larger worlds, when compared with PLASTIC-Policy's DQNs trained for different world sizes. If we look into Figure 4.11(c), we can see that when learning a DQN for team τ_3, PLASTIC-Policy's performance increases after timestep 4500, which supports this hypothesis that is worst performance is caused by using a DQN policy which was trained not only for a different team, but also for a different world size. Finally, as expected, PLASTIC-Model was able to outperform the two other agents.

Performance Convergence Experiment

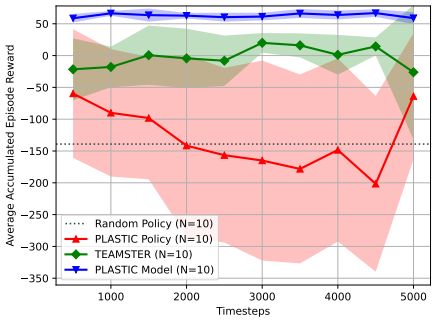
We conduct a final experiment in the Pursuit domain to assess the asymptotic performance of each learning agent. We deploy each agent into a team of four agents, this time each with a different policy (one with a teammate aware policy, another with a greedy policy and a third with a probabilistic destinations policy). We call the new team 'Mixed'.



(a) Team τ_1, 10×10.



(b) Team τ_2, 10×10.



(c) Team τ_3, 10×10.

FIGURE 4.11: Average accumulated reward for the different approaches on each team on a 10×10 world, in the Pursuit domain. Shaded areas correspond to the confidence interval of 95%.

We then train PLASTIC-Model, PLASTIC-Policy, TEAMSTER with this new team, for three times as many timesteps as in the original evaluation. Given that PLASTIC-Model does not have, in its library, this new team, it is forced to learn a new team model, reducing it to a Learns Team agent. Similar to the original evaluation, we stop the interaction every 500 timesteps and evaluate all agents. We conduct a total of $N = 8$ trials. Figure 4.12 showcases the learning curves for all agents averaged over all trials with a confidence of 95%.

As expected, we can observe that PLASTIC-Model has the most stable learning performance, as the only model PLASTIC-Model is learning, is the team model. TEAMSTER and PLASTIC-Policy, on the other hand, both showcase learning curves with higher variances. Even though TEAMSTER hints of a better performance on the first six evaluation checkpoints (i.e., between 500 and 3000 steps), it is afterwards matched by PLASTIC-Policy with both agents showing similar learning curves. By the end of the interaction, all ad hoc agents converge to the same asymptotic performance.

4.2 Conclusion

This chapter addressed our first research sub-problem:

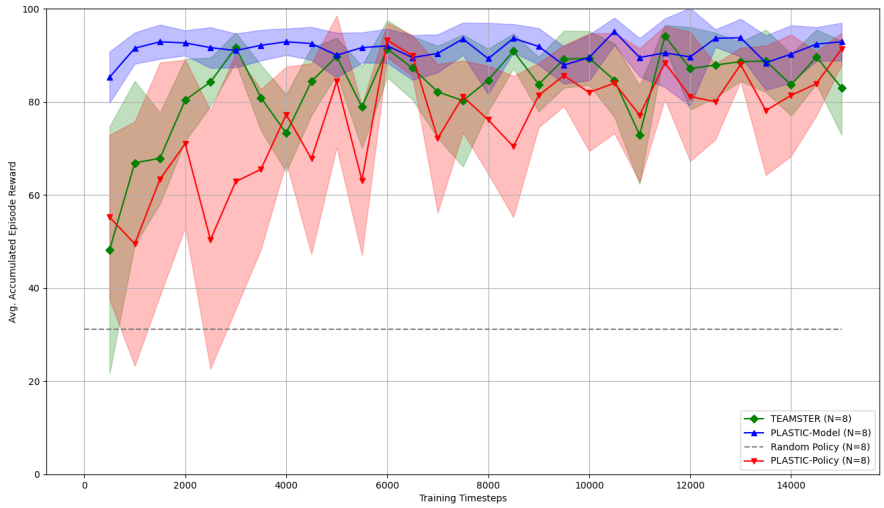


FIGURE 4.12: Asymptotic performance of each agent in an additional 'Mixed' team of the Pursuit domain. Shaded areas correspond to the confidence interval of 95%.

How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?

By taking advantage of model-based reinforcement learning techniques, which not only learn the behaviour of the teammates but also the dynamics and goal of the task being performed, our contribution, TEAMSTER, is: (i) more robust and sample-efficient than PLASTIC Policy, since it does not have to learn the environment's dynamics from scratch whenever it comes into contact with a new team and (ii) more flexible than PLASTIC Model, since it can perform the task without a significantly large amount of domain knowledge—namely, not requiring a perfect model of the environment but effectively learning this model by interacting with the environment. Our results show that TEAMSTER can adapt continuously to sudden changes in team and task, therefore being a more reliable algorithm for ad hoc teamwork. Throughout this contribution, we assumed TEAMSTER was not only able to not only observe the actions of the teammates and the state of the environment, but was also able to learn via reinforcement learning as it interacted with newer teams.

◇

In the next chapter, we will drop the three assumptions made by TEAMSTER and the PLASTIC algorithms (fully observable states, visible teammate actions). In order to address our second research sub-problem, we will introduce two novel approaches for ad hoc teamwork which rely only on prior knowledge and partial observations of the environment.

Chapter 5

Bayesian Online Prediction for Ad Hoc Teamwork

This chapter addresses our second research sub-problem:

> *How can an autonomous agent adapt to unknown teams and tasks being performed without being able to observe their actions and the state of the environment?*

In the previous chapter (chapter 4), we have introduced a model-based reinforcement learning approach for ad hoc teamwork, TEAMSTER, which we showcased was able to adapt, on-the-fly, to sudden changes in both the environment and teammate behaviour. However, as discussed in chapter 1, both TEAMSTER and current approaches for the setting of ad hoc teamwork [Barrett, 2014, Barrett et al., 2017] make three assumptions which are unfeasible in real-world scenarios which may involve not only robots but also humans as teammates: (i) the ad hoc agent is able to observe the state of the environment, (ii) the ad hoc agent is able to observe the actions of the teammates and (iii) the ad hoc agent has, when performing ad hoc teamwork, a reward signal available which allows it to learn via trial and error.

In this chapter, we drop all these assumptions and further expand the literature in ad hoc teamwork by introducing two novel approaches for ad hoc teamwork under partial observabilitiy. We address our second research sub-problem — *how can an autonomous agent adapt to unknown teams and tasks being performed without being able to observe their actions and the state of the environment?* In section 5.1, we start by dropping the assumptions that the actions of the teammates are visible to the ad hoc agent. We introduce an approach which only assumes the environment is fully observable, making use of the state of the environment in order to identify, from a library of possible models, the most likely team and task being performed. In section 5.2, we drop the last assumption, that the state of the environment is fully observable to the ad hoc agent, and assume instead that the ad hoc agent only has access to an imperfect observation of the environment (obtained, for example, via some sensors). Relying on a library of possible models, both approaches resort to a Bayesian online inference algorithm in order to identify both current teammate behaviour and the tasks being performed.

5.1 Bayesian Online Prediction for Ad Hoc Teamwork (BOPA)

We start by dropping the assumptions made by TEAMSTER (chapter 4) that the ad hoc agent is able to observe the actions of the teammates and has an explicit reward signal (which could be used to learn how to interact with the team in performing their task during the ad hoc evaluation phase).

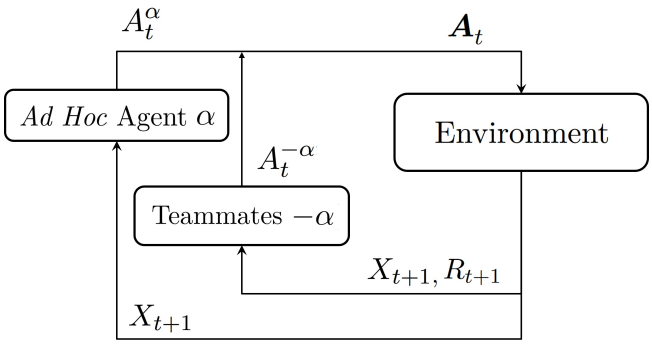


FIGURE 5.1: Interaction between an ad hoc agent α and teammates $-\alpha$, described by an MMDP M. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}

We introduce a novel approach, named BOPA, which we evaluate throughout an empirical evaluation where we measure four different metrics, further discussed in the evaluation section. Under our set of assumptions, we also contribute to the literature in ad hoc teamwork by presenting the first ad hoc teamwork algorithm which does not rely on visible teammates' actions.

5.1.1 Preliminaries

We assume the standard ad hoc teamwork problem formulation layed down in section 3.1.1. Furthermore, we assume the ad hoc agent α is, at each step t unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1} (resulting from the current state X_t and joint action A_t). Instead, the ad hoc agent is only able to observe, at each step t, the current state of the M, X_t and its own action, A_t^α. Figure 5.1 illustrates the interaction between the ad hoc agent and the teammates in the environment, described by an MMDP M where the ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}.

We then consider the ad hoc agent α to have access to a set of prior knowledge containing K possible MMDPs and teammate policies, i.e., $\left\{(M_1, \pi_1^{-\alpha}), \ldots, (M_K, \pi_K^{-\alpha})\right\}$. We then reduce each MMDP $M_k \in \left\{(M_1, \pi_1^{-\alpha}), \ldots, (M_K, \pi_K^{-\alpha})\right\}$ to an MDP $M_k^\alpha = (\mathcal{X}, \mathcal{A}^\alpha, P_k^\alpha, r_k, \gamma)$ and consider the 'target' MMDP M to likewise be reduced to an MDP M^α according to $\pi^{-\alpha}$ using Equation 2.2.

Figure 5.2 illustrates the interaction between the ad hoc agent and the teammates in the environment, described by an MDP M^α which reduces M according to $\pi^{-\alpha}$ using Equation 2.2 where the ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}.

We then assume each MDP M_k^α is solved in order to obtain its optimal policy π_k^α. Finally, we store each MDP M_k^α in a *model library* $\mathcal{M} = \{M_1^\alpha, \ldots, M_K^\alpha\}$ and each corresponding policy π_k^α in a *policy library* $\Pi = \{\pi_1^\alpha, \ldots, \pi_K^\alpha\}$.

5.1.2 Algorithm

We start by letting p_0 denote some (prior) probability distribution over the model library $\mathcal{L}$, with $p_0(m) = \mathbb{P}\left[M^\alpha = \mathcal{M}_m^\alpha\right]$, where M^α represents the reduced, unknown MMDP describing the environment and teammate policies. More generally, we define

$$p_t(m) = \mathbb{P}\left[M^\alpha = M_m^\alpha \mid \left\{X_0, A_0^\alpha, X_1, \ldots, X_{t-1}, A_{t-1}^\alpha, X_t\right\}\right]. \tag{5.1}$$

From Bayes theorem,

$$p_{t+1}(m_k) = \frac{1}{\rho} P_k^{\alpha}(X_{t+1} \mid X_t, A_t^{\alpha}) p_t(m_k),$$

where ρ is a normalisation constant.

At each time step t, the ad hoc agent selects an action A_t^{α} in the current state, X_t. To that purpose, it may choose to follow the action prescribed by any of the optimal policies in the policy library $\Pi = \{\pi_1, \ldots, \pi_M\}$. The agent is only able to observe the transition between states and its own action. After observing a transition (x, a^{α}, y), and independently of which policy is followed,

$$\mathbb{P}\left[(x, a^{\alpha}, y) \mid M^{\alpha} = M_m^{\alpha}\right] = P_m(y \mid x, a^{\alpha}).$$

The agent can thus update its current belief over which model M_m^{α} is the target model M^{α} using (5.1). We define the loss of policy selecting action a^{α} at time step t given that the target model is M_m^{α} as

$$\ell_t(a^{\alpha} \mid M_m^{\alpha}) = v^{\pi_m}(x) - q^{\pi_m}(x, a^{\alpha}),$$

where π_m is the solution to the MDP M_m^{α}.

It is important to note that both $v^{\pi_m}(x)$ and $q^{\pi_m}(x, a^{\alpha})$ can be computed offline when solving the MDP M_m^{α}. Note also that $\ell_t(a^{\alpha} \mid m) \geq 0$ for all a^{α}, and $\ell_t(a^{\alpha} \mid m) = 0$ only if $\pi_m(a^{\alpha} \mid x_t) > 0$. The action for the ad hoc agent at time step t can now be computed using our Bayesian setting as

$$\pi_t(a^{\alpha} \mid x) \triangleq \mathbb{P}\left[A_t^{\alpha} = a^{\alpha} \mid X_t = x\right] = \sum_{m=1}^{M} \pi_m(a^{\alpha} \mid x) p_t(m). \tag{5.2}$$

5.1.3 Evaluation

We evaluate BOPA in a simulated environment—*Panic Buttons*, or PB [Brockman et al., 2016]. PB is a benchmark grid-world environment where N agents must simultaneously press N buttons.

We consider three different configurations, which correspond to the different models in the ad hoc agent's library.[1] Each configuration is described as an MMDP

[1] In the PB environment, different configurations correspond to different positions for the buttons.

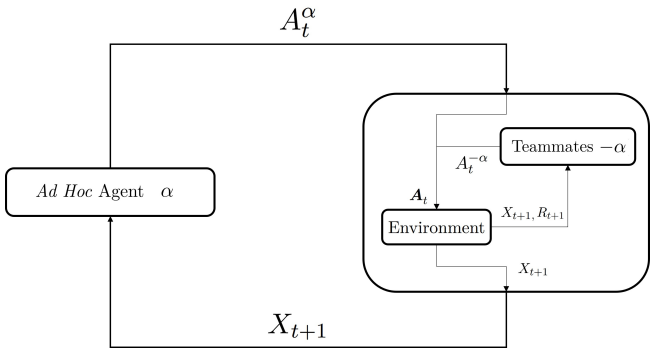


FIGURE 5.2: Interaction between an ad hoc agent α and teammates $-\alpha$, described by an MDP M^{α} that reduces M according to $\pi^{-\alpha}$ using Equation 2.2. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}.

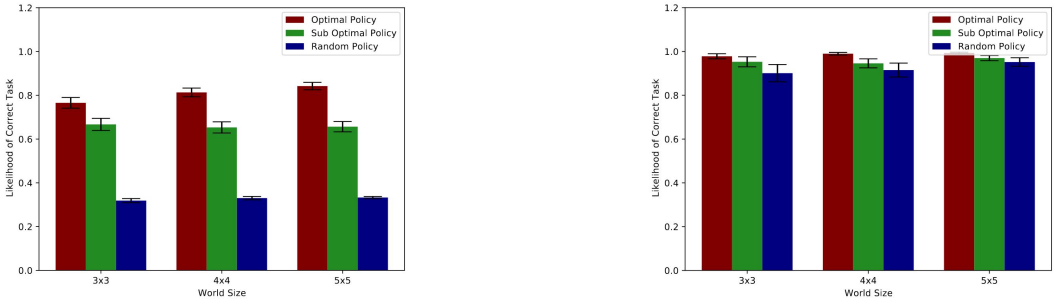


FIGURE 5.3: Probability of correct task averaged across the whole episode, $1/T \sum_t p_t(r)$ (left) and probability of correct task, at the last step of the episode, $p_T(r)$ (right). The error bars correspond to the variability in the agent's estimate.

with a distinct reward function. The joint optimal policies are computed using value iteration for the underlying MDP. In both environments, the ad hoc agent observes only the state of the teammate (i.e., its position in the environment) and must infer the task (i.e., configuration) and act accordingly.

Evaluation procedure

In the PB scenario, the ad hoc agent can only observe the state of the MMDP, and can observe neither the teammate's actions nor any reward. In our evaluation, we assess the scalability, efficiency and robustness to different teammates. To the best of knowledge, our work is the first addressing ad hoc teamwork problems where the ad hoc agent has only state information available. To evaluate our approach, we compare BOPA against two baselines: an "ad hoc agent" following a random policy (named *random*), and an "ad hoc agent" following optimal policy for the task at hand (named *greedy*). The two baselines provide upper and lower bounds on the performance of BOPA.

Metrics

In the PB scenario, the reported values consist of averages and 95% confidence intervals over 32 independent trials, where a single trial consists of running the three agents (greedy, BOPA, and random) against an unknown teammate. To gain some additional insight regarding the robustness of our approach, we pair the ad hoc agent with different teammates—an optimal teammate, that knows the task and acts optimally; a sub-optimal teammate, that knows the task but chooses not to act with a probability 0.3, and a teammate that acts randomly.

We report four different metrics that seek to assess effectiveness, efficiency, scalability and robustness to sub-optimal teammates. Effectiveness is measured by determining whether or not BOPA is able to identify the correct task. Efficiency is measured by evaluating whether or not the ad hoc agent is able to solve the task in near-optimal time. Scalability is assessed by observing the performance of the ad hoc agent in different problem sizes (3×3, 4×4 and 5×5 grids). Robustness is evaluated by reporting whether or not an ad hoc agent is able to cope with non-optimal teammates.

Results

We now present and discuss the results of our experiments.

TABLE 5.1: Average number of steps required for task completion.

	Optimal	Sub-optimal	Random
3×3 Greedy	2.7± 0.5	3.4± 1.2	24.2± 23.3
3×3 Bopa	3.3± 0.8	4.3± 2.3	35.9± 37.7
3×3 Random	25.2± 27.4	26.7± 28.2	144.8± 154.1
4×4 Greedy	4.0± 0.8	5.2± 1.9	54.0± 56.1
4×4 Bopa	4.7± 0.8	6.0± 2.4	61.0± 67.0
4×4 Random	47.3± 50.6	56.8± 59.2	497.4± 426.3
5×5 Greedy	5.3± 0.9	6.7± 1.6	85.7± 97.4
5×5 Bopa	5.8± 0.8	8.5± 3.7	168.8± 169.5
5×5 Random	104.7±106.9	103.3± 109.0	1120.4±1003.1

The results for the PB scenario are summarized in Fig. 5.3 and Table 5.1. The plots in Fig. 5.3 depict the ad hoc agent's ability to identify the unknown target task, r. Figure 5.3 (left) presents—for the different environment sizes and teammates—the likelihood of r according to the agent's belief, averaged across the whole trial, i.e., for an episode of length T,

$$p_{\text{ave}}(r^*) = \frac{1}{T} \sum_{t=1}^{T} p_t(r^*).$$

Figure 5.3 (right) presents the likelihood of r^* according to the agent's belief and the final step of the trial, i.e., for an episode of length T, $p_T(r^*)$.

The plots of Fig. 5.3 allow us to conclusively assess BOPA's effectiveness: in all environments and for all teammates, the algorithm is able to identify the target task with great certainty. The plot also shows successfully identifying the target task largely depends on the teammate's behaviours: if the teammate behaves in a misleading way (i.e., sub-optimally), this will sometimes lead to poor belief updates, hindering the algorithm's ability to identify the target task.

In terms of BOPA's efficiency (i.e., its ability to solve the target task), we can observe in Table 5.1 that the performance of BOPA—when playing with an optimal teammate—closely follows that of the greedy agent (i.e., the agent knowing the target task). This is in accordance with our results on effectiveness: since BOPA is able to quickly identify the target task, it performs near-optimally in all tasks.

In terms of scalability and robustness (i.e., how BOPA's performance depends on the size of the problem and the quality of the teammates), two interesting observations are in order. On one hand, the difference in performance between BOPA and the greedy agent attenuates for larger environments. This can be understood as the larger environments provide more data (i.e., teammate's action effects through state observations) for the ad hoc agent to recognize the action and immediately head to the goal. On the other hand, the negative impact of playing with sub-optimal teammates is larger for larger environments.

Taking these results into account, we can see that BOPA is not only able to identify the correct task with great certainty by inferring the teammate's behaviour through state observations, but also capable of adapting to non-optimal teammates by still being able to solve the tasks. Having now established a solid baseline for ad hoc teamwork where two assumptions were dropped: (i) the actions of the teammates are not visible and (ii) there is no explicit reward signal which the ad hoc agent may use after being deployed into the team, we will now expand upon BOPA by dropping the

assumption that the state is fully observable.

5.2 Ad Hoc Teamwork under Partial Observability (ATPO)

We then expand upon BOPA by dropping the assumption that the environment is fully observable. In section 5.1, we showcase that without being able to observe the actions of the teammates' nor the environment's reward signal, an ad hoc agent is able to infer the task and teammate behaviour given only the state of the environment. Given that in the real-world environment where robots might operate, sets of sensors are used to obtain information from their surroundings, the assumption that the state of the environment is fully observable is unfeasible. We therefore present a novel contribution which, to the best of our knowledge, introduces the first approach for ad hoc teamwork capable of integrating and assisting unknown teams in completing unknown tasks which only requires a partial observation of the environment (suited to address tasks involving *ad hoc robotic agents*, accommodating the natural perceptual limitations of robotic platforms [Genter et al., 2017]).

We evaluate the effectiveness of ATPO in nine benchmark multiagent problems, showcasing that ATPO can correctly assist unknown teammates in solving unknown tasks.

5.2.1 Preliminaries

We assume the same problem formulation from section 5.1.1 (assuming the ad hoc agent α is, at each step t unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}). Furthermore, we drop the assumption that the environment is fully observable, meaning that α is only able to observe, at each step t, an observation Z_t and its previous action A_{t-1}^{α}.

Assuming the environment was described by an MMDP M (which the ad hoc agent does not know beforehand), reduced to an MDP M^{α} using Equation 2.2 according to policy $\pi^{-\alpha}$, we can, in order to incorporate partial observability, define a POMDP $\hat{M}^{\alpha} = (\mathcal{X}, \mathcal{A}^{\alpha}, \mathcal{Z}, P^{\alpha}, O^{\alpha}, r, \gamma)$ which has M^{α} as an underlying MDP. Figure 5.4 illustrates the interaction between the ad hoc agent and the teammates in the environment, described by a POMDP $\hat{M}^{\alpha}$ with an underlying MDP M^{α} which reduces an MMDP M according to policy $\pi^{-\alpha}$ using Equation 2.2 where the ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$, the reward signal R_{t+1} and the state of the environment X_t.

We then assume the ad hoc agent to have access to a pre-built model library $M = \{\hat{M}_1^{\alpha}, \ldots, \hat{M}_K^{\alpha}\}$ and policy library $\Pi = \{\hat{\pi}_1^{\alpha}, \ldots, \hat{\pi}_K^{\alpha}\}$, where each policy $\hat{\pi}_k^{\alpha}$ is the solution to the corresponding POMDP $\hat{M}_k^{\alpha}$.

5.2.2 Algorithm

We adopt a Bayesian framework and treat the target task/teammate model as a random variable, M, taking values in the set of possible model descriptions, $M = \{m_1, \ldots, m_K\}$. For $\hat{M}_k^{\alpha} \in M$, let $p_0(m_k)$ denote the ad hoc agent's prior over M. Given a history h_t, we define

$$p_t(m_k) \triangleq \mathbb{P}\left[M = m_k \mid H_t = h_t\right], \qquad \hat{M}_k^{\alpha} \in \mathcal{M}. \tag{5.3}$$

The distribution p_t corresponds to the agent's belief about the target model at time step t. The action for the ad hoc agent at time step t can be computed within our

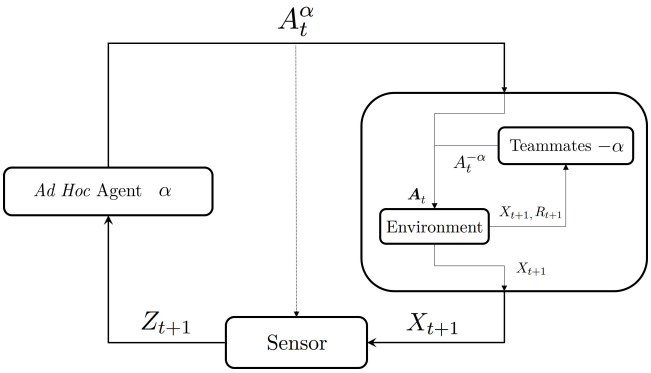


FIGURE 5.4: Interaction between an ad hoc agent α and teammates $-\alpha$, described by a POMDP $\hat{M}^\alpha$ with an underlying MDP M^α which reduces an MMDP M according to policy $\pi^{-\alpha}$ using Equation 2.2. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$, the reward signal R_{t+1} and the state of the environment X_t.

Bayesian setting as

$$\pi_t(a^\alpha \mid h_t) = \sum_{k=1}^{K} \hat{\pi}_k(a^\alpha \mid b_{k,t}) p_t(m_k),$$

where

$$b_{k,t}(x) \triangleq \mathbb{P}\left[X_t = x \mid H_t = h_t, M = m_k\right], \tag{5.4}$$

for $x \in \mathcal{X}_k$. Upon selecting an action a_t^α and making a new observation z_{t+1}, we can update p_t by noting that

$$p_{t+1}(m_k) = \frac{1}{\rho} \mathbb{P}\left[A_t^\alpha = a_t^\alpha, Z_{t+1} = z_{t+1} \mid M = m_k, H_t = h_t\right] p_t(m_k),$$

where ρ is some normalisation constant. Moreover,

$$\begin{aligned}
&\mathbb{P}\left[A_t^\alpha = a_t^\alpha, Z_{t+1} = z_{t+1} \mid M = m_k, H_t = h_t\right] \\
&= \sum_{y \in \mathcal{X}_k} O_k(z_{t+1} \mid y, a_t^\alpha) \mathbb{P}\left[X_{t+1} = y \mid A_t^\alpha = a_t^\alpha, H_t = h_t, M = m_k\right] \pi_t(a^\alpha \mid h_t),
\end{aligned}$$

where the last equality follows from the fact that the agent's action selection given the history does not depend on the model M. Therefore,

$$\begin{aligned}
&\mathbb{P}\left[A_t^\alpha = a_t^\alpha, Z_{t+1} = z_{t+1} \mid M = m_k, H_t = h_t\right] \\
&= \sum_{x,y \in \mathcal{X}_k} O_k(z_{t+1} \mid y, a_t^\alpha) P_k(y \mid x, a_t^\alpha) b_{k,t}(x) \pi_t(a^\alpha \mid h_t).
\end{aligned}$$

Putting everything together, we get

$$p_{t+1}(m_k) = \frac{1}{\rho} \sum_{x,y \in \mathcal{X}_k} O_k(z_{t+1} \mid y, a_t^\alpha) \cdot P_k(y \mid x, a_t^\alpha) b_{k,t}(x) \pi_t(a^\alpha \mid h_t) p_t(m_k), \tag{5.5}$$

with

$$\rho = \sum_{k=1}^{K} \sum_{x,y \in \mathcal{X}_k} O_k(z_{t+1} \mid y, a_t^\alpha) P_k(y \mid x, a_t^\alpha) b_{k,t}(x) \pi_t(a^\alpha \mid h_t) p_t(m_k).$$

Note that the update (5.5) requires the ad hoc agent to keep track of the beliefs $b_{k,t}$ for the different POMDPs $\hat{m}_k$. In other words, at each time step t, upon executing its

individual action a_t^α and observing z_{t+1}, the agent updates each belief $b_{k,t}$ using (2.3), yielding

$$b_{k,t+1}(y) = \frac{1}{\rho} \sum_{x \in \mathcal{X}_k} b_{t,k}(x) P_k(y \mid x, a_t^\alpha) O_k(z_{t+1} \mid y, a_t^\alpha), \tag{5.6}$$

where ρ is the corresponding normalisation constant. Since some of the computations in the update (5.5) are common to the update (5.6), some computational savings can be achieved by caching the intermediate values.

Finally, at each time step t, we define the *loss* for selecting an action $a^\alpha \in \mathcal{A}^\alpha$ when the target model is m_k to be

$$\ell_t(a^\alpha \mid m_k) = v^{\hat{\pi}_k}(b_{k,t}) - q^{\hat{\pi}_k}(b_{k,t}, a^\alpha), \tag{5.7}$$

where $\hat{\pi}_k$ is the solution to the POMDP $\hat{m}_k$.

It is important to note that both $v^{\hat{\pi}_k}(b_{k,t})$ and $q^{\hat{\pi}_k}(b_{k,t}, a^\alpha)$ can be computed from $\hat{v}^{\hat{\pi}_k}$ at runtime, while the latter can be computed offline when solving the POMDP $\hat{m}_k$. Note also that $\ell_t(a^\alpha \mid m_k) \geq 0$ for all a^α, and $\ell_t(a^\alpha \mid m_k) = 0$ only if $\hat{\pi}_k(a^\alpha \mid b_{k,t}) > 0$.

5.2.3 Loss bounds for ATPO

We conclude this section by providing a bound on the loss incurred by our approach. The bound can be derived from a standard compression lemma from Banerjee [2006], by comparing the performance of an agent using an arbitrary constant distribution over task/teammate combinations. Let p denote a distribution over $\mathcal{M}$, and define

$$L_t(p) = \sum_{k=1}^{K} p(m_k) \ell_t(\hat{\pi}_k \mid m^*),$$

where m^* is the (unknown) target task/teammate combination, and

$$\ell_t(\hat{\pi}_k \mid m^*) = \sum_{a^\alpha \in \mathcal{A}^\alpha} \hat{\pi}_k(a^\alpha \mid b_{k,t}) \ell_t(a^\alpha \mid m^*).$$

We have the following result.

Proposition 1. *Let q denote an arbitrary stationary distribution over the tasks in $\mathcal{M}$, and $\{p_t\}$ the sequence of beliefs over tasks generated by ATPO. Then,*

$$\sum_{t=0}^{T-1} L_t(p_t) \leq \sum_{t=0}^{T-1} L_t(q) + \sqrt{\frac{2}{T}} \sum_{t=0}^{T-1} \mathrm{KL}\left(q \| p_t\right) + \sqrt{\frac{T}{2}} \cdot \frac{R_{\max}^2}{(1-\gamma)^2}. \tag{5.8}$$

Proof. We use the following compression lemma from Banerjee [2006].

Lemma 2. *Given a set of hypothesis $\mathcal{H} = \{1, \ldots, H\}$, for any measurable function $\phi : \mathcal{H} \to \mathbb{R}$ and any distributions p and q on $\mathcal{H}$,*

$$\mathbb{E}_{h \sim q}\left[\phi(h)\right] - \log \mathbb{E}_{h \sim p}\left[\exp(\phi(h))\right] \leq \mathrm{KL}\left(q \| p\right). \tag{5.9}$$

We want to bound the loss incurred by our agent after T time steps. Before introducing our result, we require some auxiliary notation. Let m^* denote the (unknown) target task/teammate combination at time step t. The expected loss of our agent at

time step t is given by

$$\begin{aligned} L_t(\pi_t) &= \mathbb{E}\left[\ell_t(A^\alpha \mid m^*)\right] \\ &= \sum_{a^\alpha \in \mathcal{A}^\alpha} \pi_t(a^\alpha)\ell_t(a^\alpha \mid m^*) \\ &= \sum_{k=1}^{K} p_t(m_k) \sum_{a^\alpha \in \mathcal{A}^\alpha} \hat{\pi}_k(a^\alpha \mid b_{k,t})\ell_t(a^\alpha \mid m^*) \\ &= \sum_{k=1}^{K} p_t(m_k)\ell_t(\hat{\pi}_k \mid m^*), \end{aligned}$$

where, for compactness, we wrote

$$\ell_t(\hat{\pi}_k \mid m^*) = \sum_{a^\alpha \in \mathcal{A}^\alpha} \hat{\pi}_k(a^\alpha \mid b_{k,t})\ell_t(a^\alpha \mid m^*). \tag{5.10}$$

Let q denote an arbitrary distribution over $\mathcal{M}$, and define

$$L_t(q) = \sum_{k=1}^{K} q(m_k)\ell_t(\hat{\pi}_k \mid m^*). \tag{5.11}$$

Then, setting $\phi(m_k) = -\eta\ell_t(\hat{\pi}_{k_i} \mid m^*)$, for some $\eta > 0$, and using Lemma 2, we have that

$$\mathbb{E}_{m\sim q}\left[\phi(m)\right] - \log \mathbb{E}_{m\sim p_t}\left[\exp(\phi(m))\right] \leq \mathrm{KL}\left((\|q) \parallel p_t\right) \tag{5.12}$$

which is equivalent to

$$-\log \mathbb{E}_{m\sim p_t}\left[\exp(\phi(m))\right] \leq \eta L_t(q) + \mathrm{KL}\left((\|q) \parallel p_t\right). \tag{5.13}$$

Noting that $-2\eta\frac{R_{\max}}{1-\gamma} \leq \phi(m) \leq 0$ and using Hoeffding's Lemma,[2] we have that

$$-\log \mathbb{E}_{m\sim p_t}\left[\exp(\phi(m))\right] \geq \eta L_t(p_t) - \frac{\eta^2 R_{\max}^2}{2(1-\gamma)^2}. \tag{5.15}$$

Combining (5.13) and (5.15), yields

$$L_t(p_t) \leq L_t(q) + \frac{1}{\eta}\mathrm{KL}\left((\|q) \parallel p_t\right) + \frac{\eta R_{\max}^2}{2(1-\gamma)^2} \tag{5.16}$$

which, summing for all t, yields

$$\sum_{t=0}^{T-1} L_t(p_t) \leq \sum_{t=0}^{T-1} L_t(q) + \frac{1}{\eta}\sum_{t=0}^{T-1} \mathrm{KL}\left((\|q) \parallel p_t\right) + \frac{T\eta R_{\max}^2}{2(1-\gamma)^2}. \tag{5.17}$$

Since η can be selected arbitrarily, setting $\eta = \sqrt{\frac{T}{2}}$ we finally get

$$\sum_{t=0}^{T-1} L_t(p_t) \leq \sum_{t=0}^{T-1} L_t(q) + \sqrt{\frac{2}{T}}\sum_{t=0}^{T-1} \mathrm{KL}\left((\|q) \parallel p_t\right) + \sqrt{\frac{T}{2}} \cdot \frac{R_{\max}^2}{(1-\gamma)^2}. \tag{5.18}$$

[2] Hoeffding's lemma states that, given a real-valued random variable X, where $a \leq X \leq b$ almost surely,

$$\mathbb{E}\left[e^{\lambda X}\right] \leq \exp\left(\lambda\mathbb{E}\left[X\right] + \frac{\lambda^2(b-a)^2}{8}\right), \tag{5.14}$$

for any $\lambda \in \mathbb{R}$.

□

Unsurprisingly—and aside from the term $\sqrt{\frac{T}{2}} \cdot \frac{R_{\max}^2}{(1-\gamma)^2}$, which is independent of q and grows sub-linearly with T—the bound in (5.8) states that the difference between the performance of ATPO and that obtained using a constant distribution (for example, the distribution concentrated on m^*) is similar to those reported by Banerjee [2006] for Bayesian online prediction with bounded loss, noting that

$$\sum_{t=0}^{T-1} \mathrm{KL}\left(q \| p_t\right) = \mathrm{KL}\left(\boldsymbol{q} \| \boldsymbol{p}_{0:T-1}\right), \tag{5.19}$$

where we write $\boldsymbol{q}$ and $\boldsymbol{p}_{0:T-1}$ refer to distributions over sequences in $\mathcal{M}^T$.

5.2.4 Evaluation

We setup the ad hoc evaluation method from Stone et al. [2010a]. A single trial therefore consists on running an agent α_0 (representing the ad hoc agent) with an agent α_1 (representing the teammate) in an environment which is modeled as a POMDP. We then register the total accumulated reward in order to assess the overall team's performance. Since the assumption that there is no available reward signal nor visible teammate's action has never been explored, we compare our approach, ATPO, against five baselines (i.e., other agents which take on the role of α_0) – *Value Iteration* (an agent which is able to perfectly observe the state of the environment and knows the underlying MMDP's optimal policy), *Perseus* (an agent, which like ATPO, is only able to observe a partial observation of the environment, but unlike ATPO, knows the task it is performing and the teammate's behaviour), *Random-Picker*, an agent which, like ATPO, also has the library of models and Perseus solutions but selects one randomly in each step (allowing an evaluation of the similarity between possible models), *BOPA*, the current state-of-the-art which assumes full observability and *Random*, an agent which selects its action randomly in every step.

We then ask three main research questions:

1. Is ATPO effective in assisting the teammates in completing the tasks?
2. Is ATPO robust in scaling to POMDPs with large state spaces?
3. Is ATPO robust in scaling to large sets of POMDPs?

In order to answer all three research questions, we setup a total of 70 POMDPs from 11 different domains. Table 5.2 summarizes the characteristics of all POMDPs from each of the eleven domains. Some domains allow for different teammates (varying in POMDP's transition probabilities P, which incorporate the policy of the teammate π_1), others for different task goals (by varying the reward function r), and others for both. One domain (abandoned power plant) even allows for different state spaces between different tasks (varying both r and $\mathcal{X}$).

Given that we are evaluating a total of six agents in the role of α_0 and three of them (ATPO, Perseus and Random-Picker) require the approximate solutions to the POMDPs, we solve all 70 POMDPs from each domain using the Perseus algorithm [Spaan and Vlassis, 2005]. The times it takes to both setup and solve each POMDP can be found in Table 5.3.

5.2.5 Domain Descriptions

We now provide detailed descriptions of our three test scenarios.

TABLE 5.2: Multi-agent domains used for the ad hoc evaluation. π_1 represents whether the domain allows for varying the teammate's policy, r represents whether the domain allows for varying the target task/teammate combination, $|\mathcal{X}|$, $|\mathcal{A}|$ and $|\mathcal{Z}|$ represent the total number of states, actions and observations per POMDP from each domain (respectively), ϵ represents the amount of noise used in computing the transition and observation probabilities for each POMDP, h represents the number of steps in the trajectory horizon during which each POMDP is ran as an environment and, finally, $|\mathcal{M}|$ represents the total number of different setup POMDPs for each domain (which also corresponds to the size of our approach's model library).

Environment	π_1	r	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{Z}\rvert$	ϵ	h	$\lvert\mathcal{M}\rvert$
gridworld	non-varying	varying	626	5	81	0.20	50	$2 \rightarrow 32$
pursuit-task	non-varying	varying	626	5	81	0.20	75	4
pursuit-teammate	varying	non-varying	626	5	81	0.15	85	2
pursuit-both	varying	varying	626	5	81	0.15	85	8
abandoned power plant	non-varying	varying	varying (97, 105)	6	6	0.20	50	2
ntu	non-varying	varying	241	5	81	0.20	75	4
overcooked	varying	non-varying	1730	4	1730	0.00	50	4
isr	non-varying	varying	1807	5	81	0.20	75	3
mit	non-varying	varying	2163	5	81	0.20	75	3
pentagon	non-varying	varying	2653	5	81	0.20	75	3
cit	non-varying	varying	4831	5	81	0.10	85	3

TABLE 5.3: Times to setup and solve POMDPs from each domain. Setup times represent the total time it takes to create the data structures of the POMDP and solve times represent the total time it takes the Perseus algorithm to obtain a policy.

Environment	**Avg. Time to Setup**	(Std. Dev.)	**Avg. Time to Solve**	(Std. Dev.)
gridworld	27s 411ms	(±668ms)	41m 02s 934ms	(±41m 10s 857ms)
pursuit-task	03m 38s 023ms	(±02s 316ms)	2h 30m 54s 820ms	(±1h 03m 45s 600ms)
pursuit-teammate	03m 21s 236ms	(±112ms)	2h 31m 23s 526ms	(±14m 45s 856ms)
pursuit-both	03m 31s 369ms	(±03s 085ms)	3h 03m 02s 597ms	(±1h 16m 14s 350ms)
abandoned power plant	01s 406ms	(±136ms)	43s 322ms	(±19s 601ms)
ntu	14s 004ms	(±502ms)	10m 05s 246ms	(±01m 24s 147ms)
overcooked	04m 00s 960ms	(±19s 409ms)	05m 22s 543ms	(±01m 16s 879ms)
isr	08m 22s 774ms	(±19s 985ms)	6h 02m 58s 980ms	(±1h 15m 47s 677ms)
mit	11m 41s 286ms	(±52s 980ms)	5h 43m 18s 890ms	(±4h 20m 11s 880ms)
pentagon	16m 14s 605ms	(±13s 107ms)	4h 55m 49s 658ms	(±1h 31m 04s 993ms)
cit	51m 45s 357ms	(±58s 266ms)	11h 53m 30s 001ms	(±5h 06m 50s 992ms)

Gridworld

The gridworld domain (Fig. 5.5) —represents a standard multi-agent navigation problem. In the gridworld domain, N agents must reach N static destination (goal cells). Goal cells are fixed and not encoded in the state, only in the reward. Different possible configurations for the goals thus correspond to the different tasks in $\mathcal{M}$.

In this scenario, the ad hoc agent can move up, down, left and right, or stay in its current cell. The teammate follows the shortest path to its closest goal. Each moving action succeeds with probability $1 - \varepsilon$, for some $\varepsilon \geq 0$, except if there is a wall in the corresponding direction, in which case the position of the agent remains unchanged. When an action fails, the position of the agent remains unchanged.

The ad hoc agent can only observe the neighbouring cells. *It cannot observe its current position*. Observations are also not flawless: whenever an element (teammate, wall) is in a neighbouring cell, there is a probability ε that the agent will fail to observe it.

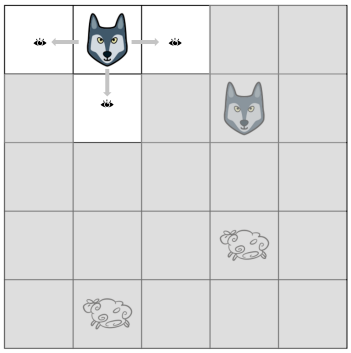


FIGURE 5.5: The gridworld domain. Two agents must each navigate to two goal cells.

We model each possible model $\hat{M}_k^\alpha \in \mathcal{M}$ as a POMDP

$$(\mathcal{X}, \mathcal{A}^\alpha, \mathcal{Z}, P_k^\alpha, O_k, r_k, \gamma).$$

A state $x \in \mathcal{X}$ contains information regarding the positions of both agents, and is therefore represented as a tuple $x = (c_1, r_1, c_2, r_2)$, where (c_n, r_n) represents the cell (column and row) where agent n is located.

Each observation $z \in \mathcal{Z}$ are also represented by a tuple $z = (\hat{u}, \hat{d}, \hat{l}, \hat{r})$, where each entry represents what is observed, respectively, above, below, to the left, and to the right of the agent. For each entry there are three possible values: *Nothing*, *Teammate*, *Wall*. The action space $\mathcal{A}^\alpha$ contains five possible actions, *Up, Down, Left, Right* and *Stay*. The transition probabilities $P^\alpha(y \mid x, a^\alpha)$ map a state x and action a^α to every possible next state y, taking into account the probability ϵ of the agent failing to move. Similarly, the observation probabilities $O^\alpha(z \mid y, a^\alpha)$ map a state y and previous action a^α to every an observation z, taking into account the probability ϵ of the agent failing to observe any given element. The reward function r_k assigns the reward of -1 for all time steps except those where both destination have been reached, in which case it assigns a reward of 100 (and an absorbent state assigned a reward of 0 afterwards). In other words, since where dealing with an horizon of 50 steps, we do not reset the environment whenever the goals have been reached. Finally, we consider a discount factor $\gamma = 0.95$.

Pursuit

The pursuit domain (Fig. 5.6) used in our work is a modification of the original Pursuit domain where we add partial state observability. In it, N agents ($N = 2$ in our experiments) must capture a single moving prey by surrounding it in a coordinated way.

In this scenario, the ad hoc agent is able to observe the elements located in its . Specifically, if either the teammate or the prey are located in one of the neighbouring nine cells surrounding the agent, it may be able to observe them with probability $1 - \epsilon$, where ϵ represents the probability of failing the observation.

We model each possible model $\hat{M}_k^\alpha \in \mathcal{M}$ as a POMDP

$$(\mathcal{X}, \mathcal{A}^\alpha, \mathcal{Z}, P_k^\alpha, O_k^\alpha, r_k, \gamma).$$

Different prey capture configurations vary the reward function r and different teammate policies vary the transition probabilities $\mathbf{P}$. We consider four different capture combinations — 1) *north of prey + south of prey*, 2) *west of prey + east of prey*, 3) *southwest*

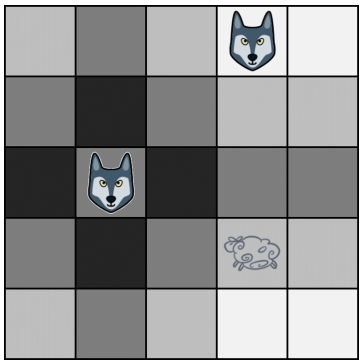


FIGURE 5.6: The pursuit domain. Two agents must capture a moving prey.

of prey + northeast of prey and 4) *northwest of prey + southeast of prey*. The capture positions may be used interchangeably between the agents (i.e., either the ad hoc agent or the teammate may occupy each of the two required positions). We also consider two possible teammate policies — 1) greedy policy and 2) teammate aware policy. The greedy policy follows the shortest path to its closest capture position, not taking into account the position of the ad hoc agent and therefore not acting as efficiently as possible, while the teammate aware policy runs an A search taking into account the position of the teammate. Each state $x \in \mathcal{X}$ contain information regarding the relative distances to the teammate and prey and is therefore represented by a tuple $x = (d^{a_1}{}_x, d^{a_1}{}_y, d^{p}{}_x, d^{p}{}_y)$, where $d^{a_1}{}_x, d^{a_1}{}_y$ represents the relative distance (in units) to the teammate and $d^{p}{}_x, d^{p}{}_y$ represents the relative distance (in units) to the prey. Each observation $z \in \mathcal{Z}$ is also represented as a tuple $z = (\hat{a_1}, \hat{p})$, where $\hat{a_1}$ represents an observation cell identifier of the teammate and $\hat{p}$ represents an observation cell identifier of the prey. A cell identifier is a value between $0 and 8$ which represents the nine surrounding cells of the agent. When an observation is failed or the teammate/prey is not in the nine surrounding cells, the ad hoc agent sees the other agent as it were standing in its own position. The action space $\mathcal{A}^{\alpha}$ contains five possible actions, *Up, Down, Left, Right, Stay*. For each teammate type k, the transition probabilities $P_k^{\alpha}(y \mid x, a^{\alpha})$ map a state x and action a^{α} to every possible next state y. Similarly, the observation probabilities $O^{\alpha}(z \mid y, a^{\alpha})$ map a state y and previous action a^{α} to every possible observation z, taking into account the probability $\epsilon(d)$ of the agent failing to observe the position of the other agents. The reward function r_k assigns the reward of -1 for all time steps except those where the prey has been cornered, in which case it assigns a reward of 100 for the one the prey was cornered in and 0 afterwards (in other words, since where dealing with a finite horizon, we do not reset the environment whenever the prey have been cornered). Finally, we consider a discount factor $\gamma = 0.95$.

Abandoned Power Plant

The abandoned power plant domain models a scenario where two agents — a robot and a human — must cooperate in order to secure an abandoned power plant with six rooms, where three are yet to be explored and two are filled with toxic waste. The ad hoc agent takes in the role of the robot and the teammate the role of the human. In this domain, there are two possible tasks which the human may be performing — charting the entire plant (exploring the three unexplored rooms) and securing the entire plant (by cleaning the two dirty rooms of all available toxic material).

Each task has different modelling requirements, and as such, even though both are modeled as POMDPs, they have different state spaces.

We model each possible model $\hat{M}_k^\alpha \in \mathcal{M}$ as a POMDP

$$(\mathcal{X}_k, \mathcal{A}^\alpha, \mathcal{Z}, P_k^\alpha, O_k^\alpha, r_k, \gamma).$$

POMDPs from both tasks vary in both state space $|\mathcal{X}|$ and reward function r. In POMDPs from the first task — exploration — each state $x \in \mathcal{X}$ is represented as a tuple $x = (n_{\text{robot}}, n_{\text{human}}, e_1, e_2, e_3)$ where n_{robot} and n_{human} represent the identifier of the room in which the robot and human are located (respectively) and e_i represents the status of room i (explored, unexplored). In POMDPs from the second task — cleanup — each state $x \in \mathcal{X}$ is represented as a tuple $x = (n_{\text{robot}}, n_{\text{human}}, d_1, d_2)$, where n_{robot} and n_{human} represent the identifier of the room in which the robot and human are located (respectively) and d_i represent the status of the room i (dirty, clean). The power plant is modeled as a topological graph with six nodes representing the rooms. The edges between nodes represent the connections between the rooms, and therefore, the action space for both the robot and the human contain four base actions — *Move to lowest-index node, Move to second lowest-index node, Move to third lowest-index node and Stay*. The robot has to itself available two additional actions — *Query human for human location and Query human for own location*. Observations $z \in \mathcal{Z}$ are represented by a a value — whenever the robot moves (instead of querying the human), it is able to observe, with a probability of failure ϵ, its own location, with z representing the id of the room it is in (and -1 when failing). Whenever the robot queries the human for the human's or its own location, the human may or may not reply, also with probability ϵ. If the query is successful, z contains the room of the human or the room of the robot (respectively according to the query type). If the query fails, $z = -1$. Finally, we consider a discount factor γ of 0.95 and the reward function r assigns a value of -1 the one where, in the case of exploration tasks, all unexplored rooms have been explored and, in the case of the cleanup tasks, all dirty rooms have been cleaned. Afterwards, an absorbent state is assigned a reward of 0.

NTU, ISR, MIT, PENTAGON and CIT

The ntu, isr, mit, pentagon and cit domains [Melo and Veloso, 2009, Hu et al., 2015] (Fig. 5.7) all model navigation tasks where two agents — α_0 and α_1 are spawned in random location of a map and must reach two destinations. Unlike the gridworld domain, which models an open gridworld, the ntu, isr, mit, pentagon and cit domains all model close quarters gridworlds, with walls and collision between the agents. The domains abide by the same base rules yet differ in terms of the map layout (with some being smaller and easier to navigate and others being larger and harder to navigate). They therefore have different state spaces, however, with states modeled the same way.

We model each possible model $\hat{M}_k^\alpha \in \mathcal{M}$ as a POMDP

$$(\mathcal{X}, \mathcal{A}^\alpha, \mathcal{Z}, P_k^\alpha, O_k^\alpha, r_k, \gamma).$$

Different tasks have different goal locations, which translate to different reward functions r. A state $x \in \mathcal{X}$ contains information regarding the positions of both agents, and is therefore represented as a tuple $x = (c_1, r_1, c_2, r_2)$, where (c_n, r_n) represents the cell (column and row) where agent n is located.

Each observation $z \in \mathcal{Z}$ are also represented by a tuple $z = (\hat{u}, \hat{d}, \hat{l}, \hat{r})$, where each entry represents what is observed, respectively, above, below, to the left, and to the

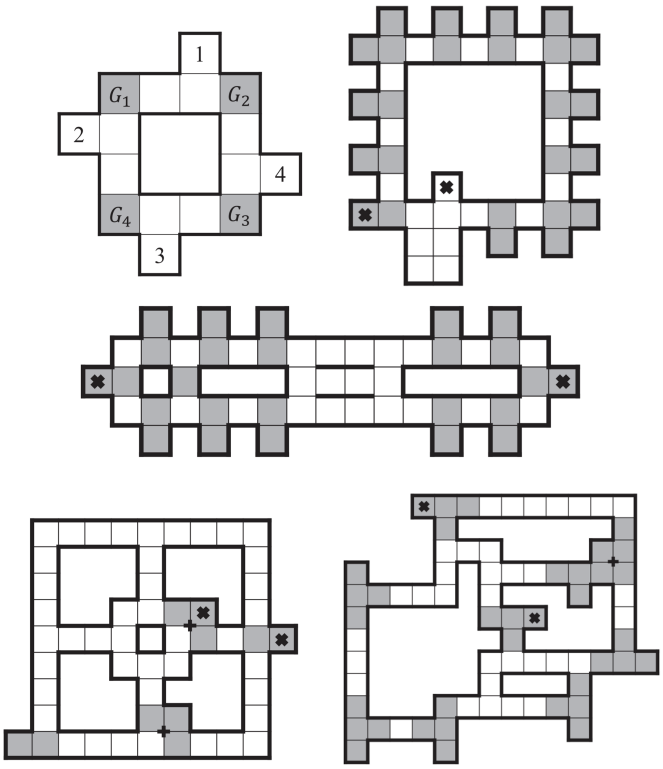


FIGURE 5.7: The ntu, isr, mit, pentagon and cit domains. Two agents must navigate in close quarters in order to each reach an exit (while taking into account collisions).

right of the agent. For each entry there are three possible values: *Nothing*, *Teammate*, *Wall*. The action space $\mathcal{A}^\alpha$ contains five possible actions, *Up, Down, Left, Right* and *Stay*. The transition probabilities $P^\alpha(y \mid x, a^\alpha)$ map a state x and action a^α to every possible next state y, taking into account the probability ϵ of the agent failing to move. Similarly, the observation probabilities $O^\alpha(z \mid y, a^\alpha)$ map a state y and previous action a^α to every an observation z, taking into account the probability ϵ of the agent failing to observe any given element. The reward function r_k assigns the reward of -1 for all time steps except those where both destination have been reached, in which case it assigns a reward of 100 (and an absorbent state assigned a reward of 0 afterwards). In other words, since where dealing with an horizon of 50 steps, we do not reset the environment whenever the goals have been reached. Finally, we consider a discount factor $\gamma = 0.95$.

Overcooked

The Overcooked domain [Carroll et al., 2019a] (Fig. 5.8) models a scenario where two agents, a helper and a cook, are required to cooperate in order to cook soups. In this case, tasks represent different teammates which the ad hoc agent must (i) identify; and (ii) adapt to, in order to cook soups. Our ad hoc agent plays the role of the *helper*, which has the goal of providing the cook with onions and plates, placing them on a kitchen counter and the teammate plays the role of the *cook*, which has the goal of cooking a soup using three onions and serving it in a plate (which is then dispatched through a window).

We model each possible model $\hat{M}_k^\alpha \in \mathcal{M}$ as a POMDP

$$(\mathcal{X}, \mathcal{A}^\alpha, \mathcal{Z}, P_k^\alpha, O_k^\alpha, r_k, \gamma).$$

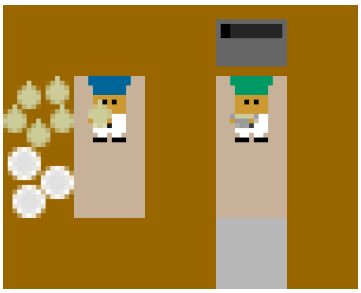


FIGURE 5.8: The overcooked domain. Two agents, a cook and an assistant, are required to deliver soups as fast as possible.

Varying the teammates varies the transition probabilities. Since the state is fully observable in this domain, observations correspond to the states themselves. Since the policy of the teammate may vary, each different model has distinct transition probabilities. We consider four different cook teammate policies — 1) a teammate which has the optimal policy, 2) a teammate which acts randomly, 3) a teammate which has the optimal policy but always stands near the bottom balcony when receiving ingredients and plates and 4) a teammate which has the optimal policy but always stands near the upper balcony when receiving ingredients and plates. Each state $x \in \mathcal{X}$ is represented as a tuple $x = (p_a, p_c, h_a, h_c, tb, bb, s)$, where, respectively, (p_a, p_c) represents the cell (top or bottom) and (h_a, h_c) represents the objects (nothing, onion, plate or soup) in hand of the helper and the cook. (tb, bb) represents the objects on the top kitchen counter balcony and bottom kitchen counter balcony (respectively). Finally, s represents the contents of the soup pan (empty, one onion, two onions, cooked soup). The action space $\mathcal{A}^{\alpha}$ contains four possible actions, *Up*, *Down*, *Noop* and *Act*. The reward function r assigns the reward of 100 to states x where the cook delivers a cooked soup through the kitchen window, and -1 otherwise. After the soup is delivered, the task is considered complete, leading to an absorbent state which is assigned a reward of 0. We consider a discount factor $\gamma = 0.95$.

5.2.6 Hyperparameters for the Perseus Algorithm

TABLE 5.4: Hyperparameters used for the Perseus [Spaan and Vlassis, 2005] algorithm. The discount factor was also used for the Value Iteration algorithm.

Environment	Horizon	Beliefs	Tolerance	Discount Factor
gridworld	50	5000	0.01	0.95
pursuit-task	75	5000	0.01	0.95
pursuit-teammate	85	5000	0.01	0.95
pursuit-both	85	5000	0.01	0.95
abandoned power plant	50	2500	0.01	0.95
ntu	75	5000	0.01	0.95
overcooked	50	1800	0.01	0.95
isr	75	5000	0.01	0.95
mit	75	5000	0.01	0.95
pentagon	75	5000	0.01	0.95
cit	85	8000	0.01	0.95

TABLE 5.5: Average accumulated reward for the six agents in the eleven domains over 32 trials (adding to a total of 1152 trials). BOPA requires a common state-space, so it cannot be run in the abandoned power plant scenario.

Environment	Value Iteration		Perseus		ATPO		Random-Picker		BOPA		Random	
gridworld	95.62	(±1.54)	93.44	(±3.13)	86.53	(±9.12)	39.91	(±61.34)	93.38	(±3.14)	5.88	(±64.13)
pursuit-both	90.19	(±8.32)	84.44	(±11.76)	82.72	(±12.81)	35.34	(±63.18)	90.06	(±7.47)	20.72	(±69.47)
pursuit-task	92.72	(±4.87)	88.03	(±7.51)	82.59	(±11.18)	47.59	(±55.46)	92.81	(±3.48)	14.59	(±71.34)
pursuit-teammate	89.88	(±8.38)	86.97	(±12.04)	86.41	(±14.52)	81.34	(±31.56)	91.38	(±6.13)	-7.66	(±78.7)
abandoned power plant	94.91	(±1.99)	95.12	(±2.1)	94.75	(±2.35)	93.81	(±4.97)	N.A.	N.A.	47.44	(±62.31)
ntu	98.00	(±0.0)	97.25	(±0.43)	97.19	(±0.53)	97.28	(±0.51)	98.0	(±0.0)	88.00	(±7.42)
overcooked	39.12	(±60.73)	39.00	(±60.63)	25.31	(±66.43)	27.94	(±64.58)	24.81	(±66.19)	-46.75	(±18.1)
isr	92.41	(±2.68)	91.16	(±4.68)	82.62	(±12.19)	60.59	(±59.15)	91.66	(±4.22)	-23.31	(±68.05)
mit	84.50	(±0.87)	84.09	(±0.58)	83.72	(±0.57)	83.75	(±0.75)	84.31	(±0.88)	-71.41	(±20.01)
pentagon	91.00	(±0.0)	90.62	(±0.6)	81.19	(±11.04)	53.28	(±56.53)	85.5	(±6.71)	-45.94	(±55.14)
cit	86.22	(±2.27)	85.09	(±2.53)	75.66	(±9.52)	69.91	(±14.46)	84.88	(±2.94)	-81.25	(±20.88)

5.2.7 Results

We then run, for all 11 environments, 32 independent trials for each of the six agents in the role of α_0. In each trial, we randomly select one of the possible POMDPs $m \in \mathcal{M}$ and run the interaction for a horizon of h timesteps (the same horizon used for the Perseus algorithm). We then compare the average accumulated reward over the h timesteps and 32 trials for all agents. Table 5.5 displays the obtained results.

At first glance, we may from these results conclude which domains are easier and which domains are harder. This can be seen by looking at the performance obtained by the Random Agent. Domains such as ntu and abandoned power plant, in which the Random Agent was able to achieve its highest possible results, are also the domains with the smallest state space. On the other hand, domains like cit and mit, where the Random Agent obtained its worst results, are also the domains with the highest number of possible states, showcasing that the problem difficulty is associated with the number of states in the POMDP.

From these results we are able to observe that all domains were effectively solved by the Perseus algorithm, providing a solid foundation for ATPO's library. In one domain (abandoned power plant, one of the easiest), the agent which knows the task and teammate behaviour, but is unable to observe the state of the environment and therefore resorts the Perseus policy (Perseus agent), was even able to, on average, slightly outperform the Value Iteration agent (although an insignificant difference).

Given that ATPO is required to identify, within its library $\mathcal{M}$, the POMDP which best models the environment (unlike the Perseus agent which already knows which one it is), we expected it to under perform the Perseus agent. In all domains this phenomenon was observed, further showcasing that our testbed is, in fact, a fair testbed for ATPO's evaluation.

We can also compare ATPO against the Random-Picker agent in order to identify which environments contain the most similar tasks and teammates. Given that the Random-Picker can be seen as an instance of ATPO with a constant uniform belief over the possible models, it is expected in most cases to under perform ATPO, since Random-Picker will most likely use the wrong policy. In domains where this doesn't happen, and Random-Picker is able to obtain similar results to ATPO, it means the tasks and teammates are very similar. As expected, ATPO outperforms Random-Picker in the different environments, showcasing the impact of the task/teammate combination identification ability of ATPO. In some scenarios, the difference is not very large (e.g., MIT, NTU), indicating that the task/teammate combination identification component is not critical to attain a good performance. However, in several other tasks, the difference is quite significant (e.g., Gridworld, Pentagon, Pursuit),

TABLE 5.6: ATPO's results from Table 5.5 normalised between the best possible performance (100%, obtained by the Value Iteration agent) and worst possible performance (0%, obtained by the Random Agent).

Environment	ATPO
gridworld	89.87%
pursuit-both	89.25%
pursuit-task	87.03%
pursuit-teammate	96.44%
abandoned power plant	99.66%
ntu	91.90%
overcooked	83.92%
isr	91.54%
mit	99.50%
pentagon	92.84%
cit	93.69%

and clearly shows the advantages of ATPO. As for BOPA, since the agent has full state observability, it is unsurprising that it attains better performance than ATPO - the difference attesting, to some extent, the impact that partial state observability can have in the ability of the agent to act.

Regarding the impact of the ATPO's Bayesian inference, our experiments already hint to the impact of model identification in the performance of ATPO. However, we should note that during our experiments we also keep track of the beliefs over tasks, as ATPO interacts with the environment. By analysing how the beliefs evolve over time, we can highlight a couple of more salient aspects. Specifically, if we look at the average probability associated to the correct model against that associated with the other tasks, we note that i) the correct task is identified, in average, after 5 timesteps, ii) in average, after 10 timesteps, the correct task has an associated probability of around 70% and iii) in average, after 20 timesteps, the correct task/teammate combination has an associated probability of around 90%. These values are not uniform across domains, but roughly indicate that the proposed approach is indeed able to effectively "zero-in" on the correct task/teammate combination. Analysing this same results in more detail, now considering how the beliefs evolve for each scenario individually, we can report that, in the most difficult domains, it takes around 10-15 timesteps for the correct task/teammate combination to reach the highest probability (becoming the most likely one), and by the end of the interaction, ATPO had near 100% certainty it was in fact the correct one. In the smaller and easier domains, it took only around 3-4 timesteps for the correct task/teammate combination to be the most likely one and, by the end of the interaction, ATPO also reached around 100% certainty that it was the correct one.

Moving on, we can treat the results obtained by the Random Agent as a lower bound to an agent's performance and the performance of the Value Iteration agent (which knows the task being performed, the behaviour of the teammate, and the current state of the environment) as an upper bound. If we normalise our results into a relative scale (where 100% corresponds to the performance of the Value Iteration agent and 0% corresponds to the performance of the Random agent) we are now able to more effectively evaluate our approach. Table 5.6 displays the same results from Table 5.5, but normalised.

Having now established a solid evaluation testbed, we can start looking at our

FIGURE 5.9: Varying the total number of possible POMDPs from the gridworld domain in ATPO's library. Results correspond to the relative performance, normalised between the best possible performance (100%, obtained by the Value Iteration agent) and worst possible performance (0%, obtained by the Random Agent).

approach's results (ATPO). Overall, ATPO was able to obtain above 83.92% performance on all domains (showcasing its effectiveness). Its worst results were on the overcooked domain (with 83.92% relative performance) and its best results were on the abandoned power plant and mit domains (with 99.66% and 99.50%, respectively). These results therefore showcase that our approach is effective in assisting unknown teammates in solving unknown tasks resorting only to a partial observation of the environment, and robust in adapting to domains with large state spaces.

5.2.8 Increasing ATPO's Library

We now evaluate ATPO's robustness in scaling to larger libraries of possible POMDPs (our third and final research question). We resort to a single domain — the gridworld domain — and conduct an experiment where we increase the number of POMDPs in ATPO's library $|\mathcal{M}|$ from 2 to 32. We then run, per each of the 31 library sizes, 32 independent trials on the environment (adding up to a total of 992 additional trials). Figure 5.9 reports our obtained relative results (normalised between the best possible performance and worst possible performance, taken from the results obtained in Table 5.5 by the Value Iteration and Random Agent, respectively).

As we can see from these results, as we increase the number of possible POMDPs in ATPO's library, its performance follows a small downward trend. This is expected, given that the more possible POMDPs there might be to explain the environment, the harder it will be to identify the correct one. Nevertheless, ATPO's performance with up to 32 possible models was always above 70%, strongly showcasing our approach's robustness in scaling to a higher number of possible models.

5.3 Conclusion

In this chapter we addressed our second research sub-problem:

> *How can an autonomous agent adapt to unknown teams and tasks being performed without being able to observe their actions and the state of the environment?*

We expand upon chapter 4 by dropping the assumptions that the ad hoc agent is able to fully observe the state of the environment, the actions of the teammates and

may use an explicit reward signal provided by the environment. We introduce two novel approaches for ad hoc teamwork under partial observability which, relying on a library of prior knowledge, are able to infer the correct teammate behaviour and tasks being performed resorting to Bayesian online prediction algorithm: (i) BOPA, which assumes there is a way to model and observe the state of the environment and (ii) ATPO, which relies only on a partial observation of the state of the environment.

Our empirical evaluation in several domains successfully showcased that it is possible to adapt to different teams performing different tasks, under conditions from real-world scenarios that may involve robots and humans as teammates.

◇

In the next chapter, we expand upon ATPO, proposing and evaluating a novel approach for ad hoc teamwork under partial observability that unlike ATPO is able to handle domains with large observation spaces or state spaces.

Chapter 6

Ad Hoc Teamwork in Large Partially Observable Domains

This chapter addresses our third research sub-problem:

> *How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?*

In chapter 5, we have introduced and evaluated ATPO, an approach capable of performing ad hoc teamwork under conditions of partial observability where no actions of the team were visible and no reward signal was available to adapt on-the-fly.

ATPO was able to perform ad hoc teamwork under these conditions by modelling each possible team/task combination as a distinct POMDP. Combined with the agent's observations, observation and transition probabilities could be used to perform Bayesian inference and compute the most-likely POMDP. Afterwards, ATPO then accessed a pre-trained PERSEUS [Spaan and Vlassis, 2005] policy to compute its best-responses. This method was possible since the state and observation spaces were finite, allowing the usage of these tabular methods. However, if the environments have larger or continuous state/observation spaces, such as high-dimensional ones which require function approximation techniques such as Convolutional Networks [LeCun et al., 2015] to extract latent representations, both the Bayesian inference performed by ATPO, as well as its PERSEUS policies cannot be employed.

Taking this limitation into account, in this chapter we explore how an ad hoc agent can perform ad hoc teamwork in domains with arbitrarily large state and observation spaces. We expand upon ATPO and present a novel approach, named ATLPO [1], and conduct an empirical evaluation in two domains where the observations are represented by 2D matrices. We show that by relying on state-of-the-art Deep Learning methods for representation learning, both a Bayesian Classifier and a set of pre-trained policies can be trained and employed in the same ad hoc setting under partial observability.

6.1 Ad hoc Teamwork in Large Partially Observability domains (ATLPO)

We drop the assumptions made by BOPA and ATPO (chapter 5) that the environments have tabular state and observation spaces. We hold the same assumptions that the agent is unable to observe the actions of the teammates and has an explicit reward

[1] ATLPO stand for Ad hoc Teamwork in Large Partially Observable domains

signal which could be used to learn how to interact with teams on-the-fly during the ad hoc evaluation phase.

6.1.1 Preliminaries

Similar to in chapter 5, we assume the same problem formulation from section 5.1.1, assuming the ad hoc agent α is, at each step t unable to observe the actions of the teammates $A_t^{-\alpha}$ and the reward signal R_{t+1}. Furthermore, we assume the ad hoc agent α is only able to obtain a partial observation Z_t described by an arbitrarily large or continuous feature function $\phi(z)$, unlike in chapter 5 where the tabular observation could be obtained. We still assume the environment is described by an underlying POMDP $\hat{M}^\alpha = (\mathcal{X}, \mathcal{A}^\alpha, \mathcal{Z}, P^\alpha, O^\alpha, r, \gamma)$. Figure 6.1 illustrates the interaction between the ad hoc agent and the teammates in the environment, described by a POMDP $\hat{M}^\alpha$ where the ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$, the reward signal R_{t+1} and the state of the environment X_t and a tabular representation of the observation Z_t

We then assume the ad hoc agent to have access to a pre-built model library $M = \{\hat{M}_1^\alpha, \ldots, \hat{M}_K^\alpha\}$ and policy library $\Pi = \{\hat{\pi}_1^\alpha, \ldots, \hat{\pi}_K^\alpha\}$, where each policy $\hat{\pi}_k^\alpha$ consist on a pre-trained policy for the corresponding POMDP $\hat{M}_k^\alpha$.

6.1.2 Algorithm

We adopt the Bayesian framework from chapter 5, treating the target task/teammate model as a random variable M taking values in the set of possible model descriptions $M = \{m_1, \ldots, m_K\}$. For $\hat{M}_k^\alpha \in M$, let $p_0(m_k)$ denote the ad hoc agent's prior over M. Given a history h_t, we keep the definition

$$p_t(m_k) \triangleq \mathbb{P}\left[M = m_k \mid H_t = h_t\right], \qquad \hat{M}_k^\alpha \in \mathcal{M}. \tag{6.1}$$

Where the distribution p_t corresponds to the agent's belief about the target model at time step t.

Taking into account the Eq. 5.5 from chapter 5,

$$p_{t+1}(m_k) = \frac{1}{\rho} \sum_{x,y \in \mathcal{X}_k} O_k(z_{t+1} \mid y, a_t^\alpha) \cdot P_k(y \mid x, a_t^\alpha) b_{k,t}(x) \pi_t(a^\alpha \mid h_t) p_t(m_k),$$

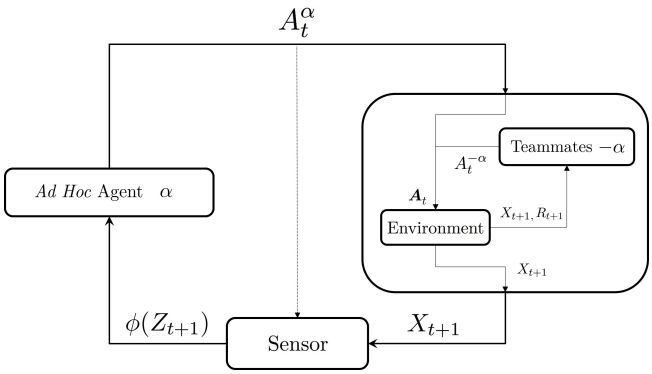


FIGURE 6.1: Interaction between an ad hoc agent α and teammates $-\alpha$, described by a POMDP $\hat{M}^\alpha$ with an underlying MDP M^α which reduces an MMDP M according to policy $\pi^{-\alpha}$ using Equation 2.2. The ad hoc agent is unable to observe the actions of the teammates $A_t^{-\alpha}$, the reward signal R_{t+1} and the state of the environment X_t.

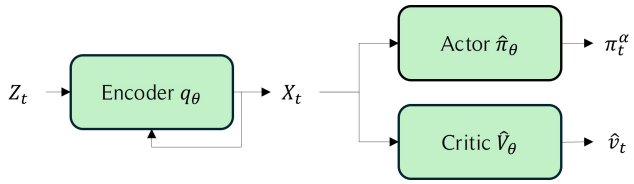


FIGURE 6.2: Actor-Critic architecture used for each policy in ATLPO-MF's library. An Encoder q, Actor $\hat{\pi}$ and Critic $\hat{V}$ are trained end-to-end using using proximal policy optimisation [Schulman et al., 2017].

where

$$\rho = \sum_{k=1}^{K} \sum_{x,y \in \mathcal{X}_k} O_k(z_{t+1} \mid y, a_t^\alpha) P_k(y \mid x, a_t^\alpha) b_{k,t}(x) \pi_t(a^\alpha \mid h_t) p_t(m_k).$$

this Bayesian update can no longer be performed without z_{t+1}. We now compute the update to p as a distribution over $\mathcal{M}$, given $\phi(Z_{t+1})$ and A_t^α relying on a recurrent function approximator $\hat{p}$ parametrised by θ_p, i.e.

$$\boldsymbol{p}_{t+1}, h_{t+1} = \frac{1}{\rho} \hat{p}(\phi(z_{t+1}), a_t^\alpha, h_t; \theta_p), \tag{6.2}$$

where ρ is a normalisation constant and h_t a latent history variable recurrently kept by $\hat{p}$.

Finally, similar to ATPO (chapter 5), we rely on a policy library $\Pi = \{\hat{\pi}_1^\alpha, \ldots, \hat{\pi}_K^\alpha\}$, where each policy $\hat{\pi}_k^\alpha$ is a pre-trained policy for POMDP $\hat{M}_k^\alpha$.

6.1.3 Choice of Policy Library

Following our findings from chapter 4 that model-based reinforcement learning methods might more efficiently learn new policies, we decide to present and evaluate two variants of ATLPO when it comes to the set up of its policy library Π. Instead of exact solvers such as PERSEUS [Spaan and Vlassis, 2005], we now rely on either (i) a pre-trained library of model-free reinforcement learning approaches or (ii) a pre-trained library of model-free reinforcement learning approaches. We name these two approaches ATLPO-MF and ATLPO-MB, respectively. We expect the quality of each approach, ATLPO-MF and ATLPO-MB to scale according to the quality of the chosen reinforcement learning algorithms. In other words, if the current state-of-the-art in model-based reinforcement learning outperforms the current state-of-the-art in model-free reinforcement learning, we expect ATLPO-MB to outperform ATLPO-MF.

6.1.4 ATLPO-MF

ATLPO-MF, as the name suggests, relies on a policy library $\Pi = \{\hat{\pi}_1^\alpha, \ldots, \hat{\pi}_K^\alpha\}$ where each $\hat{\pi}_k^\alpha$ corresponds to a pre-trained reinforcement learning model-free policy. In this work, we follow the standard actor-critic architecture trained end-to-end with an Encoder q through proximal policy optimisation [Schulman et al., 2017]. Given an observation Z_t each policy $\hat{\pi}_k^\alpha$ outputs (i) the policy distribution π_t^k, (ii) a value v_t^k. Figure 6.2 showcases an overview of the architecture.

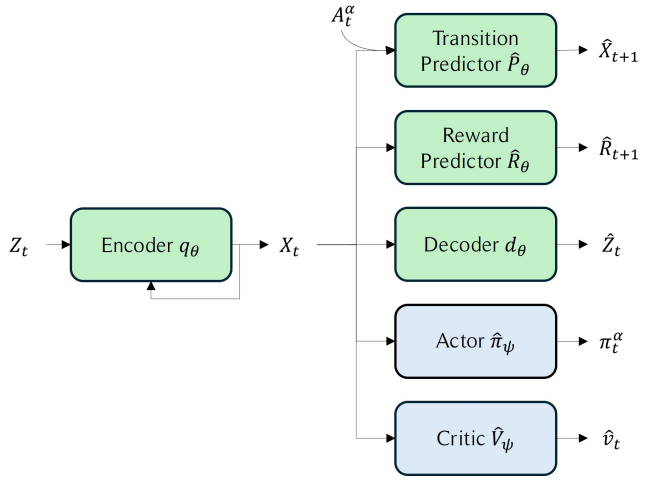


FIGURE 6.3: Architecture used for each policy in ATLPO-MB's library. An Encoder q, Transition Predictor $\hat{P}$, Reward Predictor $\hat{R}$ and Decoder d are trained end-to-end, computing latent state representations used to train the Actor $\hat{\pi}$ and Critic $\hat{V}$ independently as a standard fully observable reinforcement learning problem.

6.1.5 ATLPO-MB

ATLPO-MB, as the name suggests, relies on a policy library $\Pi = \{\hat{\pi}_1^\alpha, \ldots, \hat{\pi}_K^\alpha\}$ where each $\hat{\pi}_k^\alpha$ corresponds to a pre-trained reinforcement learning model-based policy. In this work, we follow recent advances in model-based reinforcement learning and combine elements from approaches such as MuZero [Schrittwieser et al., 2020a] and DreamerV3 [Hafner et al., 2023b].

Similar to ATLPO-MF, each policy on ATLPO-MB's library also returns both a policy distribution π_t^k and a value prediction v_t^k, using an Actor and Critic. However, unlike with ATLPO-MF, the Actor and Critic are not trained end-to-end alongside the Encoder. Instead, the Encoder returns a latent state representation $\hat{X}$ which is then passed through three tails — (i) a Decoder which reconstructs the original observation, (ii) a transition predictor which predicts a possible next latent state $\hat{X}_{t+1}$ given an action to execute and (iii) a reward predictor which predicts the next possible reward $\hat{R}_{t+1}$. These components are trained end-to-end, and the latent state space used to train the Actor and Critic using proximal policy optimisation as a standard fully observable reinforcement learning problem. Figure 6.3 showcases an overview of the architecture.

In chapter 4, through TEAMSTER, we were able to show that when possible to learn a shared model of the dynamics and goal, this model can be used throughout different teams. However, according now to Eq. 6.2, the world's dynamics, i.e. transition probabilities $P(y \mid x, a)$ are now coupled with the policies of the teammates $\pi^{-\alpha}$, requiring an individual policy to be trained. We leave for future work the possibility of further exploring this limitation by employing techniques from fields such as transfer learning or meta-learning.

6.2 Evaluation

We adopt the same evaluation procedure from chapter 5. For an evaluation domain, we consider three independent evaluations, each for a specific ad hoc set of K possible environments, each environment with a given task and team policy. In the first set we consider the team policy to be fixed and the task to be variable, leaving the goal

for the ad hoc agent to identify the correct task on-the-fly. In the second set we fix the task and vary the team policy, leaving the goal for the ad hoc agent to identify the correct team. In the third and final set, we vary both the task and the team policy, leaving the goal for the ad hoc agent to identify both.

Our evaluation aims to answer three main research questions:

1. Are ATLPO-MF and ATLPO-MB effective in identifying the correct team and task being performed?
2. Are ATLPO-MF and ATLPO-MB effective in assisting the team in completing the tasks?

In order to answer all three research questions, we set up the three sets of environments and allowing the agents a pre-training stage similar to the one where ATPO trained its PERSEUS policies. In this stage, both ATLPO-MF and ATLPO-MB train an individual policy for each environment and, after fixing these policies, an additional Bayesian Classifier which identifies the most likely task and/or team policy from observations alone.

6.2.1 Domains

We rely on two benchmark domains from the current ad hoc teamwork literature — the *Level-Based Foraging* domain and *Predator-Prey* domain.

Level-Based Foraging

The Level-Based Foraging domain [Christianos et al., 2020] describes environments where team of foraging agents have the goal of pickup up all apples in a field. Each agent and apple have their own levels, and an apple can only be foraged if the sum of the levels of its foraging agents is greater than or equal to its own. An episode ends when all apples have been successfully foraged.

State Space The foraging field of the Level-Based Foraging domain is modelled as a $(R \times C)$ grid, where R refers to the total number of rows and C the total number of columns. Therefore, for N agents and M apples, a state $x \in \mathcal{X}$ is described as a tuple:

$$x = ((r_{\alpha_0}, c_{\alpha_0}, l_{\alpha_0}), (r_{\alpha_1}, c_{\alpha_1}, l_{\alpha_1}), \ldots, (r_{\alpha_N}, c_{\alpha_N}, l_{\alpha_N}), \\ (r_{a_0}, c_{a_0}, l_{a_0}), (r_{a_1}, c_{a_1}, l_{a_1}), \ldots, (r_{a_M}, c_{a_M}, l_{a_M}))$$

where $(r_{\alpha_i}, c_{\alpha_i}, l_{\alpha_i})$ represents the row, column and level for agent α_i and $(r_{a_j}, c_{a_j}, l_{a_j})$ represents the row, column and level for apple a_j. For both agents and apples, rows $r \in [0, R-1]$, columns $c \in [0, C-1]$ and levels $l \in [0, \infty[$.

Action Space Each forager agent has five actions to choose from in a given timestep — *Up*, *Down*, *Left*, *Right* and *Forage*. Each movement action, *Up*, *Down*, *Left* and *Right*, moves the agent to the adjacent cell if the cell is empty. If the cell is occupied, the action fails and the forager agent stays within the same cell. If a forager agent moves towards the borders of the grid, the toroidal nature of the environment transports it to the opposite side. Finally, if all foragers surrounds an apple, they must execute the *Forage* action in the same timestep to effectively forage it.

Observation Space To conduct our evaluation, we require a partially observable domain. For an agent with partial observability, we set it a field-of-view grid with f rows and f columns, centred on its position and limiting its perception of the full state of the environment. We then setup a different channel for the detection of each teammate (total of $N-1$ channels), detection of each apple (total of M channels), detection of grid boundaries (single channel) and detection of levels for self, teammates and apples (single channel). Each teammate and each apple has its own individual channel given that some agent decision policies take into account its identifiers. In other words, the identifier of each teammate and apple may be required to make decisions and combining them all in a single channel would mean a loss of information. An observation $z \in \mathcal{Z}$ is therefore represented by tuple containing a total of $(N-1)+M+2$ channels, each a matrix of shape $(f \times f)$. Figures 6.4 showcases an example of the observation for two different states.

Reward Space We consider a sparse reward setting, with $R_t = 1.0$ if all apples have been foraged and $R_t = 0.0$ otherwise.

Predator-Prey

The Predator-Prey, or Pursuit domain [Barrett and Stone, 2015, Barrett et al., 2017, Ribeiro et al., 2023b,a], used previously in the evaluations of TEAMSTER (chapter 4) and ATPO (chapter 5), describes environments where team of up to four predator agents has the goal of capturing moving preys in the field. Whenever all predators are within the vicinity of the prey, an episode ends and the prey is considered captured.

State Space Similar to the Level-Based Foraging domain, the field of the Predator-Prey domain is modelled as a $(R \times C)$ grid, where R refers to the total number of rows and C the total number of columns. Therefore, for $N \leq 4$ agents and M preys, a state $x \in \mathcal{X}$ is described as a tuple:

$$x = ((r_{\alpha_0}, c_{\alpha_0}), (r_{\alpha_1}, c_{\alpha_1}), \ldots, (r_{\alpha_N}, c_{\alpha_N}), \\ (r_{p_0}, c_{p_0}), (r_{p_1}, c_{p_1}), \ldots, (r_{p_M}, c_{p_M}))$$

where $(r_{\alpha_i}, c_{\alpha_i})$ represents the row and column for predator agent α_i and (r_{p_j}, c_{p_j}) represents the row and column for prey p_j. For both predators and preys, rows $r \in [0, R-1]$ and columns $c \in [0, C-1]$.

Action Space Each predator has five actions to choose from in a given timestep — *Up*, *Down*, *Left*, *Right* and *Capture*. Each movement action, *Up*, *Down*, *Left* and *Right*, move the predator to the adjacent cell if the cell is empty. If the cell is occupied, the action fails and the predator stays within the same cell. If a predator moves towards the borders of the grid, the toroidal nature of the environment places the agent in the opposite side. Finally, if a predator surrounds the prey alongside its teammates, all predators must execute the *Capture* action to effectively capture the prey.

Observation Space We rely on the same observation space from the Level-Based Foraging domain, removing the levels channel. For a predator agent with partial observability, we set it a field-of-view grid with f rows and f columns, centred on its position and limiting its perception of the full state of the environment. We then

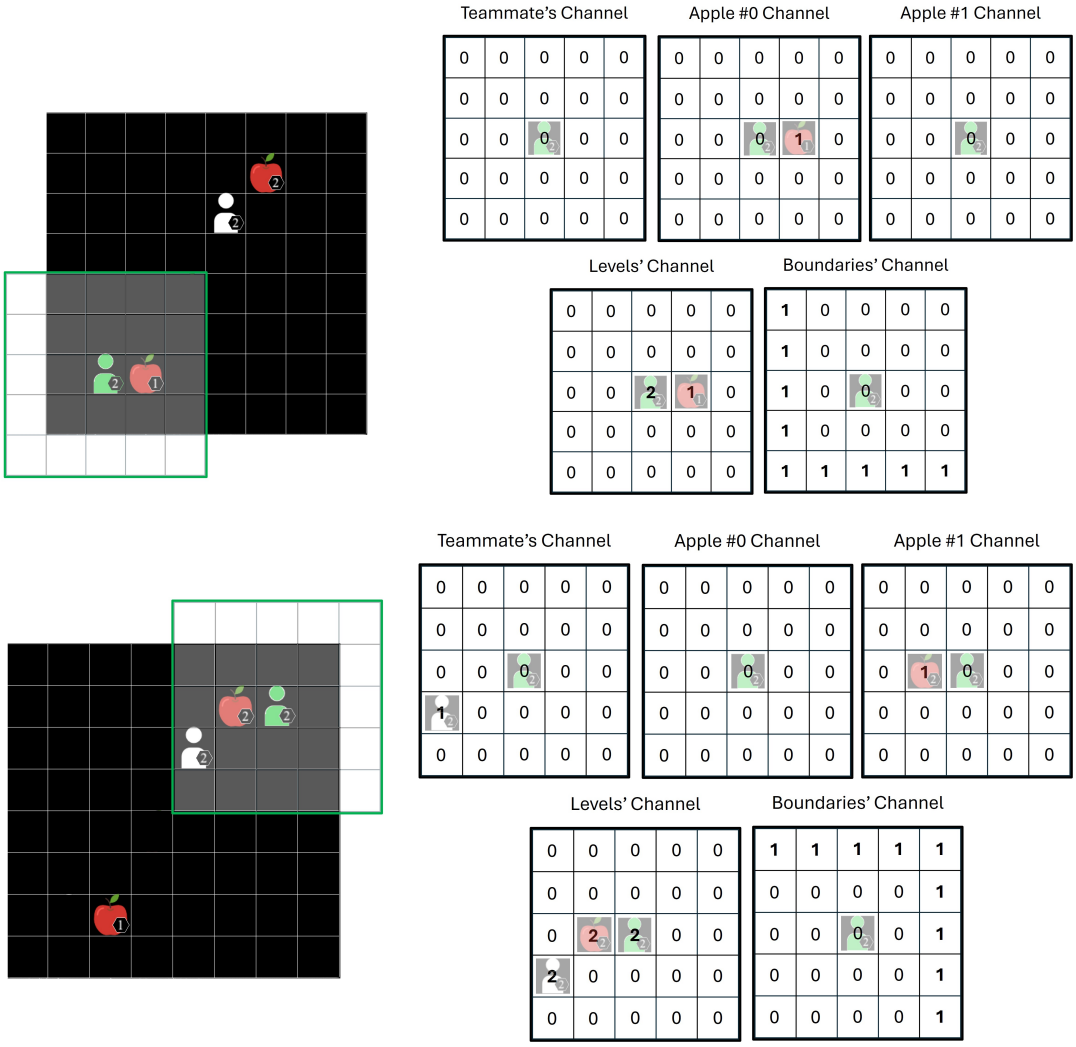


FIGURE 6.4: Observations for the Level-Based Foraging domain for two example states. A total of $C = (N-1) + M + 2$ channels, each a matrix of shape $(f \times f)$ representing a specific piece of information, with $f = 5$ throughout our experiments. A total of $N - 1$ channels indicate the relative positions of each teammate, $N = 2$ above, A channels indicate the relative positions of each apple, $A = 2$ above, one channel indicates the levels of agents and apples within the field-of-view and another channel indicates the boundaries of the world. Each apple and teammate are given their own individual channel, since strategies in this domain sometimes use their indexes for resolving stalemates and without an individual channel there would be no way to distinguish different apples and teammates.

setup a different channel for the detection of each teammate (total of $N - 1$ channels), detection of each prey (total of M channels) and detection of grid boundaries (single channel). An observation $z \in \mathcal{Z}$ is therefore represented by tuple containing a total of $(N-1) + M + 1$ channels, each a matrix of shape $(f \times f)$.

Reward Space We consider a sparse reward setting, with $R_t = 1.0$ if all preys have been captured and $R_t = 0.0$ otherwise.

6.2.2 Tasks & Teams

In both domains, we consider four different tasks and three distinct teams.

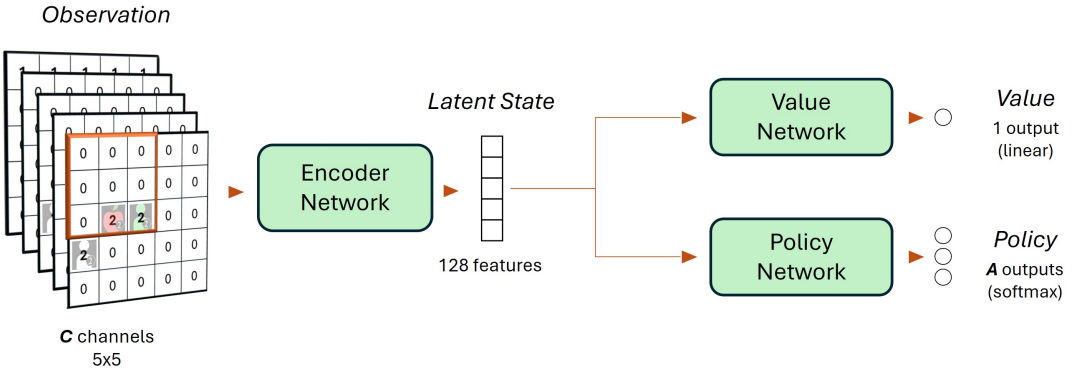


FIGURE 6.5: Implementation each policy's architecture from ATLPO-MF's library.

Tasks

Each task in both domains is associated with the relative positions of the four agents, foragers in the case of the Level-Based Foraging domain and predators in the case of the Predator-Prey domain, when foraging an apple or capturing a prey, respectively. For instance, in the first task, the first forager needs to forage the apple from the north of it, the second from the south, the third from west and the forth from the east. The remaining three tasks rotate these forage locations.

Teams

Teams from both domains can have one of three different team policies. The policy of the Original Teammates in the first team τ_0 finds the closest apple/prey and moves towards it without taking into account obstacles in the field. The policy of the Original Teammates in the second team τ_1 looks for the apple/prey (in the case of Level-Based Foraging it selects the apple with the highest level) and moves towards it assuming the teammates to do the same. This policy takes into account obstacles in the field to avoid collisions. Finally, the policy of the Original Teammates in the third team τ_2 searches for the positions of its teammates and computes a plan which encircles the target apple/prey to avoid collisions, closing in the circle in each timestep.

6.2.3 Policy Implementations

ATLPO-MF's Policies Each policy in ATLPO-MF's library has three components — (i) an Encoder q, (ii) an Actor $\hat{\pi}$ and (vi) a Critic $\hat{V}$. We implement the Encoder q as a Recurrent Convolutional Network and the Actor $\hat{\pi}$ and Critic $\hat{V}$ as two feed-forward multi-layer perceptrons (MLPs). Figure 6.5 showcases an overview of the implemented architecture for a policy in ATLPO-MF's library. The whole model is trained end-to-end using proximal policy optimisation [Schulman et al., 2017].

ATLPO-MB's Policies Each policy in ATLPO-MB's library has a total of six components — (i) an Encoder q, (ii) a Transition Predictor $\hat{P}$, (iii) a Reward Predictor $\hat{R}$, (iv) a Decoder d, (v) an Actor $\hat{\pi}$ and (vi) a Critic $\hat{V}$. We implement the Encoder q as a Recurrent Convolutional Network, the Decoder d as a Transposed Convolutional Network, and the remaining components as feedforward multi-layer perceptrons (MLPs). Figure 6.6 showcases an overview of the architecture for a policy in ATLPO-MB's library.

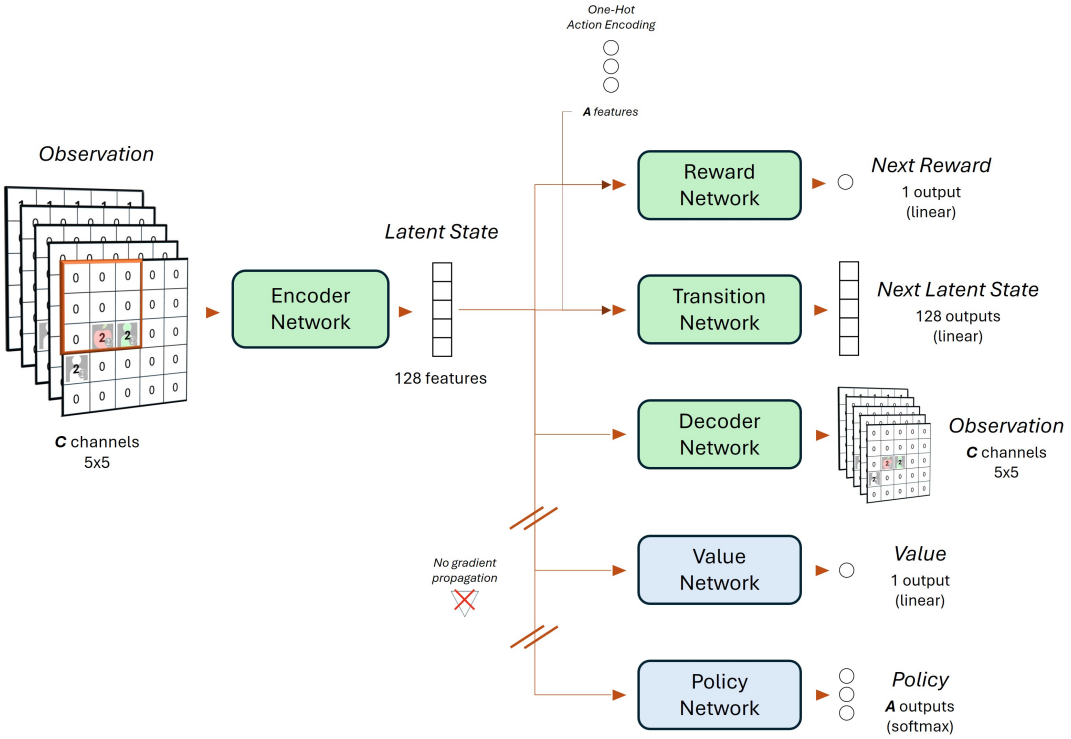


FIGURE 6.6: Implementation each policy's architecture from ATLPO-MB's library.

◇

We now discuss each component individually.

Encoder Network q

The goal of the Encoder Network q is to take as input observations $Z_t \in \mathbb{R}^{(C\times5\times5)}$, where C refers to the number of channels, variable for each domain, and extract a latent state representation $\hat{X}_t \in \mathbb{R}^{128}$. Throughout our experiments we fix a latent state dimension of 128 features, but also leave this choice as an hyperparameter. Given the partial observability of the environment, we implement q as a Recurrent Convolutional Network. Figure 6.7 showcases its architecture, used by both ATLPO-MF and ATLPO-MB.

Decoder Network d

The Decoder Network d, only used by ATLPO-MB, has the goal of reconstructing the original observation $Z_t \in \mathbb{R}^{(C\times5\times5)}$ from the latent state $\hat{X}_t \in \mathbb{R}^{128}$. To achieve this reconstruction, the latent state $\hat{x}$ goes first through a single Linear layer which projects its dimension to 64 features, afterwards passed through two 2D Transposed Convolutional Layers with similar characteristics to the 2D Convolutional Layers used to extract the latent state. Figure 6.8 showcases the architecture of the Decoder Network d. We use a smooth $l1$ loss as the loss component l_d for the Decoder Network, added to the loss components $l_{\hat{p}}$ and $l_{\hat{R}}$ of the Transition and Reward Networks, minimised by an Adam Optimiser with a learning rate of 0.001.

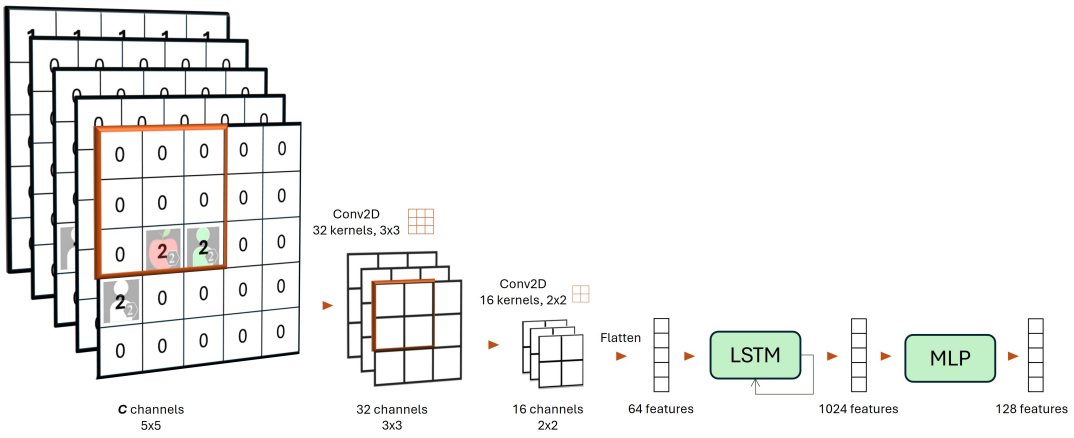


FIGURE 6.7: Encoder Network q used to extract latent states $\hat{X}_t$ from observations Z_t. Observations $Z_t \in \mathbb{R}^{(C\times5\times5)}$ are passed through two 2D Convolutional layers, one LSTM layer and an MLP, computing a latent state $\hat{X}_t \in \mathbb{R}^{128}$.

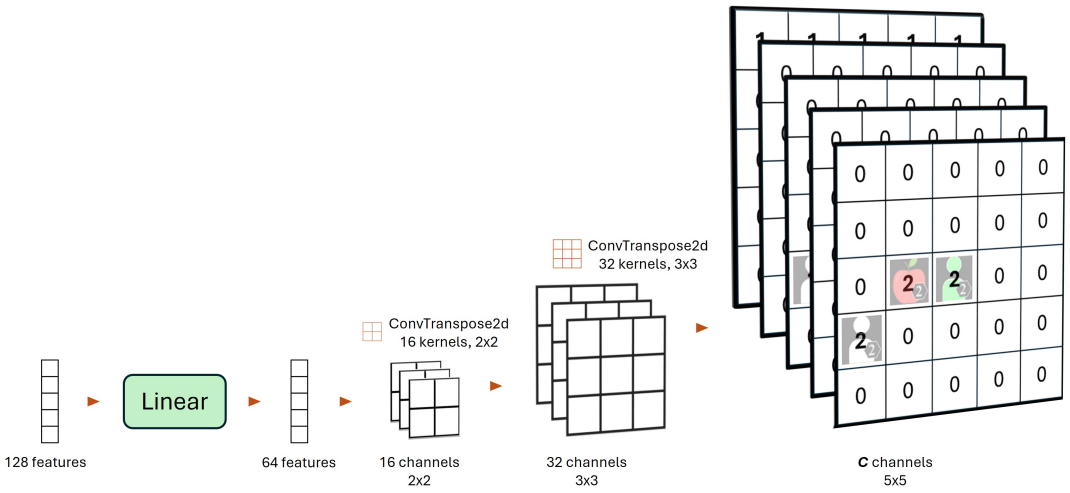


FIGURE 6.8: Decoder network d used to reconstruct observations Z_t from latent states $\hat{X}_T$. Latent states $\hat{X}_t$ are passed through a single Linear layer, reducing it to 64 features and afterwards through two 2D Transposed Convolutional Layers, reconstructing the observation $\hat{Z}_t \in \mathbb{R}^{(C\times5\times5)}$.

Transition and Reward Networks $\hat{P}$, $\hat{R}$

The Transition and Reward Networks $\hat{P}$, $\hat{R}$, used only by ATLPO-MB, have the goal of predicting transitions and rewards directly from the latent state space. Both receive as input a latent state $\hat{X}_t$ and one-hot encoding representation of the action, $\mathbf{e}_{A_t^\alpha}$ and output, respectively, a next predicted latent state $\hat{x}_{t+1}$ and next predicted reward $\hat{r}_{t+1}$. Both networks take as input the concatenated vector $(X_t \oplus \mathbf{e}_{A_t^\alpha}) \in \mathbb{R}^{128+A}$ and each rely on an MLPs to output $\hat{x}_{t+1}$ and $\hat{r}_{t+1}$, respectively. We use a smooth $l1$ loss as the loss component $l_{\hat{P}}$ for the Transition Network and a mean-squared error loss as the loss component $l_{\hat{R}}$ of the Reward Network [2], both added to the loss component l_d of the Decoder Network, minimised by an Adam Optimiser with a learning rate of 0.001.

[2]Given the sparse reward space of both domains, we employ a modification, converting the linear regression problem of learning the reward into a binary classification problem, changing the loss to a negative log likelihood.

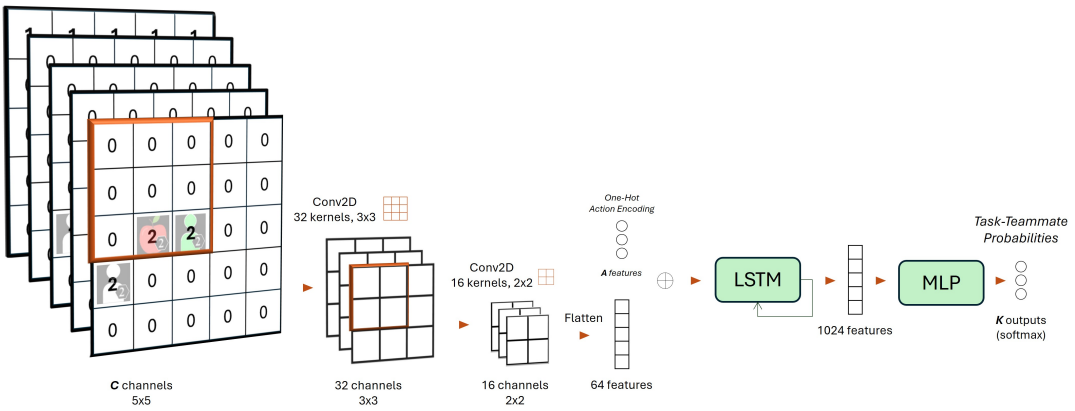


FIGURE 6.9: Recurrent Bayesian classifier used to identify the most likely task and team combination from observation-action pairs. Observations $Z_t \in \mathbb{R}^{(C\times 5\times 5)}$ are first passed through two 2D Convolutional Layers, obtaining a flattened latent vector $Z'_t \in \mathbb{R}^{64}$, concatenated with the one-hot representation $\mathbf{e}_{A_t^\alpha}$ of the action executed by the ad hoc agent α. The concatenated vector $(Z'_t \oplus \mathbf{e}_{A_t^\alpha}) \in \mathbb{R}^{64+\mathbf{A}}$ is then passed through an LSTM and MLP layer, outputting the logits used for classification which can be used to compute a distribution $p(m_k)$ using an additional Softmax.

Actor and Critic Networks $\hat{\pi}$, $\hat{V}$

The Actor and Critic Networks, used by both ATLPO-MF and ATLPO-MB, have the goal of computing policies and value predictions directly from the latent state space. Both receives as input a latent state $\hat{X}_t$ output, respectively, a policy π_t and predicted value $\hat{v}_t$. Both networks are implemented as MLPs. The Actor Network $\hat{\pi}$ has a Linear output layer with A units that returns the logits used to construct the probability distribution π_t using a Softmax function. The Critic Network $\hat{v}$ has a Linear output layer with a single output unit that returns the value prediction $\hat{v}_t \in \mathbb{R}$. We optimise both Networks via proximal policy optimisation and an Adam Optimiser with a learning rate of 0.001.

Bayesian Classifier $\hat{p}$

The Bayesian Classifier Network $\hat{p}$, used by both ATLPO-MB and ATLPO-MF, has the goal of identifying the most likely task and teammate k from a set of possible tasks and teammates $\{k_1, k_2, \ldots, k_K\}$. It takes as input an observation Z_t and the previous action A_{t-1}^α and returns a probability distribution over $\{k_1, k_2, \ldots, k_K\}$. We implement $\hat{p}$ as a Recurrent Convolutional Classifier. Similar to the Encoder Network q, $\hat{p}$ starts by extracting a flattened latent vector $Z'_t \in \mathbb{R}^{64}$ from Z_t using two 2D Convolutional Layers, and concatenating it with a one-hot representation $\mathbf{e}_{A_t^\alpha}$ of action A_t^α. The resulting vector $(Z'_t \oplus \mathbf{e}_{A_t^\alpha}) \in \mathbb{R}^{64+A}$ then goes through an LSTM layer and an MLP which returns the logits used for classification. The classifier is trained with a Categorical Cross Entropy loss using the labels $\{k_1, k_2, \ldots, k_K\}$, minimised by an Adam Optimiser with a learning rate of 0.001. Figure 6.9 provides an overview of the architecture.

6.2.4 Baselines

We compare our approaches, ATLPO-MF and ATLPO-MB, against four baselines (i.e., other agents which take on the role of α_0) – *Original Teammate*, the original

teammate from the environment which is given the full state, therefore acting as an oracle approach, *Random*, an agent which selects its action randomly in every step, *Model-Free RL*, an agent which knows the task and team beforehand and loads the correct pre-trained model-free policy, and *Model-Based RL*, an agent which knows the task and team beforehand and loads the correct pre-trained model-based policy.

6.2.5 Evaluation Procedure

We consider an ad hoc scenario where an ad hoc agent α is deployed into an unknown environment ϵ where a team $-\alpha$ is performing a task T. Before the agent is deployed into this ad hoc evaluation, its first sets up its policy library Π and recurrent Bayesian classifier $\hat{p}$ in a pre-training stage. It is only after pre-training for a given set of environments ϵ_k that the agent enters an ad hoc evaluation stage, being deployed into an unknown environment ϵ with goal of identifying the correct team and task being performed and assisting in completing the task as fast as possible. We therefore use as metric the average number of steps it takes the team to complete an episode.

Pre-Training Procedure

For a given set of K environments, each with an associated team and task being performed, the pre-training procedure for ATLPO-MF and ATLPO-MB is as following:

1. For each environment ϵ_k, with an associated team and task, the ad hoc agent pre-trains a policy $\hat{\pi}^{\alpha}_k$ by interacting with ϵ_k via trial-and-error for 30000 steps.
2. For each environment ϵ_k, with an associated team and task, the ad hoc agent collects a dataset D_k of trajectory sequences containing observation-action pairs (Z_t, A^{α}_t). These trajectories are collected by interacting with each environment ϵ_k with the final trained policies $\hat{\pi}^{\alpha}_k$, now fixed.
3. The agent creates its policy library $\Pi = \{\hat{\pi}^{\alpha}_1, \hat{\pi}^{\alpha}_2, \ldots, \hat{\pi}^{\alpha}_K\}$ and using the set of combined datasets $\{D_1, D_2, \ldots, D_K\}$ and team/task labels k, trains the Bayesian recurrent classifier $\hat{p}$ to identifies the most likely team and task k given a sequence of observation-action pairs (Z_t, A^{α}_t).

Ad Hoc Evaluation Procedure

After completing the pre-training stage, an agent is evaluated by being deployed into an environment ϵ without knowing in advance the team and task being performed. For a given set of K environments, each with an associated team and task being performed, the evaluation procedure for an agent α is as following:

1. For each environment ϵ_k, evaluate the team with an Original Teammate for $n = 16$ episodes, recording the number of steps s^{*}_k it took to solve each episode.
2. For each environment ϵ_k, replace the Original Teammate with the agent targeted for evaluation and evaluate the resulting team for $n = 16$ episodes, recording the number of steps s^{α}_k it took to solve each episode.

6.3 Results

We now present the results of our experiments. We break down our analysis by research question.

Research Question #1: Are ATLPO-MF and ATLPO-MB effective in assisting the team in completing the tasks?

To answer our first research question, we run trials on each domain on three distinct ad hoc sets — (i) one for Task Identification, where the team policy is fixed on all environments ϵ_k (ii) one for Team Identification, where the task is fixed on all environments ϵ_k and (iii) one for Task and Team identification, where both the task and team policy vary.

We now break down our results by set.

Task Identification

We start with the problem of Task Identification. Agents are deployed into an environment where the team could be performing one out of $K = 4$ tasks. In total, each task is ran, without the agent knowing, a total of 16 trials. Furthermore, to account for the random initialisation of the parameters in our Deep Learning models, we conduct the entire process, which includes pre-training the policies and Bayesian classifiers 3 times, adding to a total of 192 trials per agent. Figures 6.10 showcases the results and confidence intervals with a confidence of 95%.

From these results, we can observe that the agents that know the task being performed, i.e., the Original Teammate, the Model-Free RL agent and the Model-Based RL agent, obtained the best performances, as expected. On the Level-Based Foraging domain, the Original Teammate was able to assist the team in solving episodes in an average of 6.72 steps, the Model-Free RL policy in 7.56 steps and the Model-Based RL policy in 8.13 steps. On the Predator-Prey domain, the Original Teammate was able to assist the team in solving episodes in an average of 9.36 steps, the Model-Free RL policy in 11.22 steps and the Model-Based RL policy in 9.79 steps. These three agents were then followed by the two ad hoc approaches, ATLPO-MF and ATLPO-MB, which are required to identify the correct task. As expected, these two approaches took a slightly longer average than the agents that know the task beforehand, with ATLPO-MF taking an average of 13.02 steps to solve an episode on the Level-Based Foraging domain and ATLPO-MB taking an average of 15.35, and on the Predator-Prey domain, ATLPO-MF taking an average of 21.36 steps while ATLPO-MB taking only an average of 17.34 steps. Finally, as expected, the Random Policy agent obtained the worst performance, achieving an average of 59.05 steps to solve an episode on the Level-Based Foraging domain and 50.66 steps on the Predator-Prey domain. Taking these results into account, we can conclude that on the Task Identification set, both ATLPO-MF and ATLPO-MB were effective in performing ad hoc teamwork, being able to complete the tasks in a near-optimal number of steps.

Team Identification

We then move on to the problem of Team Identification. Similarly to with Task Identification, agents are deployed into an environment where $K = 3$ possible teams could be performing a fixed task. Once more, each team is ran, without the agent knowing, a total of 16 trials, 3 times to account for the random initialisation of the Deep Learning models, adding to a total of 144 trials per agent. Figure 6.11 showcases the results and confidence intervals with a confidence of 95%.

Unlike with the Task Identification set, we can now observe that replacing the Original Teammate actually had a positive impact on performance. This phenomenon is unexpected, given teams where all agents are sharing the same strategies should in theory achieve relatively good performances. However, if we break down the

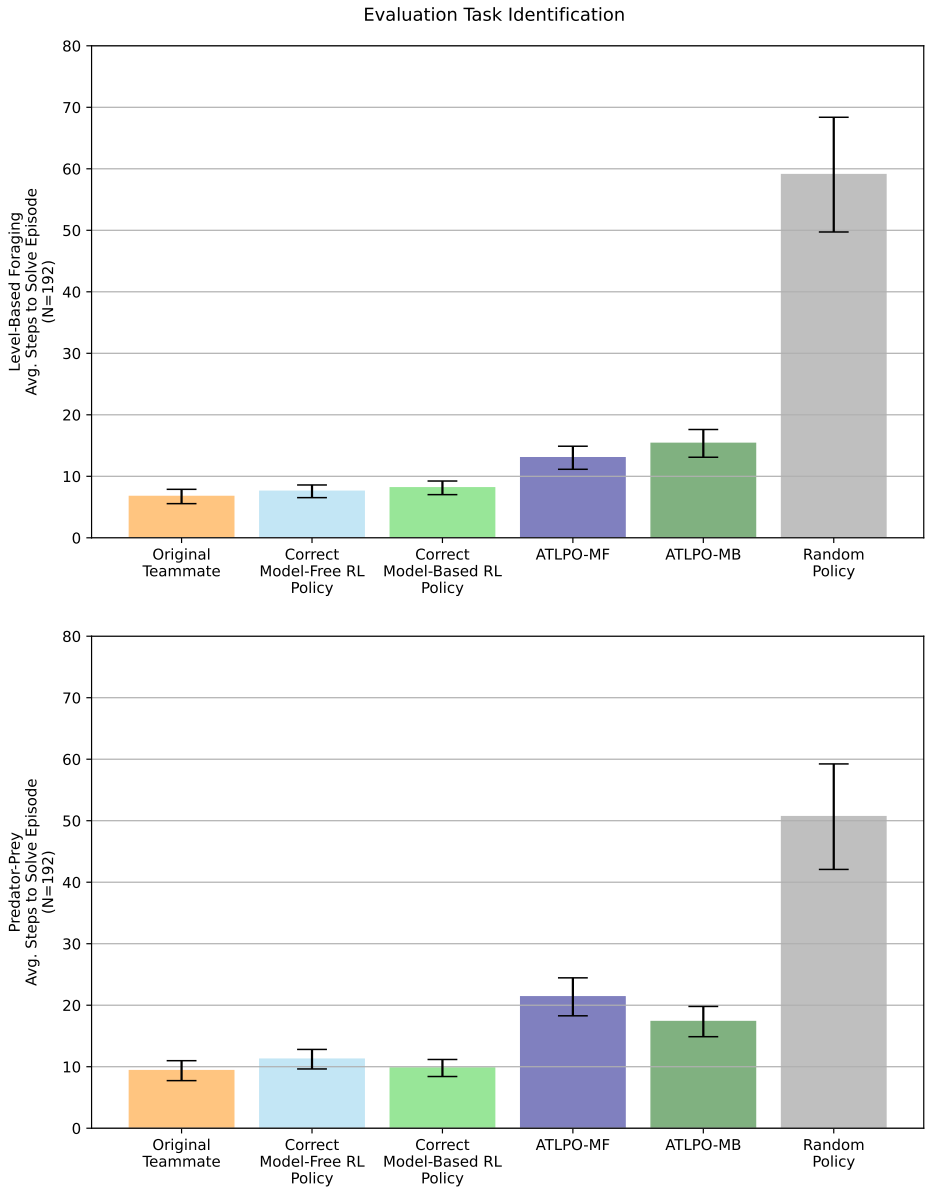


FIGURE 6.10: Average steps to solve an episode on the task identification set. The six agents are evaluated for sixteen trials on each of the four tasks on each of the two domains

evaluation by team (results shown in Table 6.1), we can see that this is in fact not the case for one of the three teams, team τ_2. In the Predator-Prey domain, replacing one of team τ_2's Original Teammates any other agent, excluding the Random Policy agent, actually improves the overall performance of the team. In the Level-Based Foraging domain this effect is less evident, with only the Model-Free RL and Model-Based RL agents outperforming the Original Teammate and with ATLPO-MF and ATLPO-MB achieving a similar performance.

When averaged across all teams, the Original Teammate was able to assist the team in solving episodes in an average of 9.2 steps, the Model-Free RL policy in 7.68 steps and the Model-Based RL policy in 7.31 steps. On the Predator-Prey domain, the Original Teammate was able to assist the team in solving episodes in an average of

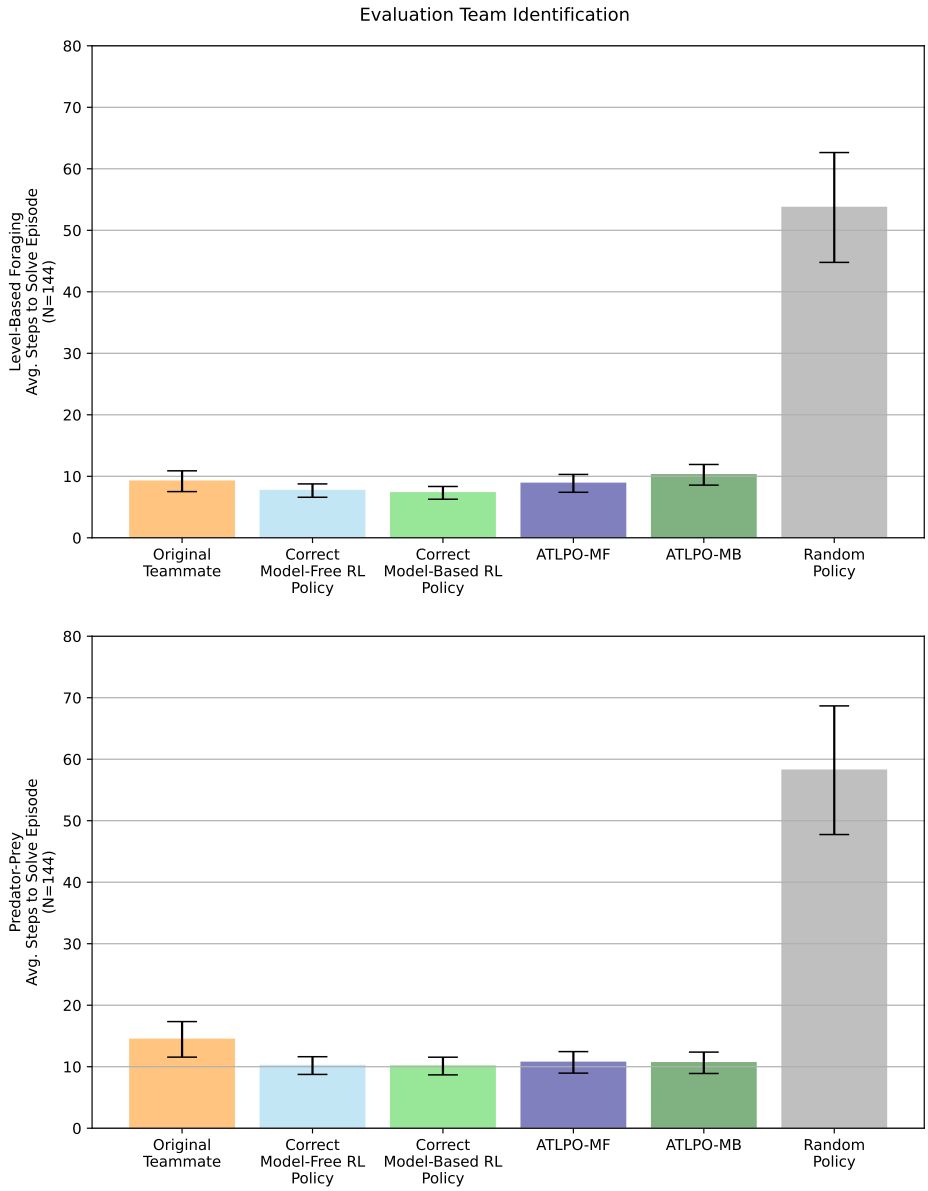


FIGURE 6.11: Average steps to solve an episode on the team identification set. The six agents are evaluated for sixteen trials with each of the three team on each of the two domains

14.45 steps, the Model-Free RL policy in 10.19 steps and the Model-Based RL policy in 10.11 steps. ATLPO-MF and ATLPO-MB, required to identify the correct task took an average of 8.85 and 10.24 steps respectively on the Level-Based Foraging domain. On the Predator-Prey domain, ATLPO-MF took an average of 10.71 and ATLPO-MB an average of 10.64 steps. Finally, as expected, the Random Policy agent obtained the worst performance, achieving an average of 53.71 steps to solve an episode on the Level-Based Foraging domain and 58.21 steps on the Predator-Prey domain.

Taking these results into account, we can once more conclude that on the Team Identification set, both ATLPO-MF and ATLPO-MB were effective in performing ad hoc teamwork, being able to assist the teams in completing the tasks in a near-optimal number of steps.

TABLE 6.1: Average steps it took each agent to solve the episodes when placed in each team of each domain, in the Team Identification problem set.

	Original Teammate		Model-Free RL (Knows Task/Team)		Model-Based RL (Knows Task/Team)		ATLPO-MF (Ours)		ATLPO-MB (Ours)		Random Policy	
Level-Based Foraging												
Team τ_0	4.88	(±1.67)	5.73	(±2.96)	6.69	(±3.72)	7.98	(±6.74)	9.44	(±7.41)	53.85	(±29.86)
Team τ_1	4.41	(±1.11)	7.14	(±2.88)	6.43	(±3.75)	8.04	(±8.33)	9.66	(±9.58)	55.29	(±36.5)
Team τ_2	15.04	(±13.4)	10.16	(±7.03)	9.58	(±6.9)	10.52	(±7.07)	11.62	(±7.83)	51.96	(±31.71)
Predator-Prey												
Team τ_0	8.91	(±5.93)	7.44	(±4.08)	7.42	(±3.69)	8.6	(±3.71)	7.31	(±4.21)	57.32	(±37.74)
Team τ_1	5.94	(±2.4)	7.97	(±4.38)	8.58	(±4.57)	9.6	(±5.63)	9.9	(±5.4)	56.19	(±37.56)
Team τ_2	28.5	(±21.51)	15.16	(±8.98)	14.47	(±8.17)	13.92	(±10.29)	14.71	(±8.07)	61.58	(±36.99)

Task & Team Identification

Finally, we evaluate the agents on the joint problem of Task and Teammate Identification. We follow the same procedure, deploying the agents into an environment where $K = 7$ possible task and team combinations could be selected. Each task/team combination is ran, without the agent knowing, a total of 16 trials, 3 times to account for the random initialisation of the Deep Learning models, adding to a total of 336 trials per agent. Figure 6.12 showcases the results and confidence intervals with a confidence of 95%.

Findings from this set were similar to the ones from the Task Identification set. We once more observe that the agents that know the task being performed, i.e., the Original Teammate, the Model-Free RL agent and the Model-Based RL agent, obtained the best performances. On the Level-Based Foraging domain, the Original Teammate was able to assist the team in solving episodes in an average of 7.89 steps, the Model-Free RL policy in 7.62 steps and the Model-Based RL policy in 7.73 steps. On the Predator-Prey domain, the Original Teammate was able to assist the team in solving episodes in an average of 11.54 steps, the Model-Free RL policy in 10.71 steps and the Model-Based RL policy in 9.95 steps. These three agents were then followed by the two ad hoc approaches, ATLPO-MF and ATLPO-MB, which are required to identify the correct task. As expected, these two approaches took a slightly longer average than the agents that know the task beforehand, with ATLPO-MF taking an average of 11.96 steps to solve an episode on the Level-Based Foraging domain and ATLPO-MB taking an average of 14.06, and on the Predator-Prey domain, ATLPO-MF taking an average of 16.75 steps while ATLPO-MB taking only an average of 15.67 steps. Finally, as expected, the Random Policy agent obtained the worst performance, achieving an average of 56.52 steps to solve an episode on the Level-Based Foraging domain and 54.21 steps on the Predator-Prey domain. Taking these results into account, we can once more conclude that on the Task and Team Identification set, both ATLPO-MF and ATLPO-MB were effective in performing ad hoc teamwork, being able to assist the teams in completing the tasks in a near-optimal number of steps.

Research Question #2: Are ATLPO-MF and ATLPO-MB effective in identifying the correct team and task being performed?

To answer our second research question, we record the ATLPO-MF and ATLPO-MB's beliefs, i.e., probabilities returned by the Bayesian Classifier $\hat{p}$ in each timestep. We assess whether the probability of the correct environment increases with time, becoming the most-likely one.

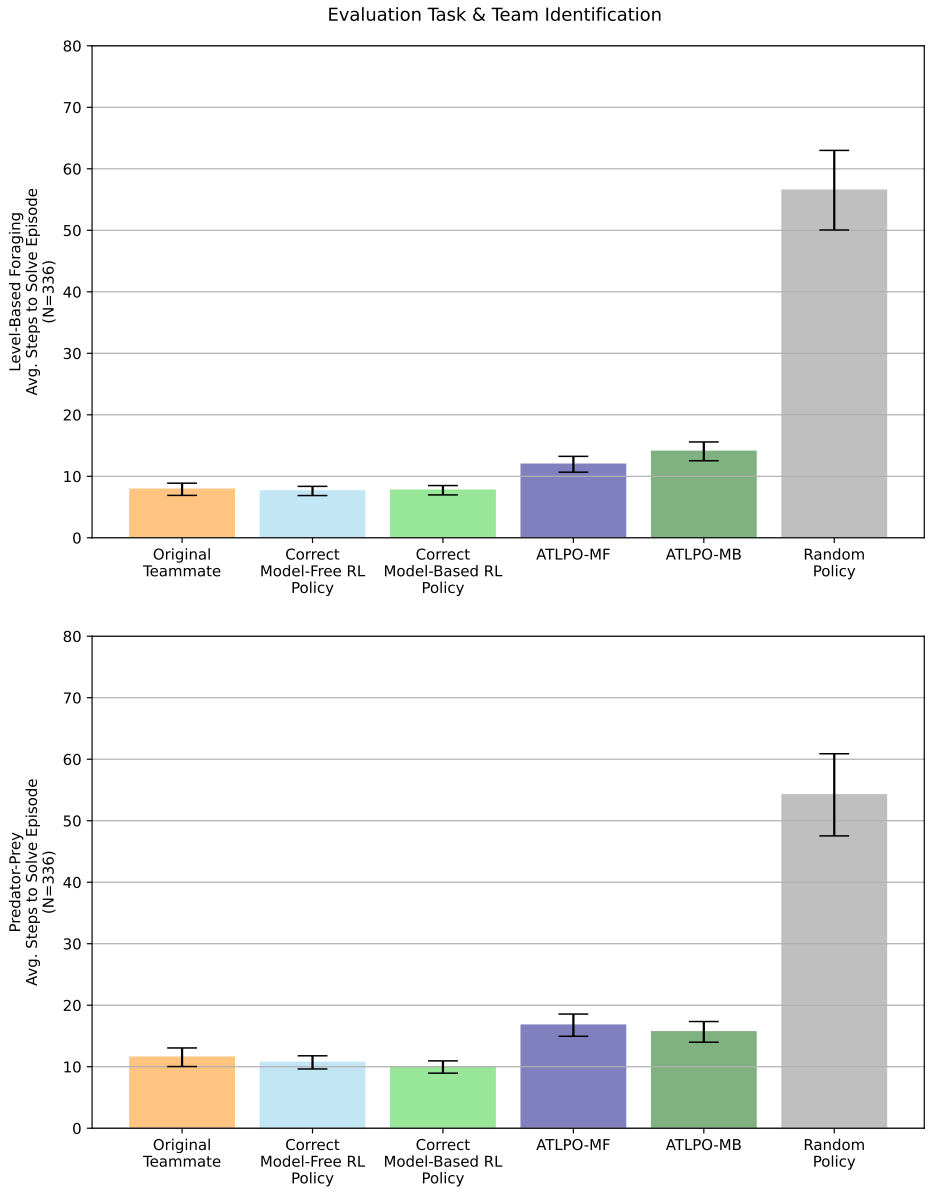


FIGURE 6.12: Average steps to solve an episode on the task and team identification set. The six agents are evaluated for sixteen trials on each of the seven task/team combinations on each of the two domains

Task Identification

Figure 6.13 plots the belief progressions of both ad hoc agents, ATLPO-MF and ATLPO-MB on both domains, Level-Based Foraging and Predator-Prey. Beliefs over the correct task are shown in green, and the average sum of the remaining tasks in red.

As we can see from these results, both approaches were very effective in identifying the correct task being performed in under 10 steps. These results show that when it comes to distinguishing different tasks, the Recurrent Bayesian Classifier is very effective in doing so from observations alone.

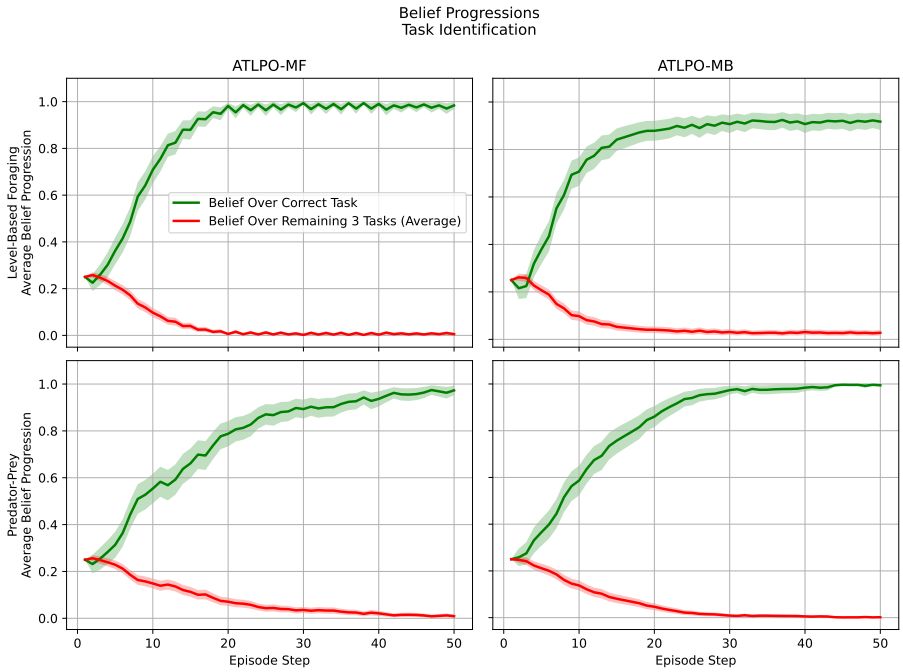


FIGURE 6.13: Belief progressions for ATLPO-MF and ATLPO-MB in both domains. In each timestep, the Recurrent Bayesian Classifier $\hat{p}$ outputs a probability distribution over all tasks, with each entry representing the belief over its respective task. The belief for the correct task is shown in green, while the average sum of the incorrect ones is shown in red.

Team Identification

Figure 6.14 plots the belief progressions of both ad hoc agents, ATLPO-MF and ATLPO-MB on both domains, Level-Based Foraging and Predator-Prey. Beliefs over the correct team are shown in green, and the average sum of the remaining teams in red.

From these results, we can clearly observe that identifying the correct team is harder than identifying the correct task. The most plausible explanation is that for a fixed task, different team policies τ share a lot in common. However, as shown by our evaluation in Figure 6.11, the average number of steps it takes to solve a task is not as affected as in with the problems of Task Identification and joint Task and Team Identification, where the correct setting was more easily identified.

As we can see that both ATLPO-MF and ATLPO-MF, in this case, altough able to identify the correct task in under 10 steps, do not converge to a high degree of certainty. However, one can once more see that this does not affect the performances of the agents when effectively solving the tasks.

Task & Team Identification

Figure 6.15 plots the belief progressions of both ad hoc agents, ATLPO-MF and ATLPO-MB on both domains, Level-Based Foraging and Predator-Prey. Beliefs over the correct task-team combination are shown in green, and the average sum of the remaining task-team combinations in red.

The results in the problem of Task and Team Identification are similar to the ones from the problem of Task Identification. Both approaches were very effective in identifying the correct task being performed in under 10 steps, showcasing the

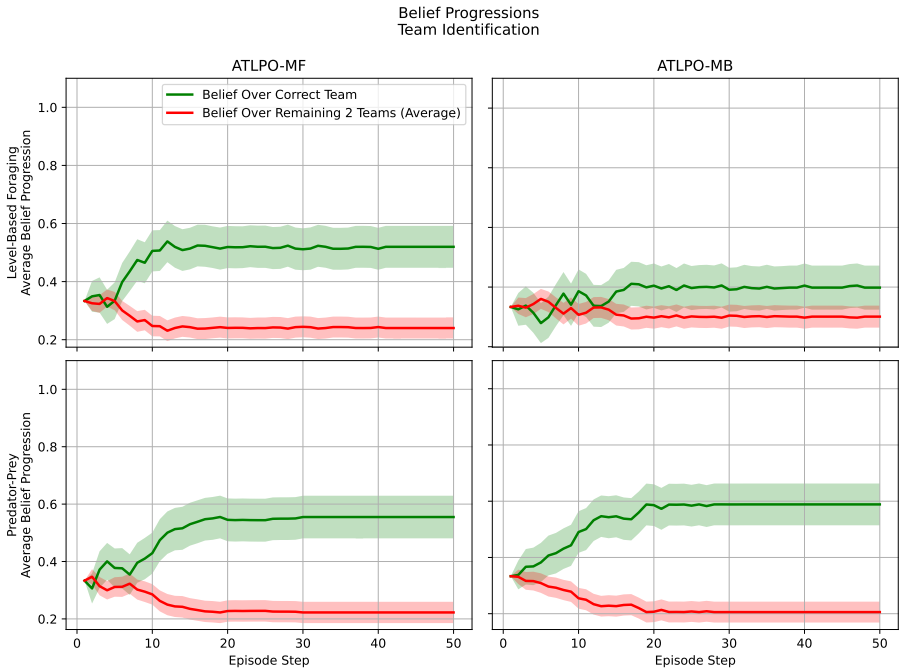


FIGURE 6.14: Belief progressions for ATLPO-MF and ATLPO-MB in both domains. In each timestep, the Recurrent Bayesian Classifier $\hat{p}$ outputs a probability distribution over all teams, with each entry representing the belief over its respective team. The belief for the correct team is shown in green, while the average sum of the incorrect ones is shown in red.

effectiveness of our Recurrent Bayesian Classifier in distinguishing different team-task setups from observations alone.

6.4 Conclusion

In this chapter we addressed our third research sub-problem:

> *How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?*

We expanded upon our prior work, ATPO, proposing and evaluating a novel approach for ad hoc teamwork under partial observability that unlike ATPO is able to handle domains with large observation spaces or state spaces. Our approach, ATLPO, employs state-of-the-art Deep Learning techniques to approximate ATPO's belief update function and library of PERSEUS policies. In our evaluation, we assess the effectiveness of two versions, one model-free, ATLPO-MF, and one model-based, ATLPO-MB, showcasing that according to the current state-of-the-art in both settings, the best one can be chosen. In our results, state-of-the-art techniques were employed in both cases with the performances ending up being similar. Furthermore, we also shown that by using a Recurrent Bayesian Classifier, an ad hoc agent is able to identify the correct team and/or task being performed in a given environment.

◇

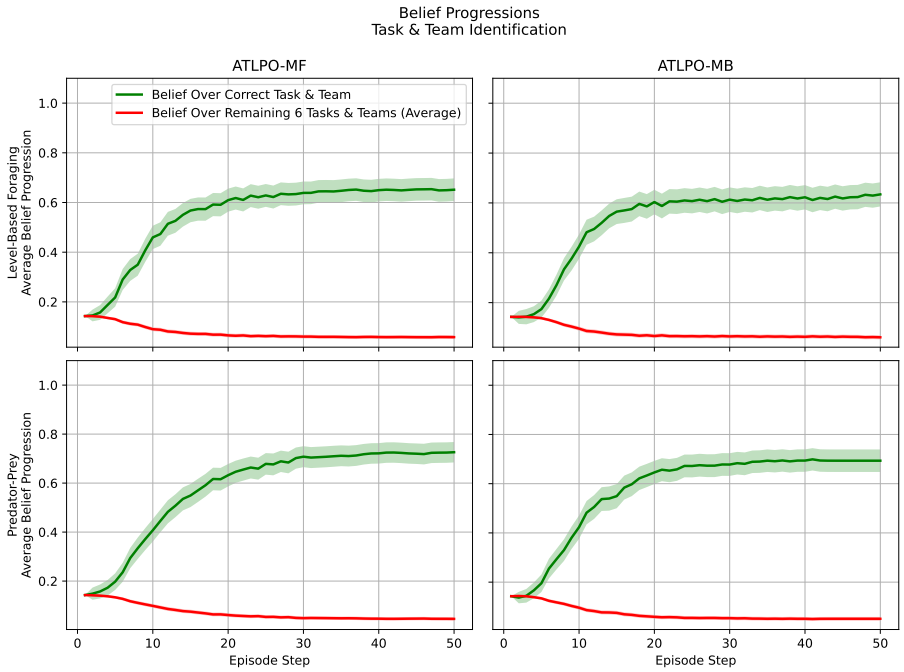


FIGURE 6.15: Belief progressions for ATLPO-MF and ATLPO-MB in both domains. In each timestep, the Recurrent Bayesian Classifier $\hat{p}$ outputs a probability distribution over all team-task combinations, with each entry representing the belief over its respective team-task combination. The belief for the correct team-task combination is shown in green, while the average sum of the incorrect ones is shown in red.

In the next chapter, we conclude the contribution of this thesis, introduce a framework for human-robot interaction which allows ad hoc agents to be deployed seamlessly into a live robot tasked with assisting a human in completing a given task.

Chapter 7

An Ad Hoc Teamwork Platform for Mixed Human-Robot Teams

This chapter addresses this thesis' fourth and final research sub-problem:

> *How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?*

The three previous chapters addressed problems such as the lack of full observability of the environment, the lack of visibility of the teammates' actions and dealing with arbitrarily large environments. Even though chapter 5 presented experiments in a real-world environment with both humans and robots, the evaluated approach (BOPA), required a full state of the environment. In our laboratory setting used throughout the evaluation, a small room and usage of an RGB camera allowed to compute the current state of the environment. In more realistic scenarios, however, relying on sensors to reliably compute the state of the environment is seldom feasible. In such environments, robots will most likely have to rely on their own sensors, and therefore three main challenges arise.

First, the environment is not fully observable, as the robot has to rely on sensors to obtain its perceptions. In fully observable environments, agents are assumed to have access to the current state of the environment, without any errors or imperfect information. With a properly modelled fully observable state (i.e., one that contains all information relevant to the task at hand) given to an agent, optimal planners can be used to compute optimal actions. In a partially observable environment, a state is still assumed to exist, but not observable to the agent (or in this case, the robot). In real-world environments, it is not feasible to, in real time, compute the current state of the environment to allow the robot to plan for optimal action. By using the robot's sensors, however, one may compute a partial observation of such state, and use internal models within the robot to infer the most-likely states and act accordingly. In the HOTSPOT framework, we assume the robot to have only access to partial observations, acquired via its sensors.

Secondly, when interacting with humans as teammates, it is not feasible to assume they will explicitly communicate their actions in real time, as robots would possibly communicate between them if they all shared the same communication protocol (even though in the setting of ad hoc teamwork communication between agents is also assumed to not be always possible due to the fact that the robot knows nothing about the team's members). For this reason, the robot has to rely on other information to infer what the human teammate is doing. In the HOTSPOT framework, we mitigate this issue with two solutions: (i) we assume the human may sometimes communicate with the robot (even though this may never happen at all), and the robot has to be fully

prepared to understand the human's voice and (ii) the actions of the human influence the state of the environment, which in turn influences the partial observations made by the robot. We propose, respectively, a module capable of parsing the human's voice and converting it into a possible action and an approach to infer the most likely action given the partial observations of the environment.

Furthermore, it is also not feasible to assume the behaviour of the human user will always be optimal and efficient (as humans may stop what they are doing without warning or even act inefficiently). This challenge requires robots to be robust to sub-optimal behaviour. In the HOTSPOT framework, we assume that even though human teammates know the task being performed, they may suddenly not act or act sub-optimally, evaluating how we are able to mitigate this issue by using teammate models of the humans.

Our proposed architecture therefore relies on two main modules — (i) the decision-making module which also handles partial observability and (ii) the communication module enables our robot to leverage the communication capabilities of the human user toward the completion of the joint task.

Regarding Natural Language Processing (NLP), this is a field of computer science that enables machines to understand and process human communication [Manning and Schütze, 1999] by transforming unstructured data, like audio or text, into structured data, which are more suitable for machines. It can also work in the opposite direction by generating communication for humans to understand easily.

Many works address the use of NLP in interaction with robots. Scheutz et al. [2011] presented the challenges of designing mechanisms that allow robots to develop human dialogues in interactions between humans and robots. In addition to building a small survey of the area, its main objective is to help build better, more flexible robotic architectures that can enable more natural language dialogues between humans and robots. Furthermore, the authors briefly propose DIARC, a Distributed, Integrated, Affective, Reflective, and Cognitive architecture that allows robotic systems to conduct human dialogues without providing much detail about the techniques involved in the process.

On the other hand, Kilicaslan and Tuna [2014] explored the use of NLP resources to improve human-robot interaction through the use of ontologies that represent the grammatical and lexical structures of the language. Implementations have been generated for English and Turkish languages to allow the robot to express the spatial motions of observed objects. Through the use of these ontologies, a representation of the robot's observation can be described by a tuple $<$Location, Source, Goal, Path$>$, where Location refers to the position of the observed object, Source represents where the object moved from, Goal refers to the target position of the object, and Path represents the trajectory that the object has made. A prototype was built using the ROS framework, where a camera captures object movements in front of the robot to describe the movement in natural language.

Briggs et al. [2017] used pragmatic and dialogue-based mechanisms to understand typical human directives and create suitable responses. Specifically, utterances are used to represent the speech act classification, as well as the speaker, robot, and semantics analysis. Then, rules associate the utterances with a tuple containing the set of inferred beliefs based on the intended meaning of the utterance. For instance, a question inquiring whether some assertive is true or false. Finally, a dialogue-based mechanism handles and generates the responses based on expectations generated by the utterances. Experiments were then conducted to show the viability of the proposed methodology in identifying indirect speech acts and coverage of the utterance forms.

Li et al. [2021] used NLP to infer human-given commands for robots, by using keyword extraction, visual object recognition, and similarity computation. Its main intent is to use visual semantic information to allow a robot to deduce task intents, avoiding simple keywords that map predefined tasks explicitly. The proposed method uses rule matching and conditional random fields to analyse and extract information from the processed sentences.

Despite several works that use NLP in robots, none of them are tailored to the ad hoc teamwork scenario.

This chapter thus presents a novel contribution to the scientific literature, namely an architecture that combines decision-making and NLP for human-robot collaboration within ad hoc teamwork settings.

7.1 The HOTSPOT Framework

Figure 7.1 presents the two main modules in the HOTSPOT architecture, namely:

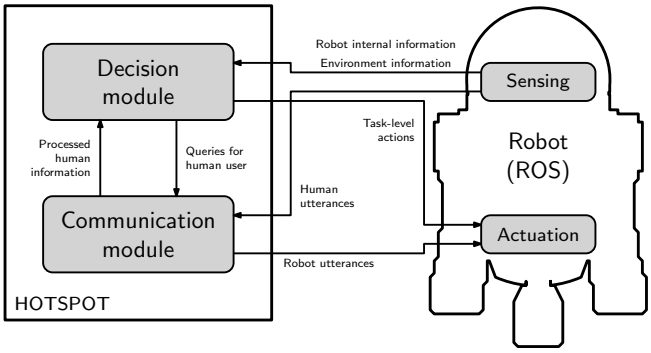


FIGURE 7.1: **HOTSPOT Diagram.** Diagram depicting the interaction between the different modules in HOTSPOT.

- *A decision module* that is responsible for ad hoc teamwork decisions with the human. This module receives as input the robot and environment information from the sensors and the relevant information from the human speech (i.e., the information processed by the communication module). The decision module then uses such information to reconstruct/estimate the current state of the environment and the human-robot team. Finally, the robot uses the state information to estimate the current task (*task identification*), to identify how the human is executing such task (*teammate identification*), and to act accordingly (*planning*).

- *A communication module* that is responsible for the communication with the human user. It receives queries from the decision module and translates them into spoken utterances that the robot must execute (verbalise and animate). It is also responsible for processing human utterances, as perceived by the sensors, providing the decision module with their relevant information.

In the remainder of this section, we describe both modules in greater detail.

Decision Module

The decision module is depicted in Figure 7.2. The module processes the information coming from the sensors and communication module to estimate the *state* of the

current task. Specifically, HOTSPOT maintains a distribution over possible states—a *belief*—which is updated from the perceived information using a standard Bayesian update.

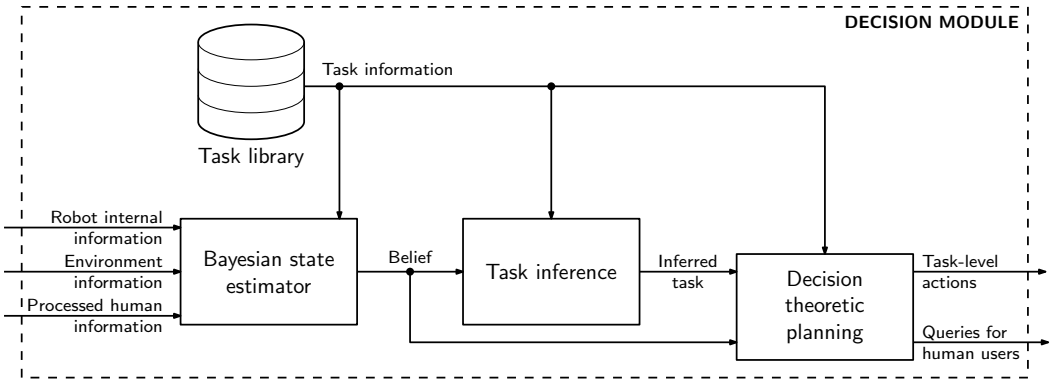


FIGURE 7.2: **HOTSPOT Decision Module Diagram.** Diagram depicting the decision module of HOTSPOT.

The robot then uses information about possible tasks (stored in a *task library*) to infer the current task by checking which tasks in the library are most likely to yield the perceived information from the environment and the teammate. Finally, using the belief and task information, the robot plans the actions to complete the task. It also determines which (if any) communication actions it should perform toward the human.

In the continuation, we formalise each of these processes in detail.

Task description

We follow the same ad hoc problem formulation described in section 3.

We consider that there is a set of M possible tasks, each described as an MMDP $\mathcal{M}_m = (\mathcal{X}, \{\mathcal{A}^\alpha, \mathcal{A}^{-\alpha}\}, \mathbf{P}_m, r_m, \gamma)$, where all tasks share the state and action spaces but may have different dynamics and goals. Furthermore, we assume that the robot does not know beforehand which task is being performed—henceforth referred to as the *target task* T—but the human user does know. Task identification thus consists of inferring the target task from the information that the robot can observe during the interaction.

Bayesian state estimator

During the interaction, the robot can observe the information available through its sensors and information provided by the communication module regarding the human spoken utterances. We denote by Z_t the information observed by the agent—which we assume takes values in a finite set of possible observations, $\mathcal{Z}$. We also denote by $\mathbf{O}$ the *observation probabilities*, which essentially provide a probabilistic description of the sensing process of the robot. In particular,

$$\mathbf{O}(z \mid x, a^\alpha) = \mathbb{P}\left[Z_{t+1} = z \mid X_{t+1} = x, A_t^\alpha = a^\alpha\right],$$

with $z \in \mathcal{Z}$ and $x \in \mathcal{X}$. The observation probabilities describe how likely it is for the robot to observe z in state x, given that the last action of the robot was a^α.

Let $b_t(x)$ denote the probability that, at time step t, the state is $x \in \mathcal{X}$, given the history of observations and actions of the robot up to that time step (henceforth

H_t). Let us further assume that, at time step t, the robot performs the action a^α, and the human user performs the action a^α. As a consequence, the environment will transition to state X_{t+1} and the robot will observe $Z_{t+1} = z$. Then, if the target task is m, we can update our belief b_t using a standard Bayesian update to have

$$\begin{aligned} b_{t+1}(x) &= \mathbb{P}\left[X_{t+1} = x \mid H_{t+1}\right] \\ &= \frac{1}{\rho} \sum_{y \in \mathcal{X}} \mathbf{P}_m(x \mid y, a^\alpha, a^{-\alpha}) \mathbf{O}(z \mid y, a^{-\alpha}) b_t(y), \end{aligned}$$

where ρ is a normalisation constant.

There are two difficulties with using this update: first, we do not know which is the action of the human teammate, $a^{-\alpha}$; and second, we do not know which is the target task, m.

To address the first difficulty, and since we assume that the teammate knows the target task, we consider that—if the target task is m—the action of the teammate can be *any* optimal action for the task m. Then, if we average out the action selection of our human teammate, we get the (task-dependent) transition probabilities

$$\bar{\mathbf{P}}_m(y \mid x, a^\alpha) = \frac{1}{|\mathcal{A}^{*-\alpha}(x)|} \sum_{a^{-\alpha} \in \mathcal{A}^{*-\alpha}(x)} \mathbf{P}_m(y \mid x, a^\alpha, a^{-\alpha}),$$

where $\mathcal{A}^*_{-0}(x)$ denotes the set of optimal teammate actions in state x. This yields a task-dependent belief update

$$\begin{aligned} b_{m,t+1}(x) &= \mathbb{P}\left[X_{t+1} = x \mid H_{t+1}, T = m\right] \\ &= \frac{1}{\rho} \sum_{y \in \mathcal{X}} \bar{\mathbf{P}}_m(x \mid y, a^\alpha) \mathbf{O}(z \mid y, a^{-\alpha}) b_{m,t}(y), \end{aligned} \tag{7.1}$$

where ρ is, again, a normalisation constant.

Regarding the second difficulty, since the robot does not know beforehand the target task, it maintains a distribution p_t over the set of possible tasks. In other words, we write $p_t(m)$ to denote the probability that the target task is m given the history of observations and actions of the robot up to time step t. Then, given the distribution p_t, we can write the "average" belief at time step t as

$$b_t(x) = \sum_{m=1}^{M} p_t(m) b_{m,t}(x). \tag{7.2}$$

Task inference

We now describe how to maintain the distribution p_t, used to perform Bayesian state estimation. Much like with the state estimation, we adopt a Bayesian framework. Let T denote the unknown target task. Then, if the agent observed z at time step $t+1$ after executing a^α at time step t,

$$\begin{aligned} p_{t+1}(m) &= \mathbb{P}\left[T = m \mid H_{t+1}\right] \\ &= \frac{1}{\rho} \mathbb{P}\left[Z_{t+1} = z \mid A^\alpha_t = a^\alpha, T = m, H_t\right] \mathbb{P}\left[A^\alpha_t = a^\alpha \mid H_t\right] p_t(m), \end{aligned}$$

where, once again, ρ is the necessary normalisation constant. We used the fact that the action selected by the agent at each moment depends only on the history of observations and not on the target task T. Then,

$$\mathbb{P}\left[Z_{t+1} = z \mid A_t^\alpha = a^\alpha, T = m, H_t\right] = \sum_{x,y \in \mathcal{X}} \mathbf{O}(z \mid y, a^\alpha) \mathbf{P}_m(y \mid x, a^\alpha) b_{m,t}(x).$$

Decision-theoretic planning

To decide what action to take, and given the uncertainty in the robot's perception of its state, we adopt as the planning approach a well-established information-gathering heuristic Melo and Ribeiro [2006]. In particular, at each time step t, the robot selects its actions to balance *information gathering* and *task completion*.

Information gathering consists of selecting actions that decrease the uncertainty in the state estimation, b_t. Task completion actions are selected to solve the target task, T.

To this purpose, we compute the *normalised entropy* of b_t, given by

$$\bar{H}(b_t) = -\frac{1}{\log |\mathcal{X}|} \sum_{x \in \mathcal{X}} b_t(x) \log b_t(x).$$

The normalised entropy measures the uncertainty in the agent's belief. Let $b_{\max}(z, a^\alpha)$ denote the robot's belief upon observing z after executing a^α in task m from a belief with maximum entropy, i.e.,

$$b_{m,\max}(z, a^\alpha) = \frac{1}{\rho} \sum_{x \in \mathcal{X}} \bar{\mathbf{P}}_m(y \mid x, a^\alpha) \mathbf{O}(z \mid y, a^\alpha) \frac{1}{|\mathcal{X}|}.$$

We define the *information gain* associated with (z, a^α) as

$$\Delta H_m(z, a^\alpha) = 1 - \bar{H}(b_{m,\max}(z, a^\alpha)).$$

We also define the *reward gain* associated with (z, a^α) as the maximum reward that the robot can get upon observing z after executing a^α in task m from a belief with maximum entropy, i.e.,

$$\Delta R_m(z, a^\alpha) = \max_{a^{\alpha'} \in \mathcal{A}^\alpha} \sum_{x,y \in \mathcal{X}} \bar{\mathbf{P}}_m(y \mid x, a^\alpha) \mathbf{O}(z \mid y, a^\alpha) \frac{\bar{r}_m(y, a^{\alpha'})}{|\mathcal{X}|},$$

with

$$\bar{r}_m(x, a^\alpha) = \frac{1}{|\mathcal{A}^{*-\alpha}(x)|} \sum_{a^{-\alpha} \in \mathcal{A}^{*-\alpha}(x)} r_m(x, a^\alpha, a^{-\alpha}).$$

Following Melo and Ribeiro Melo and Ribeiro [2006], we define an *information gathering reward function* as

$$r_{m,\text{info}}(x, a) = \sum_{z \in \mathcal{Z}} \mathbb{P}\left[Z_{t+1} = z \mid X_t = x, A_t^\alpha = a^\alpha, T = m\right] \Delta H_m(z, a^\alpha) \Delta R_m(z, a^\alpha).$$

Then, for each task m, we can now define two standard MDPs, $(\mathcal{X}, \mathcal{A}^\alpha, \bar{\mathbf{P}}_m, r_m, \gamma)$ and $(\mathcal{X}, \mathcal{A}^\alpha, \bar{\mathbf{P}}_m, r_{m,\text{info}}, \gamma)$, which can be solved to yield two optimal Q-functions [Puterman, 2005], Q_m^* and $Q_{m,\text{info}}^*$.

Finally, the action selected at each step t is given by

$$a_t^{\alpha} = \text{argmax} \sum_{m=1}^{M} p_t(m) \sum_{x \in \mathcal{X}} b_t(x) \left[(1 - \bar{H}(b_t)) Q_m^*(x, a) + \bar{H}(b_t) Q_{m,\text{info}}^*(x, a) \right].$$

When the uncertainty in b_t is high (close to 1), the robot selects an information-gathering action, i.e., an action that maximises $Q_{m,\text{info}}^*$; when the entropy is low (close to 0), the robot selects a task competing action, i.e., an action that maximises $Q_m^*(x, a)$.

Communication Module

Figure 7.3 depicts the communication module, which plays two roles in the overall HOTSPOT architecture. First, it plays a *sensing role*, transforming the human speech (captured through a microphone) into state information that is then used by the decision module. Second, it also plays an *acting role*, converting the communication actions received from the decision module into utterances that the robot then speaks to the human user. Each role corresponds to a well-defined pipeline, as depicted in Figure 7.3.

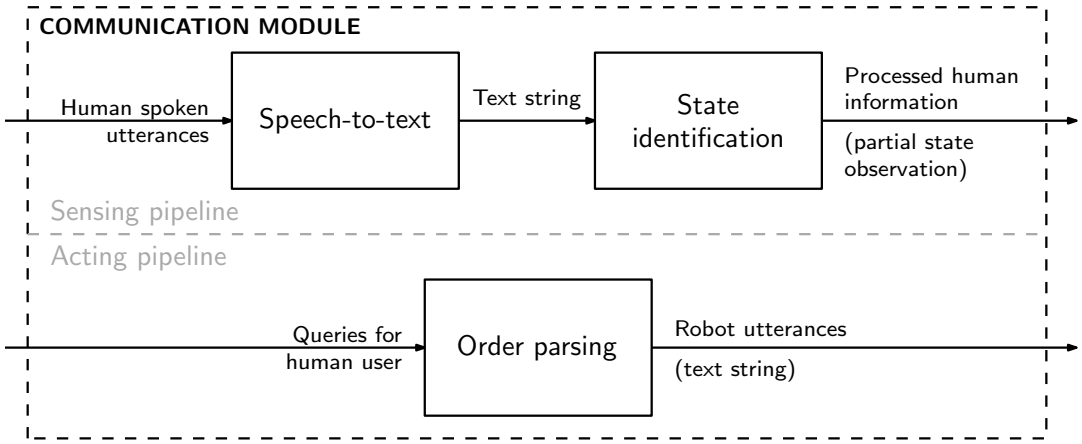


FIGURE 7.3: **HOTSPOT Communication Module Diagram.** The communication module, comprising both a *sensing* pipeline that converts speech to state information, and an *actuation* pipeline, converting communicating actions to text to be then spoken by the robot.

Concerning the sensing pipeline, the communication between the robot and the human user occurs through speech, captured by a microphone, and transformed into audio data, usually in the form of a .wav file containing the recorded human spoken utterances. Next, the audio data is transcribed by a speech-to-text block into a text format that better represents the audio in the language being spoken. Subsequently, in the state identification block, the transcribed text is passed into an NLP processor to extract semantic information, which is then translated into a partial state description.

The description of the state is then used as one of the several inputs to the decision module, which returns action(s) based on the behaviour explained in the Decision Module section. These actions are mapped both into task-level and communication actions. The latter actions are passed to the communication module once again (i.e., the acting pipeline).

On the other hand, the acting pipeline is responsible for converting the communication actions into a text string, which is then sent to the robot. Specifically, this is done by the order parsing module, which provides the utterances to the robot, thus closing the cycle of communication between HOTSPOT and the robotic platform.

Interaction with the Robot

The interaction between HOTSPOT and the robot relies on the *robot operating system* (ROS) [Stanford Artificial Intelligence Laboratory, 2018], which is responsible for sensor handling (namely, processing all sensor readings), robot control, and communication. In other words, ROS supervises all sensing and actuation of the robot, and it is through a ROS interface that HOTSPOT interacts with the robot.

In particular, ROS collects all sensor data, arriving both from the robot sensors—such as odometry sensors used in dead-reckoning, lasers, contact sensors, etc.—and environment sensors—such as microphones, cameras, and other sensors that may exist in the environment. The speech data is sent to the communication module, while the remaining sensor data is processed and sent to the decision module.

ROS is also responsible for the actuation of the robot. In particular, it receives the task-level actions (such as moving) from the decision module and the text strings (corresponding to the utterances that the robot should speak) from the communication module and performs these on the robot.

Experimental setup

To evaluate the effectiveness of our approach, we created a controlled environment where a live robot interacts with a human teammate. In this environment, the robot aims to assist the human in cleaning up a room, with the interaction being restricted by a set of rules from the Toxic Waste Domain.

The Toxic Waste Domain

The Toxic Waste domain has a two-agent team composed of a human cleaner and a robot container. The team is in a building with several rooms and has to clean three rooms with toxic or radioactive waste. A specific task from this domain lays out the rooms in a topological map, where the nodes represent the rooms, and the edges represent the doors that connect them. In addition, some rooms may contain toxic material on the ground. Hence, in each time step, both the human and the robot may choose to move from one room (node) to another or stay in the same node (i.e., no-op).

Whenever it finds itself in a node containing toxic waste, the human can pick it up from the ground or release it (if already holding it). After picking up the toxic waste, the human must remain standing still on his current node and wait for the robot to get close to disposing of the toxic material into the robot's container. The robot also has an additional action to query the human by location (which the human may or may not respond to). Finally, in each time step, the robot receives its current location as an observation, inferred by the dead reckoning module.

Figure 7.4 shows the Toxic Waste domain that we created within our laboratory. Precisely, we recreate two tasks by dividing our laboratory room into five distinct areas, representing separate rooms: 0 - door, 1 - open space, 2 - robot station, 3 - single workbench, and 4 - double workbench. To represent the toxic waste that humans can collect, we use three colored balls placed in three different nodes, as depicted in Figure 7.5. The location of the three balls models each task; that is, there are two possible tasks, each with the three balls placed in three respective areas, as shown in Figure 7.6.

To represent the human cleaner, we rely on people from a small focus group who have been told the goal in advance and know how to act according to the domain's

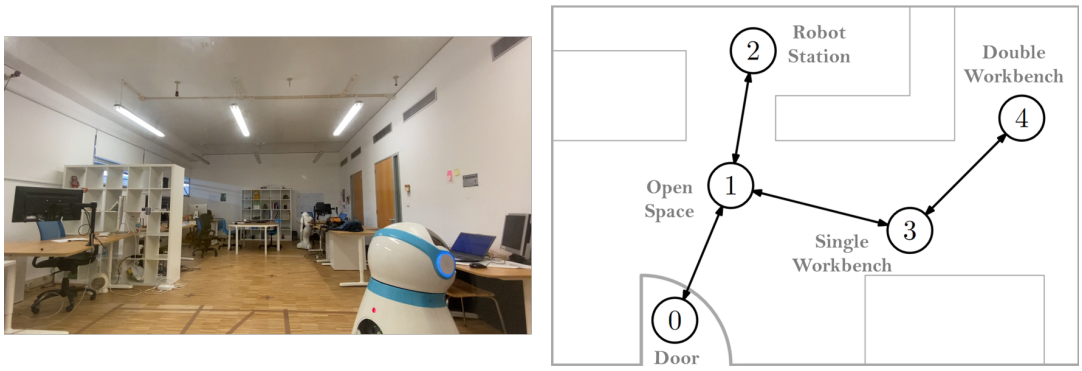


FIGURE 7.4: **Lab Room Setup.** Lab room used to simulate the Toxic Waste domain (left) and respective layout (right). Each area is represented as a node in a topological map.

FIGURE 7.5: **Toxic Waste Materials.** The three balls representing the toxic waste materials.

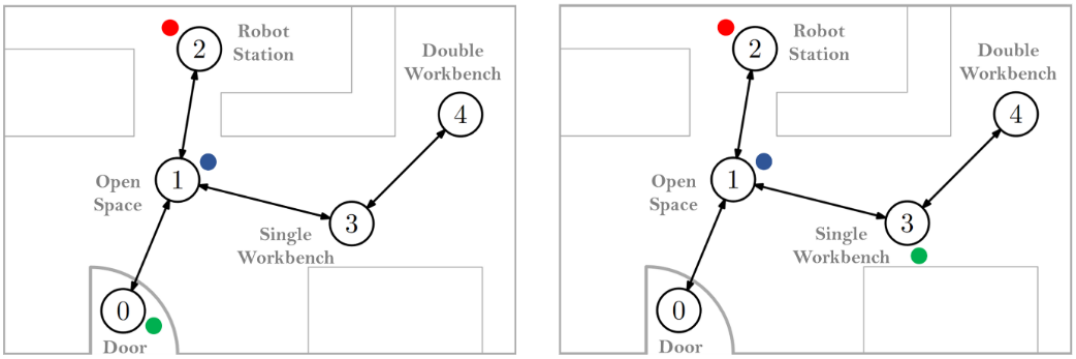


FIGURE 7.6: **Task Configurations.** The two task configurations (i.e., locations of the three balls representing the toxic waste materials).

rules, namely: i) they may only make one move at a time, ii) they may reply to the robot's questions, and iii) they may only move from one area to the another if they are connected. We rely on Astro (Figure 7.4 - left) to be the robot container. It is a robot from our laboratory capable of moving around the room and possessing a front recipient equipped with an RFID sensor to detect when a ball is placed inside the recipient. For each person in the focus group, we randomly chose a task from the two possible tasks, with the human starting in the area of their preference and the robot always starting at the door.

The Decision Module

We instantiate the decision module as a Python 3 ROS node running on a laptop (connected via wifi to a ROS master node running on the robot). The implementation of the module requires only a library of possible tasks, which are then used by our Bayesian state estimator, task inference, and decision-making algorithm.

To model the tasks in the Toxic Waste domain in our framework, we must specify each corresponding MMDP, as well as the observation space and probabilities, describing the sensing process of the robot. Each MMDP is a tuple $(\mathcal{X}, \mathcal{A}^{\alpha}, \mathcal{A}^{-\alpha}, \mathbf{P}_m, r_m, \gamma)$ with distinct transition probabilities and reward functions. Together with the specification of the observation space and observation probabilities for the robot, they provide all the necessary information required by the decision-making module.

In our experiments,

- the state space $\mathcal{X}$ contains information regarding both agents' nodes and the status of the three toxic materials (i.e., on the ground, picked up, or disposed of). In particular, a state $x \in \mathcal{X}$ is a tuple $(n_r, n_h, w_1, w_2, w_3)$, where n_r and n_h represent the node of the robot and human, respectively, and w_i represents the status of the toxic waste material i.

- the action space for the robot, $\mathcal{A}^{\alpha}$, has five possible actions available: *move the lowest-index node*, *move the second lowest-index node*, *move the third-index node*, *stand still*, and *ask the human for his location*. The human action space, $\mathcal{A}^{-\alpha}$, has similar move actions and, additionally, a *pick waste* and *drop waste* actions.

- the transition probabilities $\mathbf{P}_m$ describe how the robot and human move as a consequence of their movement actions, and the status of the waste material as a consequence of the actions of the human user.

- the reward functions r_m assign a penalty of -1 for each toxic waste on the ground, -2 for each toxic waste on the human's hands, and 0 for each toxic waste material disposed of.

- we use a discount $\gamma = 0.95$.

- the observations describe what the robot can observe regarding the state of the environment and the human user. Specifically, each observation $z \in \mathcal{Z}$ corresponds to a tuple $(\hat{n}_r, \hat{n}_h, rfid)$, where $\hat{n}_r$ represents the robot's node (determined via dead reckoning) and $\hat{n}_h$ represents the human's node (determined from speech). *rfid* is a boolean flag indicating that the container detected the collection/disposal of toxic waste.

- as for the observation probabilities $\mathbf{O}$, regarding the location of the robot and RFID sensor, we empirically assessed that the error in these measurements was negligible. As for the position of the human (perceived from human speech), we ran a preliminary study where, for each possible human node, we script out several phrases from a small focus group (i.e., 4 different speakers reading 257 different phrases). We then build a confusion matrix (Figure 7.7) using Python's machine learning library scikit-learn, which tells, for each true node, the probability of identifying every other node or even failing to identify any node. Finally, to handle live errors, we smooth the probabilities when loading the model using the confusion matrix to ensure that no entry has an absolute zero.

The Human Teammate

When it comes to modeling human behaviour in the toxic waste domain, we model the human teammate as an agent that knows the task being performed and fully

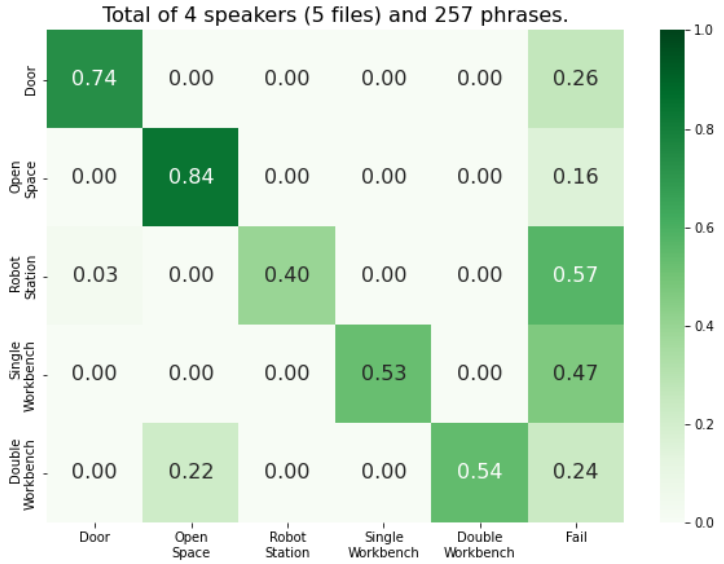


FIGURE 7.7: **Confusion Matrix.** Communication module's pipeline confusion matrix.

observes the environment, unlike the ad hoc robot which has to identify the task and is only able to partially observe the environment.

In each time step t, the human teammate has a probability ϵ of not selecting the optimal action a^*_{-0}, computed from the optimal policy $\pi^*_{-0}(x_t)$. This parameter represents the probability of the human not moving, simulating distractions such as looking at the phone, talking with someone, or moving to a non-optimal node by mistake. Furthermore, and although the option of replying to the robot's action 'locate human' is not part of the human's MMDP action space $\mathcal{A}^{-\alpha}$, in the POMDP's observation probabilities $\mathbf{O}$ for the robot action *locate human*, we consider an additional probability of the human not successfully replying to the robot. This takes into consideration situations where, for any particular reason, the human doesn't reply at all or situations where the communication module fails to correctly interpret the human's reply.

Metrics

Given that our approach has several independent modules, we evaluate the system with the following metrics: i) the number of steps it takes to solve a task, ii) TEBOPA's number of steps to identify the correct task, iii) entropy in TEBOPA's belief over tasks, iv) communication module's speech recognition accuracy, v) communication module's named entity recognition accuracy, vi) communication module's node location recognition accuracy and vii) robot's dead-reckoning accuracy in identifying the correct node. We now break down each individual metric.

Metric 1 - The number of steps it takes to solve a task

Our first and main metric used was the number of steps to solve a task. This metric is used to assess the performance of the team, and the fewer the number of steps it takes the team to solve a task, the better. In our real-world scenario, a third-party human assistant, observing the experiment, registers the number of steps until the task is considered complete, i.e., when all three balls have been placed within the robot's compartment.

Metric 2 - TEBOPA's number of steps to identify the correct task

Our approach, TEBOPA, keeps a belief over the possible tasks, as it interacts with the environment. Each entry in this belief vector measures the likelihood of the respective task. As the robot interacts with the environment, our algorithm TEBOPA, performs a Bayesian update to these probabilities using the observations made in each step. We log, in each step, the task with the highest belief, which represents the task our approach 'guesses' as being the correct one. When, in a given time step T, the likelihood of the correct task becomes the highest in the belief vector, and stays so until the end of the interaction, we consider it took our approach T steps to identify the correct task.

Metric 3 - Entropy in TEBOPA's belief over tasks

As detailed in our description of our second metric, our approach, TEBOPA, keeps a belief over the possible tasks, as it interacts with the environment. One metric we can therefore use to measure the 'uncertainty' of our agent, is the entropy of the belief vector. The lower the entropy, the more 'certain' our agent is of the task being performed. We compute the entropy for a trial after the trial ends, relying on the belief vectors logged by our decision module.

Metric 4 - Communication module's speech recognition accuracy

Our fourth metric measures the accuracy of the first component of our communication module, the speech recognizer. The speech recognizer takes as input the audio of a speaker and outputs the respective text. Unlike the prior metrics, this metric is computed prior to the experiments, when setting up the confusion matrix for the model of the environment that needs to take into account the probability of failing to interpret the human's utterances. Each speaker is told to read a list of phrases, and then the resulting text output matched against the original phrase. Some nuances such as the difference in capital letters of punctuation were ignored if they did not grammatically change the phrase.

Metric 5 - Communication module's named entity recognition accuracy

Our fifth metric measures the accuracy of the named entity recognition component of our communication module. Like the previous metric, it was also measured prior to the experiments, as required by the model of the environment to take into account the probability of failure when modeling the POMDP. Taking as input the text output of the speech recognition component, the named entity recognition component outputs a location entity, such as 'door' or 'table'. Associated with each human phrase, the correct entity is labeled. If the output of the named entity recognition component matched the location entity for the respective phrase, we would consider the entity correctly identified.

Metric 6 - Communication module's node location recognition accuracy

Another communication module's metric is the node location recognition accuracy. The node location recognition component of our communication module takes as input the entity from the named entity recognition component, which it then uses to predict the correct node. A node label is given to each phrase and used to check if the output is correct. Like our previous two metrics, this metric was also assessed prior

to the trials, when computing the confusion matrix and failure probabilities for the models used by the agent.

Metric 7 - Robot's dead-reckoning accuracy in identifying the correct node

Our seventh and final metric measures the number of correct node identifications made by the robot's dead-reckoning system. Each time the robot moves from one node to another, it computes the new position by adding the required offset into its current one. Afterward, the robot moves into the new position and assumes to be there, a process known as dead-reckoning. Each point from the robot's internal referential is then mapped into the area of the nodes from the Toxic Waste domain, and, as such, the robot is able to 'guess' its own node. A third-party human observer then registers the nodes the robot navigates throughout the trial, while the decision module also registers the 'guessed' nodes. These two lists are then used to compute the accuracy of the dead-reckoning sensors.

Participants and Procedure

In this section, we describe how our participants were recruited and briefed, and how our experimental procedure was conducted.

Selection of the focus group

We recruited a total of seven members from our laboratory, *pro bono*, for the empirical evaluation of the HOTSPOT framework. The minimum age was 23 years and the maximum was 42. When contacting each participant, no details regarding the trials were discussed, only a date scheduled. On the date and at the time of the trial, participants were told by an experiment moderator what the task consisted of — having to pick up the three balls from the floor and deposit them on the robot only when the robot would share the node with them. They were told they were unable to move if they had picked up any balls and the dynamics were turn-based. No two participants from the focus group observed each other's trials.

Monitoring of the trials

Every trial was monitored by one of the authors (João G. Ribeiro) in the role of a moderator. The moderator starts by briefing the participant on what the task consisted of, and the aforementioned rules. The moderator then attached the microphone to the participant's clothes and told the participant it was free to answer the robot when asked questions. Afterwards, the moderator would ask the participant to select a starting location, and chose, at random, both a task and a starting location for the robot which did not match the location of the human. The trial would then start. Finally, the moderator manually registered both the node of the robot and the node of the participant in each time step. After the task was completed, the moderator would also register the total number of steps it took the human and the robot to complete the task.

Selection of the task and starting positions

All tasks were chosen at random by the moderator before knowing the starting position of the robot and participants. Before placing the balls on the task's location nodes, the participants would be allowed to choose their own location and move

to their starting position. After the participants reached their starting position, the moderator manually controlled the robot to its randomly chosen starting position, ensuring it did not contain any balls. The trial would then start.

Choice of performing more than a single trial

We allowed participants who wished to run additional trials to do so, after completing their first trial. In such cases, these participants would perform a different task, excluding the performed one from being selected at random by the moderator. The same procedure would then be repeated. Out of the 7 participants recruited, only 2 decided each to run an additional trial.

Additional baseline approaches

Finally, to assess the introduced difficulties associated with live robotic experiments, we evaluate three agents in a simulated environment: (i) our own approach, TEBOPA, to compare how the results from the simulated environment differ from the ones obtained in the real-world experiments; (ii) an agent following an optimal policy, able to fully observe the state of the environment, used as a reference for best possible performance; (iii) an ablation of TEBOPA which knows the task beforehand, used as a reference for optimal behaviour under partial observability and (iv) an agent following a random policy, used as a reference for worst possible performance.

Results

Figure 7.8 shows the average number of steps our approach took to complete the task assigned to the team. It also plots the number of steps required for our approach to identify the correct task from two possible tasks. We also present the number of steps it takes for four baseline agents to run in a simulated environment, namely (i) an agent following an optimal policy, representing the best possible performance (ii) an agent following a random policy, representing the worst possible performance, (iii) our approach in the simulated environment, to compare with the results obtained in the real world, further validating both our simulated environment and human model, and (iv) an ablation of our approach which knows the task beforehand and therefore represents an optimal agent under partial observability.

From the results presented in Figure 7.8, we observe that our approach always completed the target task in a near-optimal number of steps and quickly identified the target task using only partial observations (without observing the human's actions). As expected, the agent following the optimal policy solved the tasks in fewer steps than the random policy under partial observability ($p = 0.0000018, \alpha = 0.05$), our approach, TEBOPA, ($p = 0.00757, \alpha = 0.05$) and the random policy ($p = 0.00000000007, \alpha = 0.05$). Our approach, TEBOPA, was able to not only solve tasks in fewer steps than the random policy ($p = 0.0000000001887, \alpha = 0.05$), but was also able to solve tasks in the same average number of steps as the optimal policy under partial observability ($p > \alpha = 0.05$). We can also observe that, on average, our approach can identify the correct task quicker than an optimal policy can solve it ($p = 0.0028, \alpha = 0.05$).

Furthermore, our results for the multiple participants, which were allowed to pick their starting nodes, do not hint at any benefits or disadvantages of starting at any specific node. Finally, we can observe that the results obtained for our approach in the simulated environment matched the results obtained by our approach in the real

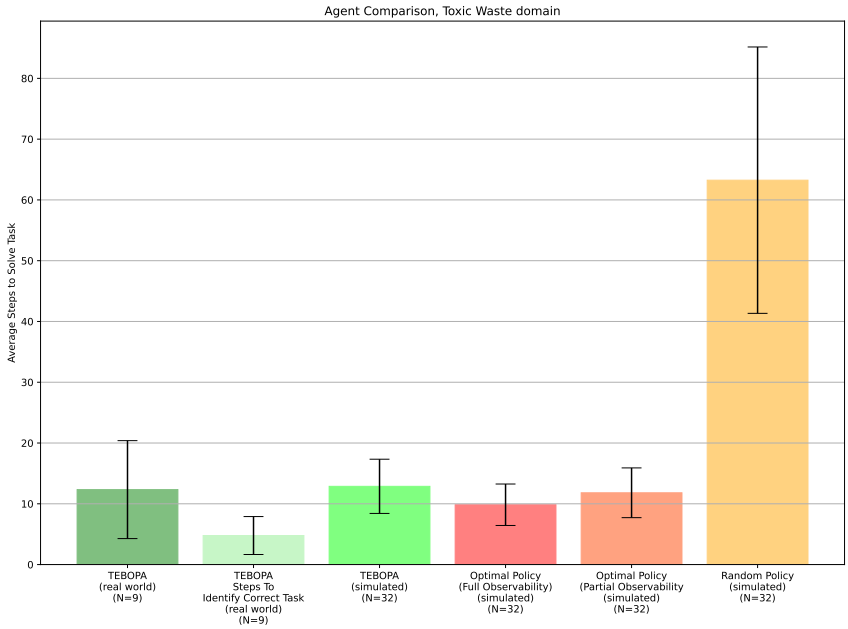


FIGURE 7.8: **Average number of steps required to solve and identify the target task was analyzed across different scenarios, considering the use of the communication module.** All simulated trials started from the same initial states as those in the live trials. Error bars correspond to a confidence interval with the confidence of 95% ($\alpha = 0.05\%$), calculated over all trials, encompassing 9 real-world trials and 32 trials in the simulated environment).

world, with no statistically significant difference ($p > \alpha = 0.05$). This demonstrates the validity of not only our simulated environment but also our model of the human teammate.

We can look deeper into the task identification by plotting the average entropy of the beliefs at each time step, as shown in Figure 7.9.

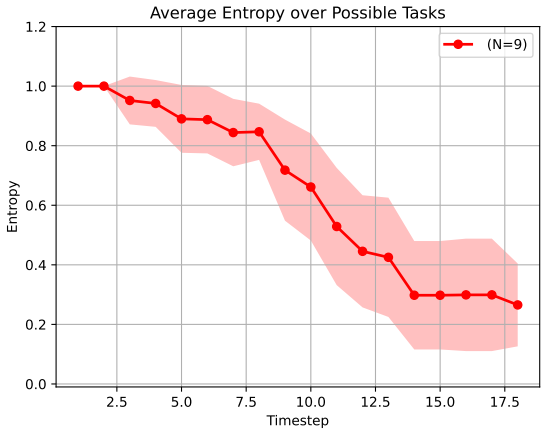


FIGURE 7.9: **Beliefs Entropy.** Entropy of the beliefs at each time step, averaged over all nine trials.

We can first observe in Figure 7.9 that the average entropy decreases with each passing time step. This is expected because the agent has more information to infer the correct task as the agent interacts with the environment. The second observation is that the average entropy does not reach 0.0, although it has dropped from 1.0 to almost 0.20. This result shows that our approach may end a trial without being

100% sure what the correct task is. However, this also indicates that our approach effectively solves the most likely task. Additionally, the fact that the average entropy is not 0.0 means that it may recover from task changes.

We also break down the evaluation of the NLP modules into three parts: i) the speech recognition module, ii) the NER module, and iii) the node identification module. Then, having recorded all the human's spoken phrases plus the outputs of the three modules, in all time steps of the trials, we start reporting the accuracy of the speech recognition module. We also recorded, for each sentence, whether or not the sentence contained the information necessary to identify the human's location. Finally, from this set of informative phrases, we evaluate the accuracy of the NER module and, subsequently, of the node identification module. The values depicted in Figure 7.10, which plots the accuracies for these three modules in a live environment, average over all iterations of the seven experiment participants.

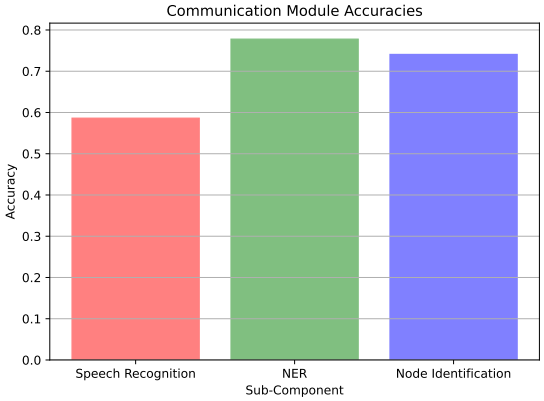


FIGURE 7.10: **NLP Modules Accuracies.** The accuracies for the three NLP modules - Speech Recognition (58.62%), NER (77.77%), and Node Identification (74.07%).

From Figure 7.10, we observe that perfectly recognizing human speech is the hardest task of the three, with only a performance of 58.62%. From the spoken phrases that are informative enough to infer the correct human location, NER was able to identify the correct location in 77.77% of them. At last, the node identification module, which takes as input the NER location string, was able to correctly output the right human nodes 74.07% of the time.

Additional Studies

In this section, we present two additional experiments to assess both the impact of information sharing on team performance and the scalability of our decision approach, TEBOPA, to larger domains.

Disabling Communication

To further evaluate the impact of communication on team performance, we conducted an additional experiment where we removed the possibility of communication between the robot and the human. The same modeling described in The Decision Module section holds, with the exception that there is now no *locate human* action available to the ad hoc robot. The same previous tasks are kept as well as the human model.

Given that the location of the human, when queried, is given to the ad hoc robot as part of its observation, this ablation can be seen as a setting where the available

information to the robot is reduced. Given that our approach deals with the task identification problem of ad hoc teamwork by using any information available, we expect that the less information we have, the better. To test this hypothesis, we evaluate a total of four approaches:

- TEBOPA: Our original ad hoc approach which doesn't know the correct task beforehand and is able to query the human for its location;
- Optimal policy under partial observability: A non-ad hoc ablation of our approach which knows the correct task beforehand and is able to query the human for its location;
- TEBOPA without communication: Our original ad hoc approach which doesn't know the correct task beforehand and doesn't have the action for locating the human within its action space;
- Optimal policy under partial observability without communication: A non-ad hoc ablation of our approach which knows the correct task beforehand and doesn't have the action for locating the human within its action space.

All agents are evaluated for 32 independent trials. We conduct this experiment in the simulated environment, given its shown reproducibility of real-world results in our main evaluation section. Figure 7.11 displays the average steps to solve the tasks over the 32 independent trials. We keep as a reference, the optimal policy (which has full observability of the environment) and random policy, for best and worst possible performances, respectively.

From these results, two main observations can be made. First, when the task is known, following the optimal policy (under partial observability) showcases no statistical difference when communication is disabled ($\alpha = 0.05$). This result is expected, since going directly to the nodes containing the waste and waiting for the human is an optimal strategy if the location of the balls is known beforehand, rendering the problem as a problem of self-location instead of coordination with another teammate. The second and most important observation we can make from these results is that when we're dealing with an unknown task in an ad hoc setting, removing the possibility of querying the human for its location yields a significant drop in performance, as shown by the results obtained by our approach, TEBOPA, and our approach without communication ($p = 0.000656, \alpha = 0.05$). The most plausible explanation for this observation is that the availability of the human's location in each time step allows the ad hoc robot to substantially enhance its belief in the correctness of the task being executed. Without communication, the robot is only able to observe where it is and when a ball is placed within its compartment, which happens at most three times throughout the episode and may not be enough information to immediately identify the correct task. These results show that without communication, the problem of ad hoc teamwork under partial observability becomes even harder.

Scaling to Larger Problems

We conclude our evaluation by conducting an additional experiment to assess the scalability of our decision module. The Toxic Waste domain has environments with a total of $|\mathcal{X}| = 260$ possible states, $|\mathcal{Z}| = 36$ possible observations, and $|\mathcal{A}_0| = 5$ actions. Given that these spaces are relatively small, we conduct an additional experiment in the Predator-Prey (or Pursuit) domain, where a team of predator

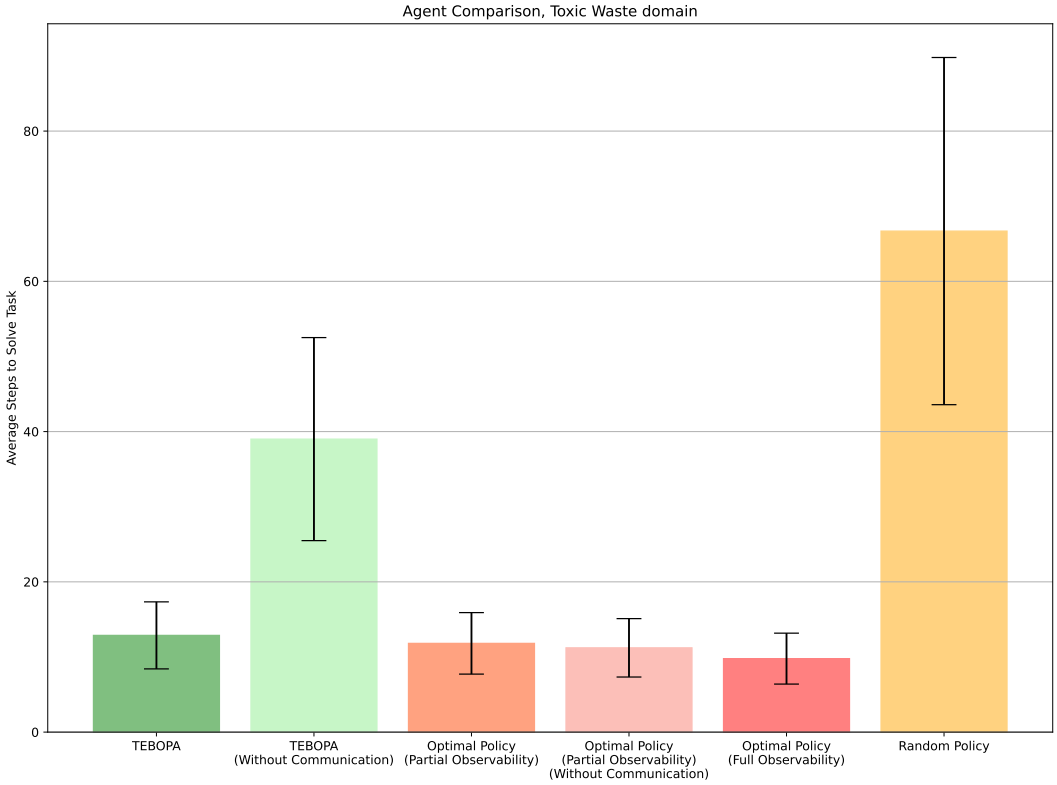


FIGURE 7.11: **Disabled Communication Experiment.** Average steps to complete a task for different approaches in the disabled communication experiment. Error bars correspond to a confidence interval with confidence of 95% ($\alpha = 0.05$), calculated over a total of 32 independent trials.

agents has the goal of capturing a moving prey. This domain is relatively larger for a team of two agents, with environments now with a total of $|\mathcal{X}| = 626$ states, $|\mathcal{Z}| = 81$ possible observations, and $|\mathcal{A}_0| = 6$ actions.

Similar to the Toxic Waste domain, we compare, in the Predator-Prey domain, our approach against (i) an optimal agent, which has full observability of the environment, (ii) an approach that knows the correct task and acts optimally under partial observability, and (iii) a random baseline. All approaches are evaluated in $N = 32$ independent trials. Figure 7.12 showcases the average steps to complete the tasks over the 32 trials. Tasks are selected randomly in each trial, and in the predator-prey domain, corresponding to the direction from which each predator must surround the prey for the team to capture it.

As expected, results show the optimal agent to have completed, on average, trials in 6.75 steps, solving episodes in fewer steps than the optimal agent under partial observability ($p = 0.007, \alpha = 0.05$), TEBOPA ($p = 0.00012, \alpha = 0.05$) and the random baseline ($p = 0.00000001, \alpha = 0.05$). The results for the optimal agent under partial observability and our approach, TEBOPA, show no statistical difference for $p > \alpha = 0.05$, showcasing similar performances by solving episodes in an average of 10.22 and 12.84 steps, respectively. They both solve episodes in fewer steps than the random baseline ($p = 0.00000004, \alpha = 0.05$ for the optimal agent under partial observability and $p = 0.000000088, \alpha = 0.05$ for TEBOPA). Finally, as expected, the random baseline took significantly longer steps to solve episodes, with an average of 76.06 steps. These results allow us to conclude that even in a significantly larger domain, our approach is able to correctly identify the correct task being performed

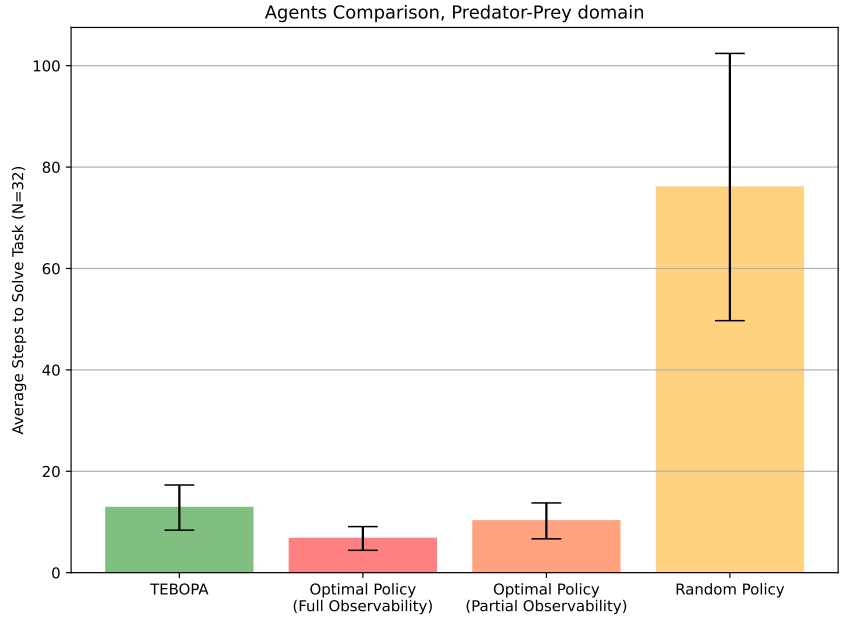


FIGURE 7.12: **Predator-Prey Domain Results.** Average number of steps to complete a task in the Predator-Prey domain. Error bars correspond to a confidence interval confidence of 95% ($\alpha = 0.05$), calculated over all trials.

by a teammate and solve it in a near-optimal number of steps.

7.2 Conclusion

In this chapter we addressed our fourth and final research sub-problem:

> *How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?*

We presented the HOTSPOT framework, a novel framework for ad hoc teamwork in human-robot teams. Specifically, our framework has two main modules, addressing the two key challenges in the interaction between a robot and a human teammate within ad hoc teamwork scenarios. The first module handles all the task-related decision-making challenges (i.e., task identification, teammate identification, and planning), and is responsible for orchestrating the robot's contributions to the collaborative effort. The second module deals with the communication challenge between robots and humans by employing NLP techniques, which enables the exchange of information between the robot and the human being, therefore enhancing the overall efficiency of the collaborative effort.

To evaluate our framework, we use a task that involves a mobile robot and a human teammate in a cooperative task of collecting objects in an open space, illustrating the main features of our framework in a real-world task. Our results show that our approach always identified and completed the task with a near-optimal number of steps while using partial and imperfect information. We also observe that, on average, the proposed approach identified the task faster than the optimal policy, showing the potential that this approach has in a real task environment and

highlighting the practicality and robustness of our framework in addressing the challenges of ad hoc teamwork.

Although our approach has shown excellent results, we can always incorporate enhancements to further improve the proposed methods' performance.

The main limitation of our work is its evaluation in a turn-based scenario, where the human and the robot act in turns. The next logical line of work would be to extend our approach to real-time tasks, where both the human and the robot are free to move when they want. In this sense, the exploration of real-time collaborative scenarios presents a promising avenue for future research and development, and adapting our framework to such dynamic environments will require the integration of responsive and adaptive decision-making processes that can handle the complexities of simultaneous human and robot actions. This aligns with the evolving landscape of human-robot interaction and will extend the applicability of the approach to real-world scenarios.

Another limitation is the speech recogniser, which currently uses an online recognition service, that is not customised for the restricted vocabulary used in human-robot conversation and is not suitable for the noisy environment of human-robot interactions. Moreover, besides the low average performance, any instability in the robot's Internet connection makes its use unfeasible. In this sense, developing an offline system customised for our domain would bring enormous advantages, both for the classifier performance and the response speed of the decision module.

Also, we plan to invest in other types of sensors for the robot, especially those that capture 360-degree scenes. This will empower the robot to gain a comprehensive view of its surroundings, enabling more robust and context-aware decision-making processes. That way, independent of the robot's position and orientation, it will be possible to apply computer vision techniques to enhance the robot's observation of the environment.

Chapter 8

Conclusion

Introduced by Stone et al. [2010b], the setting of *ad hoc* teamwork studies how autonomous agents may be able to adapt, on-the-fly, to unknown teams performing unknown tasks, without being able to pre-communicate or pre-coordinate with their teammates. Even though *ad hoc* teamwork has been under great attention by the multi-agent systems community [Mirsky et al., 2022], its approaches often rely on conditions often unattainable in real-world scenarios, such as full observability of the environment and access to the actions of others.

Motivated by the possible applications of *ad hoc* algorithms in real-world scenarios, this thesis contributes to the current literature by exploring the setting of *ad hoc* teamwork under conditions where (i) the environments are not fully observable, (ii) the actions others are executing in the environment are not visible to the *ad hoc* agent, (iii) environments might not only have with arbitrarily large state and/or observations spaces but also contain humans as teammates.

This thesis asked the following research question:

> *How can an autonomous agent assist unknown teammates in performing unknown tasks under more realistic conditions, without being able to pre-coordinate or pre-communicate with the existing team?*

Which it broke down into four sub-problems, each addressed in its dedicated chapter:

- How can an autonomous agent more efficiently assist new teammates and tasks on-the-fly, without having to learn how to interact with them from zero?
- How can an autonomous agent assist unknown teammates in performing unknown tasks without being able to observe their actions, the full state of the environment or both?
- How can an autonomous agent assist unknown teammates in performing unknown tasks in arbitrarily large, partially observable domains?
- How can an autonomous agent assist unknown mixed human-robot teams in performing unknown tasks in real-world scenarios?

From the study of each individual problem, five novel approaches for *ad hoc* teamwork were proposed:

- TEAMSTER (chapter 4), a model-based RL approach for *ad hoc* teamwork which allows an *ad hoc* agent to continuously adapt to new teammates and tasks by efficiently reusing environmental knowledge. Our results showed that TEAMSTER was not only more sample efficient than prior model-free approaches such as PLASTIC-Policy [Barrett et al., 2017] when adapting to new teams on-the-fly,

but also more robust than handcoded approaches such as PLASTIC-Model [Barrett, 2014] when adapting to changes in the task and world dynamics. This contribution is jointly published at the Artificial Intelligence Journal [Ribeiro et al., 2023b] and in the Proceedings of the 38th AAAI Conference on Artificial Intelligence [Ribeiro et al., 2024b].

- BOPA and ATPO (chapter 5), two Bayesian online approaches for *ad hoc* teamwork capable of identifying and assisting unknown teammates in performing unknown tasks without needing to observe their actions. BOPA, published at the Proceedings of the 20th EPIA Conference on Artificial Intelligence [Ribeiro et al., 2021] and winner of the conference's Best Paper Award, performs this inference by relying on the state of the environment and ATPO, published at the Proceedings of the 26th European Conference on Artificial Intelligence [Ribeiro et al., 2023a], expands upon BOPA by relying on partial observations instead.

- ATLPO (chapter 6), an approach capable of identifying and assisting unknown teammates in performing unknown tasks in arbitrarily large partially observable domains without needing to observe their actions. Two variants of ATLPO relying on state-of-the-art deep learning techniques were presented: (i) ATLPO-MF, which relies on a library of state-of-the-art model-free RL policies to assist the team and (ii) ATLPO-MB which relies on a library of state-of-the-art model-based RL world models. Our results show that even on larger domains, Bayesian inference provides an effective and efficient way to identify teammates and tasks on-the-fly.

- The HOTSPOT framework (chapter 7), an *ad hoc* teamwork platform for mixed human-robot teams that enables the deployment of *ad hoc* approaches into real-world scenarios. Our results show that (i) by relying on a modular architecture, an *ad hoc* agent can be deployed into a real-world robot, (ii) an *ad hoc* robot can effectively perform *ad hoc* teamwork in real-world domains where robots and humans cooperate and (iii) empirical results from simulated environments can be replicated in a real-world environment. This contribution is published at the Public Library of open Science (PLoS ONE) Journal [Ribeiro et al., 2024a].

8.1 Future Work

Approaches for *ad hoc* teamwork can be categorized into two classes, (i) *library-based*, also known as *type-based*, or (ii) *library-free*, also known as *type-free*. Library-based approaches, such as all introduced in this thesis, are not only considered more reliable due to their expandability to new situations, but they also have the advantage of explainability when the agent identifies the most-likely situation. However the size of a library grows exponentially with the number of possible teams and tasks, creating an additional scalability issue. In real-world environments where *ad hoc* robots operate this would require an exhaustive process where the *ad hoc* robot would need to go through a great deal of different teams and tasks. Although some recent lines of research have explored library-free methods [Fang et al., 2024], none have yet dealt with all conditions from real-world environments with humans in the loop such as the ones discussed in this thesis. This section proposes four novel lines of research that aim to break free of library-based methods.

8.1.1 Continual Multi-Task Learning For Ad Hoc Teamwork

One logical research direction for *ad hoc* teamwork that would break free of library-based methods would be to explore how advances in the fields of continual and multi-task learning [Wang et al., 2024c, Ribeiro et al., 2019] can be used to set up a single continual learning model where knowledge from all teams and tasks can be efficiently consolidated. An *ad hoc* agent continuously trained using these methods would not only be able to explicitly identify the current situation while having a single library-free model, but also be able to continuously update it using new experiences. Furthermore, if provided with an initial library of prior experiences with labels for the respective teams and tasks, training a single multi-task model also has the advantage of requiring significantly less data due to parameter sharing. Afterwards, continual learning methods can be employed to incorporate new data effectively and prevent problems such as catastrophic forgetting.

8.1.2 Bayesian Meta-Learning For Ad Hoc Teamwork

One recent work in the field of Meta-Learning has shown that by employing Bayesian RL techniques allows agents to more effectively plan for an action in unknown environments [Zintgraf et al., 2019]. The authors show how the identification of the most likely environment can be used as context to create strong task embeddings, making the planning process of the next action more robust. If current *ad hoc* approaches would be provided with similar task embeddings instead of directly identifying the most likely team and task from their library, they would not only become more robust to unknown teams, but would also become more effective in planning their actions. In a recent line of research, Fang et al. [2024] have already explored how Meta-Learning can be employed to avoid creating a library of teammate types. However, besides leaving unexplored the possibilities for Bayesian approaches, the authors also assume teammates to not have different roles. Expanding upon their work and combining Bayesian RL with Meta-Learning for *ad hoc* teamwork is therefore a very promising line of research to explore.

8.1.3 Offline Reinforcement Learning for Ad Hoc Teamwork

In recent years, model-based RL approaches have been increasingly falling under attention by both the RL [Schrittwieser et al., 2020b, Hafner et al., 2023a] and the *ad hoc* teamwork communities [Ribeiro et al., 2024b]. Unlike model-free RL algorithms [Schulman et al., 2017], which directly train policy mappings from observations into actions, model-based RL algorithms learn instead world models of the environment. These models may not only be combined with planning algorithms [Kocsis and Szepesvári, 2006] to search for the best actions but also require significantly less data than model-free algorithms to effectively train. As shown in Chapter 4, model-based methods bring significant advantages to *ad hoc* teamwork when it comes to efficiently sharing world knowledge between different teams and tasks. The key limitation with these approaches is their expensive computation time, in particular during training when models are being optimised and used by the planners as if perfect predictors.

In real-world environments where data from human interactions is abundantly available, employing techniques that train the world models offline using existing datasets [Schrittwieser et al., 2021] could potentially mitigate the problem of expensive computation times and wasteful planning iterations, allowing for a much efficient pre-training of *ad hoc* agents.

Bibliography

Noa Agmon and Peter Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 341–348. Citeseer, 2012.

Stefano V Albrecht and Subramanian Ramamoorthy. Comparative evaluation of mal algorithms in a diverse set of ad hoc team problems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 349–356, 2012.

Christopher Amato, Jilles Steeve Dibangoye, and Shlomo Zilberstein. Incremental policy generation for finite-horizon dec-pomdps. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

A. Banerjee. On Bayesian bounds. In *Proc. 23rd Int. Conf. Machine Learning*, pages 81–88, 2006.

S. Barrett. *Making friends on the fly: Advances in ad hoc teamwork*. PhD thesis, The University of Texas at Austin, 2014.

S. Barrett and P. Stone. Ad hoc teamwork modeled with multi-armed bandits: An extension to discounted infinite rewards. In *Proc. AAMAS Workshop on Adaptive Learning Agents*, 2011.

S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proc. 10th Int. Conf. Autonomous Agents and Multiagent Systems*, pages 567–574, 2011.

Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

Samuel Barrett, Noa Agmon, Noam Hazon, Sarit Kraus, and Peter Stone. Communicating with unknown teammates. In *ECAI*, pages 45–50, 2014.

Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171, 2017.

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

G. Briggs, T. Williams, and M. Scheutz. Enabling robots to understand indirect speech acts in task-based interactions. *J. Human-Robot Interaction*, 6:64–94, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Kalesha Bullard, Douwe Kiela, Joelle Pineau, and Jakob Foerster. Quasi-equivalence discovery for zero-shot emergent communication. *arXiv preprint arXiv:2103.08067*, 2021.

M. Carroll, R. Shah, M. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan. On the utility of learning about humans for human-AI coordination. In *Adv. Neural Information Processing Systems 32*, 2019a.

Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019b.

Anthony R Cassandra, Michael L Littman, and Nevin Lianwen Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. *arXiv preprint arXiv:1302.1525*, 2013.

Doran Chakraborty and Peter Stone. Cooperating with a markovian ad hoc teammate. In *AAMAS*, pages 1085–1092, 2013.

Gang Chen. A new framework for multi-agent reinforcement learning–centralized training and exploration with decentralized execution via policy distillation. *arXiv preprint arXiv:1910.09152*, 2019.

Shuo Chen, Ewa Andrejczuk, Athirai Aravazhi Irissappane, and Jie Zhang. Atsis: Achieving the ad hoc teamwork by sub-task inference and selection. In *IJCAI*, pages 172–179, 2019.

Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. Aateam: Achieving the ad hoc teamwork by employing the attention mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7095–7102, 2020.

Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Jilles S Dibangoye, Abdel-Illah Mouaddib, and Brahim Chai-draa. Point-based incremental pruning heuristic for solving finite-horizon dec-pomdps. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 569–576, 2009.

Qi Fang, Junjie Zeng, Haotian Xu, Yue Hu, and Quanjun Yin. Learning ad hoc cooperation policies from limited priors via meta-reinforcement learning. *Applied Sciences*, 14(8):3209, 2024.

Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A decision-theoretic model of assistance. In *IJCAI*, pages 1879–1884, 2007.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Elliot Fosong, Arrasy Rahman, Ignacio Carlucho, and Stefano V Albrecht. Few shot teamwork. *arXiv preprint arXiv:2207.09300*, 2022.

Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. MIT Press, 2000. ISBN 0262061945. URL http://www.amazon.com/dp/0262061945.

K. Genter, T. Laue, and P. Stone. Three years of the RoboCup standard platform league drop-in player competition. *J. Autonomous Agents and Multi-Agent Systems*, 31(4):790–820, 2017.

Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. *Advances in neural information processing systems*, 14, 2001.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019b.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023a.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy P. Lillicrap. Mastering diverse domains through world models. *CoRR*, abs/2301.04104, 2023b. doi: 10.48550/ARXIV.2301.04104.

Ravi Hammond, Dustin Craggs, Mingyu Guo, Jakob Foerster, and Ian Reid. Symmetry-breaking augmentations for ad hoc teamwork. *arXiv preprint arXiv:2402.09984*, 2024.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

Yujing Hu, Yang Gao, and Bo An. Learning in multi-agent systems with sparse interactions by knowledge transfer and game abstraction. In *AAMAS*, pages 753–761, 2015.

Y. Kilicaslan and G. Tuna. An NLP-based approach for improving human-robot interaction. *J. Artificial Intelligence and Soft Computing Research*, 3(3):189–200, 2014.

D. Kingma and J. Ba. ADAM: A method for stochastic optimization. In *Proc. 3rd Int. Conf. Learning Representations*, 2014.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Z. Li, Y. Mu, Z. Sun, S. Song, J. Su, and J. Zhang. Intention understanding in human-robot interaction based on visual-NLP semantics. *Frontiers in Neurorobotics*, 14 (610139), 2021.

M. Littman. Value-function reinforcement learning in Markov games. *J. Cognitive Systems Res.*, 2(1):55–66, 2001.

Michael L Littman. The witness algorithm: Solving partially observable markov decision processes. *Brown University, Providence, RI*, 1994.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *International Conference on Machine Learning*, pages 7204–7213. PMLR, 2021.

Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.04402*, 2021.

C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

F. Melo and M.I. Ribeiro. Transition entropy in partially observable Markov decision processes. In *Proc. 9th Int. Conf. Intelligent Autonomous Systems*, pages 282–289, 2006.

F. Melo, A. Sardinha, D. Belo, M. Couto, M. Faria, A. Farias, H. Gambôa, C. Jesus, M. Kinarullathil, P. Lima, L. Luz, A. Mateus, I. Melo, P. Moreno, D. Osório, A. Paiva, J. Pimentel, J. Rodrigues, P. Sequeira, R. Solera-Ureña, M. Vasco, M. Veloso, and R. Venturab. Project INSIDE: towards autonomous semi-unstructured human–robot social interaction in autism therapy. *Artificial Intelligence in Medicine*, 596:198–216, 2019.

Francisco S Melo and Alberto Sardinha. Ad hoc teamwork by learning teammates' task. *Autonomous Agents and Multi-Agent Systems*, 30(2):175–219, 2016.

Francisco S Melo and Manuela Veloso. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 773–780. Citeseer, 2009.

Reuth Mirsky, Ignacio Carlucho, Arrasy Rahman, Elliot Fosong, William Macke, Mohan Sridharan, Peter Stone, and Stefano V Albrecht. A survey of ad hoc teamwork: Definitions, methods, and open problems. *arXiv preprint arXiv:2202.10450*, 2022.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Frans A Oliehoek, Matthijs TJ Spaan, Jilles Steeve Dibangoye, and Christopher Amato. Heuristic search for identical payoff bayesian games. In *AAMAS*, pages 1115–1122, 2010.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*, pages 2681–2690. PMLR, 2017.

Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. *J. Artificial Intelligence Res.*, 27:335–380, 2006.

M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 2005.

Arrasy Rahman, Ignacio Carlucho, Niklas Höpner, and Stefano V Albrecht. A general learning framework for open ad hoc teamwork using graph-based policy learning. *Journal of Machine Learning Research*, 24(298):1–74, 2023.

Muhammad Rahman, Jiaxun Cui, and Peter Stone. Minimum coverage sets for training robust ad hoc teamwork agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17523–17530, 2024.

Muhammad A Rahman, Niklas Hopner, Filippos Christianos, and Stefano V Albrecht. Towards open ad hoc teamwork using graph-based policy learning. In *International Conference on Machine Learning*, pages 8776–8786. PMLR, 2021.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 4295–4304. PMLR, 2018.

Joao Ribeiro, Francisco S Melo, and Joao Dias. Multi-task learning and catastrophic forgetting in continual reinforcement learning. *arXiv preprint arXiv:1909.10008*, 2019.

João G Ribeiro, Miguel Faria, Alberto Sardinha, and Francisco S Melo. Helping people on the fly: Ad hoc teamwork for human-robot teams. In *Progress in Artificial Intelligence: 20th EPIA Conference on Artificial Intelligence, EPIA 2021, Virtual Event, September 7–9, 2021, Proceedings 20*, pages 635–647. Springer, 2021.

João G Ribeiro, Cassandro Martinho, Alberto Sardinha, and Francisco S Melo. Making friends in the dark: Ad hoc teamwork under partial observability. In *ECAI 2023*, pages 1954–1961. IOS Press, 2023a.

João G Ribeiro, Gonçalo Rodrigues, Alberto Sardinha, and Francisco S Melo. Teamster: Model-based reinforcement learning for ad hoc teamwork. *Artificial Intelligence*, 324:104013, 2023b.

João G Ribeiro, Luis Müller Henriques, Sérgio Colcher, Julio Cesar Duarte, Francisco S Melo, Ruy Luiz Milidiú, and Alberto Sardinha. Hotspot: An ad hoc teamwork platform for mixed human-robot teams. *Plos one*, 19(6):e0305705, 2024a.

João G Ribeiro, Gonçalo Rodrigues, Alberto Sardinha, and Francisco S Melo. Teamster: Model-based reinforcement learning for ad hoc teamwork (abstract reprint). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 22708–22708, 2024b.

Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

M. Scheutz, R. Cantrell, and P. Schermerhorn. Toward human-like task-based dialogue processing for human robot interaction. *AI Magazine*, 32(4):77–84, 2011.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020a.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020b.

Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *Advances in Neural Information Processing Systems*, 34:27580–27591, 2021.

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

Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for dec-pomdps. In *IJCAI*, pages 2009–2015, 2007.

Elnaz Shafipour Yourdshahi, Matheus Do Carmo Alves, Leandro Soriano Marcolino, and Plamen Angelov. Decentralised task allocation in the fog: Estimators for effective ad-hoc teamwork. In *11th International Workshop on Optimization and Learning in Multiagent Systems*, 2020.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Ho Chit Siu, Jaime Peña, Edenna Chen, Yutai Zhou, Victor Lopez, Kyle Palko, Kimberlee Chang, and Ross Allen. Evaluation of human-ai teams for learned and rule-based agents in hanabi. *Advances in Neural Information Processing Systems*, 34: 16183–16195, 2021.

Matthijs TJ Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of artificial intelligence research*, 24:195–220, 2005.

Matthijs TJ Spaan, Frans A Oliehoek, and Christopher Amato. Scaling up optimal heuristic search in dec-pomdps via incremental expansion. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

Stanford Artificial Intelligence Laboratory. Robotic Operating System, 2018. URL https://www.ros.org.

P. Stone, G. Kaminka, S. Kraus, and J. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proc. 24th AAAI Conf. Artificial Intelligence*, pages 1504–1509, 2010a.

Peter Stone and Sarit Kraus. To teach or not to teach?: Decision making under uncertainty in ad hoc teams. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 117–124, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9826571-1-9. URL http://dl.acm.org/citation.cfm?id=1838206.1838223.

Peter Stone, Gal A Kaminka, and Jeffrey S Rosenschein. Leading a best-response teammate in an ad hoc team. In *Agent-mediated electronic commerce. Designing trading strategies and mechanisms for electronic markets*, pages 132–146. Springer, 2009.

Peter Stone, Gal A Kaminka, Sarit Kraus, and Jeffrey S Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010b.

Peter Stone, Gal A. Kaminka, and Jeffrey S. Rosenschein. Leading a best-response teammate in an ad hoc team. *Lecture Notes in Business Information Processing*, 59 LNBIP(May):132–146, 2010c. ISSN 18651348. doi: 10.1007/978-3-642-15117-0{_}10.

DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34:14502–14515, 2021.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

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

Daniel Szer and Francois Charpillet. Point-based dynamic programming for dec-pomdps. In *AAAI*, volume 6, pages 1233–1238, 2006.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

Johannes Treutlein, Michael Dennis, Caspar Oesterheld, and Jakob Foerster. A new formalism, method and open issues for zero-shot coordination. *arXiv preprint arXiv:2106.06613*, 2021.

Caroline Wang, Arrasy Rahman, Ishan Durugkar, Elad Liebman, and Peter Stone. N-agent ad hoc teamwork. *arXiv preprint arXiv:2404.10740*, 2024a.

Jianhong Wang, Yang Li, Yuan Zhang, Wei Pan, and Samuel Kaski. Open ad hoc teamwork with cooperative game theory. *arXiv preprint arXiv:2402.15259*, 2024b.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024c.

Rose E Wang, Sarah A Wu, James A Evans, Joshua B Tenenbaum, David C Parkes, and Max Kleiman-Weiner. Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *arXiv preprint arXiv:2003.11778*, 2020.

C. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.