



**Monte-Carlo Based Online
planning Under Partial
Observability: Solving Single and
Multi-Agent Problems**

Matheus Aparecido do Carmo Alves, BSc

Computer Science

School of Computing and Communications

Lancaster University

A thesis submitted for the degree of

Doctor of Philosophy

August, 2024

Monte-Carlo Based Online planning Under Partial Observability: Solving Single and Multi-Agent Problems

Matheus Aparecido do Carmo Alves, BSc Computer Science.

School of Computing and Communications, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*. August, 2024.

Abstract

This thesis thoroughly explores the integration of statistical and reinforcement learning techniques, aiming to provide fresh perspectives and solutions for enhancing the current state-of-the-art methods considering the capabilities of autonomous agents to perform learning, planning and estimation in an online manner in a single and multi-agent systems context. We aim to address a critical demand in the field, steering away from the prevailing dependence on the application of intensive computational resources and large amounts of data as a requirement to achieve peak performance in our context. Our primary focus centres on studying and refining solutions in the “online planning under uncertainty” research area. We have ventured beyond the boundaries of existing literature, pushing our proposals to more complex and challenging problems. As concrete contributions, we introduce three new algorithms: IB-POMCP, an online planning algorithm which uses information entropy to augment a single agent’s decision making capabilities; OEATE, a type and parameter estimation method to handle coordination with multiple unknown teammates in cooperative environments; and BAE, a method capable of detecting adversarial agents disguised as teammates in cooperative environments on-the-fly. Our proposals contribute to the evolution of autonomous systems and are supported by empirical and theoretical results. We demonstrate that our new perspectives for

agents' reasoning processes can present generic and extendable solutions to diverse scenarios and problems. Finally, during the PhD journey, we have developed and presented to the research community a new framework designed to aggregate relevant baselines and benchmarks for multi-agent systems: the AdLeap-MAS. AdLeap-MAS framework stands out as a novel tool centred on the implementation and simulation of ad-hoc reasoning domains for multi-agent, collaborative, and adversarial contexts. The framework aims to facilitate the execution of experiments and the re-use existing codes across different environments. We provide a user-friendly environment that not only extends the frontiers of our research but also serves as a valuable resource for the research community.

Acknowledgements

I want to express my sincere gratitude to my supervisor, Dr Leandro Soriano Marcolino, lecturer at Lancaster University, and co-supervisor, Dr Yehia Elkathib, which is now lecturing at Glasgow University, for their huge support, invaluable guidance, and insightful feedback throughout the entire journey of my PhD. Their expertise and encouragement have been essential in my research, as it has been for my personal and professional development.

I am also deeply thankful to my colleagues at Lancaster University for their collaboration, shared insights, and companionship. The intellectual exchange within the university community has enriched my understanding and contributed to the success of this work. Even working remotely from Brazil, sharing most of my experiences virtually with them, I will never forget all the moments we shared.

Sincere appreciation goes to the PhD Studentship provided by Lancaster University, which has significantly supported and facilitated the realisation of this research project. The resources and opportunities made available through this grant have played a crucial role in not only ensuring the successful completion of my PhD but also enabling the publication of pertinent research contributions to the community I have actively engaged with since the beginning of this academic journey. The impact of this support extends beyond my individual accomplishment, contributing to the broader academic dialogue and fostering meaningful connections within the community for me and my group as a whole.

I extend my gratitude to the High-End Computing (HEC) cluster, especially Dr Mike Pacey, our senior High-Performance Computing (HPC) Manager, for guiding and adjusting the necessary computational resources required for all the complex simulations and analyses carried out in this research. The accessibility to advanced computing resources has been indispensable to achieving this project's goals.

I would also like to acknowledge the dedicated staff at the Faculty of Science and Technology (FST) of Lancaster University who have supported me in different degrees throughout my PhD. Their professionalism and assistance have played a

vital role in overcoming challenges and ensuring the progression of my research.

A deeply heartfelt thank you to my family for their unwavering support, understanding, and encouragement. Your steadfast belief in my abilities and unwavering devotion has been an enduring source of inspiration, propelling me forward along the path I have chosen. As the first in our family to aspire to a PhD qualification, I dedicate this significant achievement to each of you. Your love and encouragement have been the pillars that sustained me through the challenges, and I am profoundly grateful for the strength you have invested in me. Special thanks to Ana Lucia Godoy do Carmo and Marcio Daniel Simões Alves, my parents, Ignez Aparecida Godoy do Carmo, my grandmother, Danilo Aparecido do Carmo Alves and Benedito Vanelli do Carmo Neto, my brothers.

Similar to family, I need to express my deepest gratitude to all my close friends who have been a source of emotional support, offering encouragement and understanding during both the highs and lows of this academic journey.

Special thanks to my partner, Maira Yumi Fumiya, and my friends, João Pedro Ramos Belmiro and João Vítor Nasevicius Ramos, for affording me the opportunity to embark on my PhD journey and for extending the greatest possible support, both financially and emotionally, during the challenging times of the COVID-19 pandemic. I am still unable to fully express my gratitude for all your assistance, which continues to persist and profoundly impact my steps to this day.

Finally, I dedicate this thesis in loving memory of my grandfather, Benedito Vanelli do Carmo, whose guiding presence and inspiration accompany me every day and night, from the brightest sky to the most beautiful sea of stars. I trust that you persist in following your grandson's footsteps, as you always did, and it remains my sincere aspiration to make you prouder with each passing day.

This thesis is a reflection of all the collective effort and support of these individuals and institutions. I am truly grateful for their contributions to my academic and personal growth. Thank you everyone!

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 56466

Matheus Aparecido do Carmo Alves

Publications

All following publications, shown below, have been generated while developing this thesis and contribute to this thesis writing. The text in these works guided the thesis into what it has become, offering support to our findings through my PhD.

In summary, we have published 5 works in renowned and impactful scientific venues:

- *International Conference on Autonomous Agents and Multi Agents Systems (AAMAS)*: one of the most prominent conferences for Multi-Agent Systems research, a Top 94 venue in the Computer Sciences category with h5-index of 55 (according Google Scholar), ranked as an A1 conference by Qualis.
- *Journal of Autonomous Agents and Multi Agent Systems (JAAMAS)*: the official journal of the International Foundation for Autonomous Agents and Multi-Agent Systems (IFAAMAS), covering foundational, theoretical, and practical aspects of the field. It is a prestigious journal in the Multi-Agent community and has a 5 year impact factor of 2.5 (2022) according Springer.
- *Neural Information Processing Systems (NeurIPS)*: a Top 1 Conference for Artificial Intelligence solutions, Top 3 venue in the Engineering & Computer Science category and Top 9 for all categories with h5-index of 309 (according Google Scholar), ranked as an A1 conference by Qualis.

► Demo Paper

Matheus Aparecido do Carmo Alves, Amokh Varma, Yehia Elkhatib, and Leandro Soriano Marcolino. “AdLeap-MAS: An Open-Source Multi-Agent Simulator for Ad-Hoc Reasoning”. In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '22. Virtual Event, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2022, pp. 1893–1895. ISBN: 9781450392136. URL: <https://dl.acm.org/doi/10.5555/3535850.3536143>

► **AAMAS Full Paper**

Matheus Aparecido do Carmo Alves, Elnaz Shafipour Yourdshahi, Amokh Varma, Leandro Soriano Marcolino, Jó Ueyama, and Plamen Angelov. “On-line estimators for ad-hoc task execution: learning types and parameters of teammates for effective teamwork”. In: *Autonomous Agents and Multi-Agent Systems* 36.2 (2022), p. 45. ISSN: 1573-7454. DOI: <https://doi.org/10.1007/s10458-022-09571-9>. URL: <https://link.springer.com/article/10.1007/s10458-022-09571-9>

► **JAAMAS Track Paper**

Matheus Aparecido do Carmo Alves, Elnaz Shafipour Yourdshahi, Amokh Varma, Leandro Soriano Marcolino, Jó Ueyama, and Plamen Angelov. “On-Line Estimators for Ad-Hoc Task Execution: Learning Types and Parameters of Teammates for Effective Teamwork”. In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '23. London, United Kingdom: International Foundation for Autonomous Agents and Multiagent Systems, 2023, pp. 140–142. ISBN: 9781450394321. URL: <https://dl.acm.org/doi/10.5555/3545946.3598629>

► **NeurIPS Full Paper**

Matheus Aparecido do Carmo Alves, Amokh Varma, Yehia Elkhatib, and Leandro Soriano Marcolino. “Information-guided Planning: An Online Approach for Partially Observable Problems”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. New Orleans Ernest N. Morial Convention Center, Nova Orleans, Louisiana, EUA, Dec. 2023

► **AAMAS Full Paper**

Matheus Aparecido do Carmo Alves, Amokh Varma, Yehia Elkhatib, and Leandro Soriano Marcolino. “It Is Among Us: Identifying Adversaries in Ad-hoc Domains Using Q-valued Bayesian Estimations”. In: *Proceedings of the 23rd*

International Conference on Autonomous Agents and Multiagent Systems. AAMAS
'24. Accepted for publication. Auckland, New Zealand: International Foundation for
Autonomous Agents and Multiagent Systems, 2024

Contents

1	Introduction	1
1.1	Autonomous Agents and the Advancement of Artificial Intelligence . . .	1
1.2	Making Decisions while Handling Uncertainty	4
1.3	Reinforcement Learning: Developing Reward-guided Planning Methods	7
1.4	Teamwork and Decentralised Execution	9
1.5	Our Research Questions and Contributions	11
1.6	Additional Contribution: AdLeap-MAS	13
1.7	Thesis Content	14
2	Background	15
2.1	Markovian World Representation	16
2.1.1	Markov Decision Problems	17
2.1.2	Partially Observable Markov Decision Problems	18
2.1.3	Multi-agents as part of the environment	19
2.1.4	Multi-Agent Markov Decision Problem	20
2.2	Tree-Search Methods	21
2.2.1	Monte Carlo Tree-Search	23
2.2.2	Partially Observable Monte Carlo Tree-Search	28
2.2.3	Single-agent Model Approach for Ad-hoc Teamwork	30
2.2.4	Adversarial Tree-search-based Approach	32
2.3	Information Analysis	33
2.3.1	Shannon’s Entropy	34

2.3.2	Bayesian Inference	35
2.4	Estimation Methods	36
3	Related Works	39
3.1	Online Planning under Uncertainty	40
3.1.1	Latent function approximation	41
3.1.2	Belief-dependent approaches	42
3.1.3	Time-constrained planning	43
3.1.4	Entropy-based methods	44
3.1.5	Information-based Sampling Approaches	45
3.1.6	Other options to POMCP	46
3.2	Teamwork and Ad-hoc Teamwork Literature	47
3.2.1	Type-based parameter estimation	47
3.2.2	Complex models application	48
3.2.3	Task-oriented approaches	50
3.2.4	MRTA model for ad-hoc teamwork	51
3.2.5	Genetic algorithms' inspiration for OEATE	52
3.3	Adversarial Identification among a Team	52
3.3.1	Task-based algorithms and teamwork models in the adversarial detection context	53
3.3.2	Neural network solutions for adversary detection	54
4	Information-based Partially Observable Monte-Carlo Planning	56
4.1	Introduction	56
4.2	Information-based Partially Observable Monte Carlo Planning	58
4.2.1	Algorithm Outline	58
4.2.2	Root Initialisation and Update	60
4.2.3	Particle Filter Initialisation and Update	61
4.2.4	Updating the Tree Exploration Coefficient	62
4.2.5	Simulation	66

4.2.6	Best action selection	68
4.2.7	Algorithm and Practical Enhancement	68
4.3	Theoretical Analysis	68
4.4	Results	74
4.4.1	Evaluation Settings	74
4.4.2	Benchmarks study	84
4.4.3	Ablation study	87
4.5	Further Details and Discussion	89
4.6	Chapter Conclusion	91
5	Online Estimators for Ad-hoc Task Execution	92
5.1	Introduction	92
5.2	The Task-based Ad-hoc Teamwork Problem	95
5.3	The Markovian Extension for Task-based Ad-hoc Teamwork Problems	97
5.4	The Estimation Problem	100
5.5	POMCP as an Estimation Method	102
5.6	Online Estimators for Ad-hoc Task Execution	103
5.6.1	OEATE Fundamentals	103
5.6.2	Process of Estimation	107
5.6.2.1	Initialisation	108
5.6.2.2	Evaluation	110
5.6.2.3	Generation	113
5.6.2.4	Estimation	117
5.6.2.5	Update	119
5.7	OEATE with Partial Observability	121
5.7.1	POMDP model	121
5.7.2	POMCP modification	122
5.8	Theoretical Analysis	123
5.9	Study Benchmarks	127
5.9.1	Level-based Foraging Domain	127

5.9.2	Capture the Prey Domain	131
5.10	Results	135
5.10.1	Evaluation Settings	135
5.10.2	Level-based Foraging Results	137
5.10.3	Capture the Prey Results	145
5.11	Chapter Conclusion	151
6	Bayesian Adversary Estimation	152
6.1	Introduction	152
6.2	Problem Formalisation and Background	154
6.3	Online planning using Q-valued Bayesian Estimations	158
6.3.1	Initialisation	158
6.3.2	Search and Q-table Extraction	159
6.3.3	Q-table Translation and our Bayesian Update	162
6.3.4	Algorithm Outline	163
6.4	Evaluation Settings	165
6.5	Empirical Results and Discussion	173
6.5.1	Adversarial Detection	173
6.5.2	Ad-hoc Adversary Detection	176
6.5.3	Additional Plots for Results Visualisation	177
6.6	Chapter Conclusion	181
7	Conclusions	182
7.1	PhD Outline	182
7.2	Contributions	186
7.2.1	IB-POMCP	187
7.2.2	OEATE	188
7.2.3	BAE	189
7.3	Limitations	190
7.3.1	IB-POMCP	190

7.3.2	OEATE	191
7.3.3	BAE	192
7.4	Future Works	193
7.4.1	IB-POMCP	193
7.4.2	OEATE	194
7.4.3	BAE	195
Appendix A AdLeap-MAS: An Open-Source Multi-Agent Simulator		
for Ad-hoc Reasoning		196
A.1	Introduction	196
A.2	Related Work	199
A.3	The AdLeap-MAS Framework	202
A.4	Framework Architecture	203
A.4.1	High level view	203
A.4.2	Implementation	206
A.4.2.1	Background	206
A.4.2.2	Components Design	207
A.5	Performance and Results	212
A.6	Conclusion	215
References		216

List of Tables

4.1	Average reward result for the baselines in benchmark problems. The bold highlighted values in the Quick Planning category highlight the best result for the respective problem and metric with statistical significance across all Quick Planning baselines. The rewards order are $\times 10^{-2}$, except for the Foraging scenarios (F0-4).	84
4.2	Average time result for the baselines in benchmark problems. The bold highlighted values in the Quick Planning category highlight the best result for the respective problem and metric with statistical significance across all Quick Planning baselines. The time is expressed in seconds.	85
5.1	OEATE’s Initialisation example. <i>Estimator</i> sets $\mathbf{E}_{\omega_1}^{\theta_1}$ and $\mathbf{E}_{\omega_1}^{\theta_2}$ obtained from the <i>Initialisation</i> step.	109
5.2	Evaluation example. <i>Estimator</i> sets $\mathbf{E}_{\omega_1}^{\theta_1}$, $\mathbf{E}_{\omega_1}^{\theta_2}$ after updating c_e and f_e	112
5.3	Evaluation example. <i>Estimator</i> sets $\mathbf{E}_{\omega_1}^{\theta_1}$, $\mathbf{E}_{\omega_1}^{\theta_2}$ after <i>Evaluation</i> step.	113
5.4	Generation example. <i>Estimator</i> sets $\mathbf{E}_{\omega_1}^{\theta_1}$ and $\mathbf{E}_{\omega_1}^{\theta_2}$ sets after the <i>Generation</i>	117
5.5	Summary of our experiments and results.	149
6.1	Agents’ details in the small scenario (LBF.a).	166
6.2	Task’ details in the small scenario (LBF.a).	167
6.3	Agents’ details in the medium scenario (LBF.b).	167
6.4	Tasks’ details in the medium scenario (LBF.b).	167

6.5	Agents' details in the big scenario (LBF.c).	169
6.6	Tasks' details in the big scenario (LBF.c).	169
6.7	Agents' details in the cooperative scenario (LBF.d).	169
6.8	Tasks' details in the cooperative scenario (LBF.d).	169
6.9	Real and template types per scenario for the Adversarial Detection experimental setup. Note that each type in the "Real Types" column refers to the true type of each agent in the scenario, and all types in the "Template Types" column will be used to approximate each agent's behaviour.	171
6.10	Summarised results for each adversary detection approach. The highlighted values indicate when a method presents statistical significance in its results among all competitors.	173
6.11	Summarised results for the Ad-hoc Adversary Detection experimental setup. The highlighted values indicate instances when AGA-BAE improves AGA's results and when ABU-BAE improves ABU's results with statistical significance.	176

List of Figures

2.1	Illustration of MCTS's tree structure. Each node represents a state s . Some states are specified in the figure to highlight this difference. Each node is connected to another via an action $A = \{a_1, a_2\}$. N represents the number of visits and V the current value estimated to each specific node. Each box at the bottom of the figure presents the reward r found in the last node.	24
2.2	Illustration of POMCP's tree structure. The red circles indicate action nodes and the blue circles indicate observation nodes. Action nodes are connected to observations nodes through actions and observation nodes are connected action nodes through observations. B represents the belief state set of the root node (each number represent a possible state). N represents the number of visits and V the current value estimated to each specific node. Each box at the bottom of the figure presents the reward r found in the last node.	29
4.1	Illustration of IB-POMCP's search process. The red circles represent states stored in the particle filter of a node. The blue circles represent observations stored in the observation set of a node. In the simulation step, we add states to the particle filter of nodes in the path. In the back-propagation step, we add the observations to each node in the path.	65
4.2	Maze environment scenario's configuration.	76

4.3	RockSample environment scenario's configuration.	78
4.4	Tag and LaserTag scenarios' configuration.	79
4.5	Foraging environment scenario's configuration.	82
4.6	Ablation study of IB-POMCP in four different scenarios.	88
5.1	Illustration of OEATE's set of estimators. Note that, for each ω agent there is a set of <i>estimators</i> \mathbf{E}_ω^θ for each type.	104
5.2	Illustration of OEATE's bag of successful paramters. For each ω agent and each possible type $\theta \in \Theta$, there is a bag of successful estimators. Successful estimator are copied to the bag of estimators of their respective type, in order to later generate new combinations of their elements. The check (green) indicates success in predicting the task and the cross (red) indicates failure.	105
5.3	Example of ϕ thinking about ω ' behaviour, when performing foraging.	107
5.4	New Choose Target state after ω_1 completing τ^1 . At this step, ω_1 will try to find a new task to pursue.	113
5.5	Estimation set modifications from the evaluation to the end of generation process. Figures (a) and (b) present the modifications after the evaluation and after the generation, respectively. Figure (c) presents the entire modification process.	116
5.6	Possible problem scenarios in the defined Level-based Foraging Domain.	129
5.7	Possible problem scenario in the defined Capture the Prey Domain. The agent with red details is the learning agent ϕ . The agents with blue details are the non-learning agents $\omega \in \Omega$. The grey agents are the prey.	133
5.8	Parameter and type estimation errors for $ \Omega = 7$, dimension 30×30 and $ \mathbf{T} = 30$	138
5.9	Results for a varying number of tasks with full observability.	139
5.10	Results for a varying number of agents with full observability.	140
5.11	Results for various environment sizes in full observability.	141

5.12	Results for a varying number of items in partially observable problems.	142
5.13	Results for a varying number of items, with randomly selected types among 4 types.	143
5.14	Results for a varying number of items, with randomly selected types among 6 types.	144
5.15	Performance of the ad-hoc team for a varying number of items without having information of correct potential teammates types.	145
5.16	Parameter errors, type estimation errors, and performance for a varying number of items in the Capture the Prey domain.	146
5.17	Ablation Study for parameter and type estimation errors considering $ \Omega = 7$, dimension 30×30 and $ \mathbb{T} = 30$ in the Level-based Foraging domain.	151
6.1	BAE's initialisation process in an environment with 3 other agents where one is an impostor and two are non-strategic teammates.	159
6.2	A high-level view of the tree search process. The arrows with dotted lines represent the existence of further paths in the tree not included in the illustration.	159
6.3	The selection of the best action and the extraction of the Q-table for the ad-hoc ϕ and the adversary ψ agents.	161
6.4	An illustration of BAE's update.	163
6.5	Small scenario spatial configuration (LBF.a).	167
6.6	Medium scenario spatial configuration (LBF.b).	168
6.7	Spatial configuration for the big scenario (LBF.c) and the cooperative scenario (LBF.d).	168
6.8	BAE's impostor probability across iteration per agent for each defined scenario in the LBF environment.	174
6.9	AGA's impostor probability across iteration per template type for each defined scenario in the LBF environment.	178

6.10	ABU’s impostor probability across iteration per template type for each defined scenario in the LBF environment.	179
6.11	AGA-BAE’s impostor probability across iteration per agent for each defined scenario in the LBF environment.	180
6.12	ABU-BAE’s impostor probability across iteration per agent for each defined scenario in the LBF environment.	181
A.1	User’s flow to build, modify and simulate environments in AdLeap-MAS.	205
A.2	The AdLeap-MAS high-level workflow, highlighting the information delivered at each step of the simulation. The red squares represent the workflow steps. The blue squares indicate the delivered information.	208
A.3	Class diagram of the AdLeap-MAS project presenting base classes for simulation. The highlighted texts indicate the framework’s main components.	209
A.4	Complete workflow of AdLeap-MAS framework, from the user definition to the internal simulation procedure.	211
A.5	CPU usage (%) scaling the components in the environment	213
A.6	Memory usage (Mb) scaling the components in the environment.	213
A.7	Time spent in one simulation step for different reasoning methods.	213

Chapter 1

Introduction

1.1 Autonomous Agents and the Advancement of Artificial Intelligence

Autonomous agents have become recurrent and prevalent in contemporary society, shouldering critical responsibilities to streamline and minimise our efforts required for routine daily activities and intricate procedures that would otherwise demand extensive time and dedication. For example, it is easy to find robots organising and managing packages in big companies' warehouses in order to facilitate the distribution and dispatch of products. Considering the context of the exponential growth of online sales in recent years, this application has become indispensable especially because maintaining quality and efficiency in the delivery service is intrinsically related to the quality and efficiency of the autonomous system in handling all orders made. Note that the significance of this process goes beyond attending just to the personal interests of people. Besides delivering products for our homes and private usage, it considers attending to other companies and supplying materials to other markets which are also essential to the community via the distribution of essential goods.

This example illustrates just one instance in which autonomous agents play a crucial role in our society. The application of autonomous agents extends far beyond,

encompassing more intricate challenges and systems, where they can be employed in large crop fields as precision pulverisation agents to enhance agricultural practices, in big companies as robust security surveillance mechanism to ensure the protection of valuable assets, and within our homes, where they serve as smart devices, brewing our coffee and illuminating the rooms to assist us in our morning wake-up routine.

The term “autonomous agent”, or simply “agent”, lacks a precise definition due to its diversity in forms and functions, which can range from simple programs composed of a small number of rules embedded in small chips to large and complex computing systems controlling multiple components to achieve a final objective [18]. However, it is important to note that, although this discussion considers the complexities related to agents’ hardware and software implementations, the main feature of these agents is their capability to make decisions and take actions that interact with the environment, modifying the current state of the world – that’s why we often refer to autonomous agents as “*decision-making*” agents, emphasising their acting capability.

Consequently, decision-making agents are increasingly used to solve complex problems under uncertainty in nontrivial systems [55], mainly through the development and application of Artificial Intelligence (AI). In these challenging scenarios, agents often play together as a team and follow specific coordination and communication protocols to enable the collection and exchange of valuable information to achieve a final objective. However, employing these methods is challenging due to environmental and technological constraints. For example, there are circumstances where communication channels are unreliable, and agents cannot fully trust them to send or receive information. Therefore, based on available information, such agents must be capable of evaluating potential decisions to build a robust plan that either accomplishes the tasks or leads it to states that minimise uncertainty, making the accomplishment of the objective easier.

As aforementioned, in order to enable agents’ “automaticity” and “decision-making” capabilities, most autonomous agent applications consider the development and implementation of *AI algorithms which perform planning and learning*.

Historically, the proposal of autonomous agents emerged as a result of research in the field of AI [18]. At that time, agents were modelled and implemented based on complex architectures with the aspiration to model human behaviour. The argument was that robots do not necessarily have to apply precise calculations or complex models to behave efficiently and to appear intelligent, they actually need to select the optimal features for sensors and mechanic components to control their response and achieve their objectives, mimicking a defined human behaviour [22].

In recent years, innovations in terms of software applications for autonomous agents, hence, the development of AI, intended to enable decision-making but it does not necessarily try to mimic human behaviour [18, 94]. Actually, the community has found that agents can perform better if we remove the constraints posted when trying to mimic humans. Although there are still existing limitations considering the available hardware capabilities, the development of novel AI techniques is now focusing on developing intelligent components that can be embedded into complex systems that not necessarily are cyber-physical systems (CPS). The *virtualisation of agents enabled the evolution of AI* as a mathematical and statistical tool that can be applied to any generic problem which fits the world model used by the method.

Under this perspective and for the purpose of this thesis, we consider that agents are primarily represented by their software. In other words, we assume that agents must still have the capability to make decisions and modify the environment based on the available information but it does not necessarily fit a specific CPS, instead, we are aiming to provide generic solutions in AI as a mathematical and statistical tool for real or virtual agents. This perspective deliberately highlights the software component of agents and advancements in AI – while we acknowledge the significance of hardware, we steer our study away from a detailed examination of this component. Consequently, our focus lies on the study of agents’ reasoning capabilities, i.e., their ability to retrieve information, process it and make decisions, instead of how to generate cyber-physical responses from our method. The central point in this discussion is to understand that our agent is a “decision-making agent”.

1.2 Making Decisions while Handling Uncertainty

Beyond understanding what agents are and how AI is connected to its development in society, it is important to understand the problems it often faces. As we mentioned, most applications require agents to be capable of evaluating potential decisions while handling uncertainty. A particular situation that can illustrate the above context and the existence of uncertainty in decision-making domains is: Consider a hazardous environment, such as an area affected by a natural disaster. In this scenario, addressing the problem may require the deployment of multiple agents, including robots or drones, from various parties with the shared goal of swiftly resolving the crisis. Consequently, each agent must be capable of performing a quick reasoning process to identify critical locations where they can provide support and save lives. If no clear target is found, they must rapidly formulate a plan based on their current knowledge and observable information to find an area needing assistance from their current position. However, it is important to note that constructing and testing communication, coordination, or policy protocols for all the diverse agents in the environment can become unfeasible within the time constraints of the problem, where delays can penalise lives and makes the other tasks accomplishment difficult.

The aforementioned scenario characterises a “*partially observable problem*”, a relevant and recurrent subject in the literature [5, 15, 43]. Such problems describe scenarios where agents must plan actions without access to a complete model of the world’s state. In essence, agents need to make decisions in the face of uncertainty in the environment. But what does “uncertainty” refer to? In the decision-making context, uncertainty encompasses any feature or information that might impact the action-planning procedure but is partially or entirely unavailable to the agent during its reasoning process. For example, in our hazardous environment, two types of uncertainty can be identified: the first related to the environment, where the agent can possess information about the region (e.g., a map of the affected area), and understand the procedure to be carried out when identifying a critical spot but it lacks information about the specific locations of these spots. The second related to

its teammates, where the agent is aware of the collaboration context, it assumes that every agent has a common objective but lacks comprehensive information about its teammates' parameters (e.g., vision capabilities and cooperation policies). Note that other levels of uncertainty can emerge in this scenario. For example, an agent can be uncertain about its own actions due to external factors, which may influence the execution of its action (e.g., intending to turn 90 degrees but, due to external conditions, managing only to perform a 60-degree turn). While we acknowledge the existence of these potential additional layers of uncertainty that may be associated with the problem, this thesis will focus specifically on the two aforementioned types of uncertainty – related to the environment and interactions with teammates. Nevertheless, we believe that our developed techniques are robust enough to address these various forms of uncertainty. This confidence stems from the fact that our approach is founded on generic MDP-based models, supported by both theoretical insights and empirical evidence gathered throughout our research.

In summary, making decisions while handling uncertainty is challenging because agents must find a way to gain knowledge indirectly from the information available in the environment and must be capable of applying it to their planning process. For this purpose, *online planning* solutions have become a powerful tool in the literature. The idea is the following: every time an agent makes an observation of the world, it gains information that can be used to improve its internal planning process.

A popular and relevant family of algorithms in the literature that enables online planning and leverages the gathered information to improve decision-making is the Monte-Carlo Tree Search (MCTS) family. The MCTS algorithms perform decision-making by conducting multiple sequential simulations within a “*tree*” structured framework, which enables the systematic exploration and evaluation of actions within the problem space with the objective to decide the best action to take in the real world — a process which is often referred to as the “*search*” process of a tree. The paradigm that guides the search is based on the exploration and exploitation trade-off idea, which defines that the better we explore the space of possibilities, the better we

can exploit the rewards in it — hence, the better we can accomplish our objective. At each step in the environment, these methods confront their simulated world against the real world and continuously refine their understanding of the problem, hence, improving their reasoning and planning capabilities on-the-fly.

Based on this, some works in the literature suggest improvements to traditional planning processes, handling the lack of prior information, by using the approximation of the world’s dynamic models [54, 80], embedding supportive planning methods within the tree [5, 15], extracting non-explicit information from the observations [10, 54], enhancing the method’s inner calculations [1, 32, 77] or employing neural networks-based techniques to improve the search quality [61, 108].

While these strategies, considering the environmental uncertainty, can often improve the search quality within a defined simulation horizon by using extensions of traditional partially observable models and enabling the integration of additional knowledge into planning, they demand the application of significantly more computational resources to perform the task (e.g., time and memory). Moreover, they may face challenges when essential knowledge, such as conditional observation probabilities or an explicit transition function, is unavailable to agents beforehand. Similarly, addressing uncertainty related to teammates poses its own set of challenges. Although some knowledge about the model of agents in the environment may be available, such as template types from reasoning or a range of possible parameters, optimising cooperation and task accomplishment in an online planning scenario lacks a straightforward solution, requiring agents to be capable of estimating some features to improve the search quality – note that estimating features also present some burdens in terms of spending resources.

In this thesis, we will provide novel solutions based primarily on the integration and modification of some of these methods from the literature within different online planning frameworks, hence, presenting a new perspective for incorporating information and knowledge into agents considering these partially observable scenarios

and concerning the existence of environmental and teammates' uncertainty in the problem. We will use a new statistical perspective on these frameworks in order to achieve our objective, approaching these problems with convergence guarantees in probability by exploiting the information gain inherited from online planning methods, mainly referring to Monte-Carlo-based algorithms, which will be introduced and explained in Chapter 2.2. Moreover, we will present these solutions always looking for improving performance without requiring significantly more resources to run our approach. Directly, this thesis presents lightweight methods capable of handling partially observable problems in an online manner.

1.3 Reinforcement Learning: Developing Reward-guided Planning Methods

Having grasped the significance of decision-making and how this process is affected by the uncertainty of the environment, let us delve into the investigation of some strategies within the literature that are designed to develop and enhance decision-making capabilities under partial observability into autonomous agents.

One particularly important and recurrent strategy in the state of the art is categorised by the application of Reinforcement Learning (RL). RL models not only can handle the intricacies posed by uncertain environments but also stand out as a pivotal approach that empowers agents to learn, adapt, and optimise their decisions through ongoing interactions with the dynamics in their surroundings. Moreover, most of the current RL solutions are built over a versatile framework that aligns seamlessly with the autonomous agent's quest to navigate complex, unpredictable scenarios, stating a paradigm that considers agents making sequential decisions over time, with the goal of maximising a cumulative reward signal – which we denominate here as the *reward-guided planning paradigm* of RL algorithms.

Under this paradigm, the agent, as a conscious entity, learns from its experiences

by receiving feedback in the form of rewards or penalties and adjusting its strategies accordingly. This learning mechanism, inspired by behavioural psychology, enables agents to discern optimal courses of action within the context of uncertain and dynamic environments. However, the question that might arise is: what happens when there is no reward signal available to perform learning?

This question raises a fundamental academic challenge which is one of the principal topics developed in this thesis, where we try to address this significant limitation of traditional RL algorithms in handling sparse reward scenarios. As mentioned earlier, we will be studying and analysing the planning efficiency considering mainly Monte-Carlo-based algorithms. Considering the partial observability feature of our problem, a commonly suggested algorithm to address these problems is the Partially Observable Monte-Carlo Planning (POMCP) [93] which is a RL algorithm capable of performing planning in an online manner while handling uncertainty [10, 99, 109]. However, several state-of-the-art solutions that rely on POMCP often struggle when rewards are delivered sparsely in time or outside their reasoning horizon. For example, in the given example of hazardous environments, there may be no clear target to spot from the current observation and/or estimated map so far, which culminates in the absence of reward in planning, hence, penalising planning's quality.

Straightforwardly, we introduce a novel perspective over this context that transforms the reward-guided planning paradigm into what we denominate as *information-guided planning paradigm*. Our proposal is to use the information found during the collection of observations to guide the agents in planning when no rewards are available to differentiate potential strategies. IB-POCMP [27] is our main contribution related to this topic, which performs information-guided planning by valuing the observations set entropy and including it in the reasoning process. It will be properly introduced and presented in detail in Chapter 4.

1.4 Teamwork and Decentralised Execution

In light of the established understanding of autonomous agents as problem solvers in society, another crucial aspect to explore is the existent potential of applying multiple agents to solve problems in our context as a team. This approach of organising agents to act together to achieve a common final objective defines what is called in the literature a cooperative Multi-Agent System (MAS). Although it introduces a new layer of complexity to the problem, mainly related to the challenges of modelling each agent to act in the world, the collaborative synergy derived from autonomous agents working together allows the amalgamation of diverse strengths that often surpass the individual capabilities of a single agent, even if it is well-trained and well-designed for the problem.

These cooperative efficacy gains are even clearer when we confront our single-agent solutions against a MAS in problems that demand a spectrum of expertise or involve large-scale operational endeavours. Just as an example, let's consider our disaster scenario. In a single-agent approach, a lone robot handles tasks like mapping, locating survivors, and providing medical aid to all possible spots in the environment. On the other hand, building a MAS can ensure the efficient distribution of responsibilities, optimising task completion.

Considering the design of MAS solutions, *teamwork models* have shown great potential to navigate complex applications and facilitate the construction of knowledge among agents who share both the environment and a common objective. However, the effective application of these models often hinges on the formulation of a well-crafted *decentralisation strategy* for coordination and cooperation.

In essence, the development of decentralised systems involves the distribution of the decision-making procedure across multiple agents, each equipped with distinct capabilities and information sources. On the other hand, this feature fosters the independence between agents' decision-making processes, affecting the coordination

and collaboration quality. In scenarios such as our hazardous environment, the deployment of multiple agents, like robots or drones, becomes more effective when decision-making is decentralised.

This approach enables agents to autonomously assess critical situations and formulate immediate and precise responses based on their individual knowledge. On the other hand, considering a centralised mechanism to allocate tasks efficiently to each agent may prove challenging to implement, or it might even be proved infeasible due to technological or environmental constraints. In large-scale problems, these constraints could easily derail a centralised solution, ranging from the absence of reliable communication channels that covers the action area to the condition of massive computational power to manage and direct each possible procedure and task for numerous agents in the environment.

Decentralisation, while recognising the challenges posed by uncertain environments, allows agents to adapt dynamically to changing conditions without solely relying on a central authority. It enhances the flexibility and responsiveness of autonomous agents, yet it introduces complexities in coordination and cooperation among diverse agents with potentially differing objectives, as discussed earlier. Besides that, decentralisation is still a powerful strategy in developing robust solutions for decision-making under uncertainty, fostering adaptability and resilience in the face of dynamic and unpredictable environments.

As a result, this thesis contributes to enhancing the decentralised execution of agents facing uncertainty about their teammates' capabilities and roles. We introduce two distinct solutions for addressing each aspect: The first solution is OEATE [28], an online estimation algorithm capable of approximating the type and parameters of unknown teammates following a task-based strategy and without relying on prior training. The second solution is BAE [29], which is also an online estimation algorithm, but its focus lies on the identification of adversarial agents among a team of unknown agents. We will properly introduce and present more detail about both methods in Chapter 5 and Chapter 6, respectively.

1.5 Our Research Questions and Contributions

In this thesis, we aim to provide fresh solutions and a comprehensive study that surrounds the following research questions posted for the big field of AI:

- **Q1) Considering the aspects of planning under uncertainty, how can we handle the lack of information without spending significantly more resources, such as time and memory?**

For the first question, this PhD thesis aims to present new solutions for planning under uncertainty that are capable of aggregating knowledge in an online manner, without requiring prior training or the application of exhaustive strategies to find a solution. Directly, we propose smart applications of computing together with statistics which are well-designed to perform in a lightweight fashion, i.e., extracting information quickly and without using large amounts of data. Our solutions are close to the emerging big field of Data Science and follow what we denominate as an information-guided paradigm of planning (introduced in Chapter 4).

- **Q2) Considering the current reward-guided paradigm in RL solutions, how can we solve problems with sparse rewards?**

Our method that directly answers this is *Information-based Partially Observable Monte-Carlo Planning* (IB-POMCP), which is introduced in detail in Chapter 4 and represents a published Full Paper in NeurIPS 2023 [27]. IB-POMCP is an online planning algorithm that uses a novel *information framework* to boost the decision-making process under partial observability even when no rewards are available in the reasoning horizon. Our framework refines the traditional UCB1 action selection strategy by implementing our proposed **Information-guided UCB** (I-UCB) function, which is capable of leveraging entropy and estimating information gain, using both real-world and internally generated observations to identify promising states in the search process that leads the agent to quickly accomplish its objective. Its application together with our new particle filter reinvigoration strategy, which considers the

current system’s entropy to calibrate the reinvigoration, indicates that this approach may lead the agent to act hybridly when solving a partially observable problem by better weighing the exploration, exploitation, and information levels within the tree.

- **Q3) Considering multi-agent systems working under uncertainty, how can we surpass the limitations imposed by performing teamwork with unknown teammates?**

In order to tackle this question, we present *Online Estimators for Ad-hoc Task Execution* (OEATE), a work which is published as a Journal Paper in JAAMAS’22 [26], a Full Paper in AAMAS’23 [28] and we present its details in Chapter 5. OEATE is a *novel algorithm* for estimating teammates’ types and parameters in decentralised task execution. Our algorithm is light-weighted, running estimations *from scratch* at every single run, instead of employing pre-trained models, or carrying knowledge between executions. OEATE uses a genetic algorithm-inspired approach to improve the estimation quality and the performance of the whole team due to better coordination and cooperation between all agents. Under some assumptions, we show theoretically that our algorithm converges to a perfect estimation when the number of tasks to be performed gets larger. OEATE can obtain a lower error in parameter and type estimations in comparison with the state-of-the-art, leading to significantly better performance under different settings and while facing distinct challenges, such as planning in large scenarios, collaborating with a large number of unknown teammates and estimating types and parameters with wrong templates.

- **Q4) Considering the emerging discussion about adversarial agents in decision-making systems, how can we handle it without harming the algorithm planning capabilities?**

Finally, to address this specific and relevant situation in the literature, we propose *Bayesian Adversary Estimation* (BAE), which represents a Full Paper at AAMAS’24 [29] and is properly introduced in Chapter 6. BAE is a novel algorithm capable of

identifying an impostor agent among the team. In our context, an impostor agent is a smart agent acting as an “adversary”, i.e., while teammates try to enhance task completion, the impostor endeavours to hinder performance, disrupt objective achievement by blocking paths or fake its real intention. BAE performs an online impostor deduction by performing estimations solely using the observations collected while acting in the environment and simulating potential outcomes in its reasoning process. We propose the application of what we denominate as the *Q-valued Bayesian Estimation* (QvBE) approach, which considers approximate latent information about the environment (e.g., agents’ transition function) using Q-value estimations. Our approach focus on using a Monte-Carlo Tree Search (MCTS) based method; however, we emphasise that it can be extended to any online planning algorithm that estimates Q-values. We propose the embedding of our the QvBE approach inside an Adversarial-MCTS (A-MCTS), which performs planning from scratch and estimates its own actions considering the existence of an adversary. The Q-table found at the end of the search process is used to estimate the impostor among the team.

1.6 Additional Contribution: AdLeap-MAS

Having in mind all the discussion about developing autonomous agent solutions, their virtualisation and their test in synthetic scenarios, this PhD thesis lists as a side but relevant contribution to the community the development of an open-source MAS simulator for the development and evaluation of solutions for planning.

We propose *Adaptive Learning and Planning Multi-Agent Simulator* (AdLeap-MAS) [25], a novel framework focused on simulating ad-hoc reasoning problems, where potential types/policies for other agents are estimated, and sampled during an online decision-making process. We offer base classes for implementing new problems and algorithms, besides ready-to-use common benchmarks found in the literature. This proposal supports the execution of reactive algorithms, neural networks, estimation methods, reinforcement learning and online planning applications over full and partial

observability, only requiring the connection of algorithms to the ad-hoc reasoning model. In this way, our contributions with AdLeap-MAS can be summarised as: **(i)** First simulator that allows a quick switch of learning and planning algorithms across different ad-hoc reasoning scenarios; **(ii)** AdLeap-MAS enables the execution of multiple reasoning agents that run independently inside the framework and according to the domain settings; **(iii)** Our architecture guarantees information security while running scenarios under partial observability, i.e., agents do not have access to any forbidden information; **(iv)** AdLeap-MAS offer a standard set of baselines algorithms and benchmark problems to allow fair and quick experiments.

We refer the reader to our GitHub¹ or Appendix A for more information.

1.7 Thesis Content

This thesis is organised into six (6) additional chapters following the introduction. In Chapter 2, we provide essential background material necessary for understanding the rationale behind each newly proposed method designed to address our defined research questions. Chapter 3 examines pertinent literature, both contributing to and challenging the discussions developed in this thesis. From Chapters 4 to 6, we delve into a detailed exploration of the proposed methods developed during this PhD journey. Each chapter offers an in-depth examination of the methodology, theoretical and empirical evaluations, discussions on limitations, and conclusive remarks for the respective method. Upon navigating through all method-focused chapters, Chapter 7 serves as the culmination of this thesis, presenting the final considerations over our results, a comprehensive discussion about our limitations and the outline of potential future works. Additionally, supplementary material in the Appendix is referenced throughout the text to ensure a good reading experience. At the end of this document, we list all references cited in this thesis.

¹AdLeap-MAS's GitHub page: <https://github.com/lsmcolab/adleap-mas/>

Chapter 2

Background

In this chapter, we provide the essential background information necessary for understanding the contributions and discussions presented in this thesis.

We start introducing all Markovian models in Section 2.1 employed to represent environmental features tailored to the requirements of each contribution made in this thesis. These models include the full observable Markovian model, discussed in Section 2.1.1, the partially observable model, presented in Section 2.1.2, and the multi-agent representation in a Markovian context in Sections 2.1.3 and 2.1.4.

Following this, we introduce the tree-search planning methods used in our research in Section 2.2. We begin by presenting and discussing the traditional MCTS algorithm in Section 2.2.1. Subsequently, we introduce the partially observable version of MCTS, the POMCP, in Section 2.2.2. Finally, we present the single-agent and adversarial versions of these approaches tailored for an ad-hoc teamwork context in Sections 2.2.3 and 2.2.4, respectively.

In Section 2.3, we provide the necessary material to comprehend the information analysis employed in this work and how it can enhance an agent's planning capabilities. Consequently, in Section 2.3.1, we discuss the application and representation of information value through Shannon's entropy perspective and, in Section 2.3.2, we delve into the details and application of Bayesian Inference approaches to enhance an agent's decision-making process.

Finally, Section 2.4 concludes the Background section by introducing two essential estimation methods from the state-of-the-art. These methods are utilized in the experiments presented in Chapters 5 and 6. Specifically, we present the AGA and ABU algorithms, proposed by Albrecht and Stone (2017) [5], and discuss their major features, enabling readers to differentiate our approaches from the relevant literature.

2.1 Markovian World Representation

Markovian models have been recurrently used as a powerful tool to tackle and design different problems for planning and estimation problems [5, 94]. The Markovian framework models the world as a stochastic process in a discrete-time flow while maintaining the Markov properties [87], i.e., at any given time, the next state is only dependent on the current state and is independent of the past. This feature makes the model efficient and lightweight since we can work with less data in order to find solutions to complex problems. Moreover, Markovian models are generic and can be extendable to different scenarios, considering full or partial observability, single or multiple agents and even competitive or cooperative scenarios.

In this thesis, we employ different Markovian models to address each aforementioned problem adequately. Consequently, in this section, we will provide the necessary background to facilitate the understanding of our methodologies. In summary, our solutions implement the following models:

- In Chapter 4, where we delve into IB-POMCP, we will be using an extended version of the Markov Decision Process (MDP) model tailored for partially observable problems, known as *Partially Observable MDP (POMDP)*. We opted to use the POMDP model because, besides enabling the modelling of observations and uncertainty in the environment, the POMCP framework is traditionally designed over this model [93].
- In Chapter 5, for the implementation of OEATE, we propose the usage of both the MDP and POMDP models depending on the target problem. However,

although our method is applied in a multi-agent context, we will adapt both models to represent the other agents (teammates) as part of the environment. This design is inspired by prior works [2, 91] and fits OEATE’s unique teamwork context. This adaptation eliminates the need to model each team member individually, streamlining the planning and estimation process.

- Finally, in Chapter 6, which focuses on BAE’s presentation, we will employ an extended version of MDP designed for multiple agents, known as Multi-agent MDP (MMDP). In contrast to OEATE, BAE treats the impostor agent as a strategic entity while categorising other teammates as non-strategic agents, hence, modelling them as part of the environment. This adaptation enhances the track of the potential adversary during reasoning by directly modelling its action in the planning process (i.e., BAE’s tree search process).

We start this section by introducing the MDP model (Section 2.1.1) since it represents the central model from which we extend and present the others. In Section 2.1.2, we present the POMDP model as the extension of the introduced MDP model for partially observable contexts. In Section 2.1.3, we present the MDP and the POMDP model for multi-agent environments but under a single-agent perspective. Finally, in Section 2.1.4, we introduce the Multi-agent MDP (MMDP) model, which is capable of accommodating the multi-agent context while modelling more than one perspective in the Markovian model’s target environment.

2.1.1 Markov Decision Problems

The Markov Decision Process (*MDP*) is a mathematical framework to model stochastic processes in a discrete-time flow. A MDP consists of a set of states $s \in \mathbf{S}$, a set of actions $a \in \mathbf{A}$, a reward function $\mathcal{R}(s, a, s')$, and a transition function \mathcal{T} . Consequently, it can be represented by the 4-tuple:

$$MDP = (\mathbf{S}, \mathbf{A}, \mathcal{R}, \mathcal{T}) \tag{2.1}$$

A state s in the set of states \mathbf{S} is a representation of the world from the intelligent agent perspective. On the other hand, an action a in the set of actions \mathbf{A} is responsible for linking one state to another, defining what is called a “state transition”. The states transitions are modelled by the transition function \mathcal{T} , which defines the probabilities of reaching another state s' , given a state s and an action a . Mathematically, it can be described as $\mathcal{T}(s, a, s') = P(s'|s, a)$ or $\mathcal{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$. Moreover, actions are responsible for generating rewards for the agent considering its advancement towards the final objective of the problem. The MDP model enables this evaluation through its reward function \mathcal{R} , which is responsible for delivering rewards to the agent. Mathematically, it is defined as $\mathcal{R}(s, a, s') = \mathbb{R}$ or $\mathcal{R} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow \mathbb{R}$. The reward function represents the main component of MDPs used to train RL models since, through this component, we can approximate the value of taking an action from our current state intending to advance in the problem – and that is why handling the absence of rewards is a relevant topic in the community and it is one of the object of study in this thesis.

2.1.2 Partially Observable Markov Decision Problems

The Partially Observable MDP (*POMDP*) represents the extended version of the MDP model for problems in which the agent cannot directly access the current state – defining what is denominated as a “partially observable scenario” or “partial observability”. In this context, instead of receiving complete information about the world while reasoning, agents receive an observation $z \in \mathbf{Z}$ which represents the visible information from its current state and single perspective.

Similar to the MDP, the POMDP model consists of a set of states $s \in \mathbf{S}$, a set of actions $a \in \mathbf{A}$, a reward function \mathcal{R} and a transition function \mathcal{T} , however, it also includes an observation set \mathbf{Z} , an observation function \mathcal{Z} and a set of information or history $h \in \mathbf{H}$. Consequently, it can be represented by the 7-tuple:

$$POMDP = (\mathbf{S}, \mathbf{A}, \mathcal{R}, \mathcal{T}, \mathbf{Z}, \mathcal{Z}, \mathbf{H}) \tag{2.2}$$

The observations are delivered to agents following an observation function \mathcal{Z} and its probabilities $\mathcal{Z}_{s,z}^a = P(z_{t+1} = z \mid s_{t+1} = s; a_t = a)$ or $\mathcal{Z} : \mathbf{S} \times \mathbf{A} \times \mathbf{Z} \rightarrow [0, 1]$. The belief function $\mathcal{B}(h)$ is a distribution over the state space \mathbf{S} , which describes the conditional probability of being in a particular state $b \in \mathbf{S}$ given the current history h , $\mathcal{B}(h, b) = P(s_t = b \mid h_t)$, where the history $h \in \mathbf{H}$ is a sequence of action-observation pairs $h_t = \{a_0, z_0, \dots, a_t, z_t\}$ from the time 0 (start of the process) to the current timestamp t . The set of all possible information is denoted as \mathbf{H} . The decision process follows a policy $\pi(h, a) = P(a_{t+1} = a \mid h_t)$, while choosing actions. The optimal policy for a problem to decide the best actions is represented by π^* .

2.1.3 Multi-agents as part of the environment

From the provided definitions for both the MDP and POMDP models, it is possible to note that these Markovian models are conventionally utilised to design single-agent problems with a specific objective. However, the application of these models can be extended to encompass Multi-Agent Systems (MAS). As an efficient and lightweight approach to model MAS problems within an MDP or POMDP framework, we consider that there is a central or main agent, denoted as ϕ , which treats the existence of all other agents, denoted as $\omega \in \mathbf{\Omega}$ (teammates), as components of the environment. That is, all ω in $\mathbf{\Omega}$ are modelled as part of the *environment* with their actions indirectly affecting the next state and the obtained reward. Therefore, they are abstracted in the transition function.

In this context, the ϕ actively makes decisions and takes actions in a deterministic manner, while the remaining agents $\omega \in \mathbf{\Omega}$ contribute to the environment in a stochastic manner. In other words, ϕ can only decide its own actions and has no control over other environment components (e.g., actions of agents in the set $\mathbf{\Omega}$) – the next state depends on the actions of all agents, however, ϕ is unsure about the non-strategic agents' next action. This approach avoids the need to model every agent's actions as parameters for the transition, reward function, and observation, yet it still accommodates their impact on the environment as a probabilistic event.

We consider that given a state s , an agent $\omega \in \Omega$ has a (unknown) probability distribution (pdf) across a set of actions \mathbf{A} , which is given by ω 's internal algorithm. This pdf is going to affect the probability of the next state. Therefore, we can say that the uncertainty in the Markovian model comes from the randomness of the actions of the ω agents, besides any additional stochasticity of the environment. Moreover, this model allows us to employ single-agent online planning techniques like UCT Monte Carlo Tree Search [58] while performing a tree search process. We will present the details of its application in Section 2.2.3.

2.1.4 Multi-Agent Markov Decision Problem

While addressing the MAS scenario by modelling it as a single-agent Markovian problem proves to be a viable approach, this model might not be universally suitable for all scenarios. As previously mentioned, employing single-agent MDP or POMDP modelling serves as an effective and lightweight solution for our intended investigation. However, in certain instances, avoiding the design of other agents' actions can have a significant impact on algorithm performance, consequently affecting its ability to achieve objectives. For instance, in an adversarial deduction context, modelling of potential adversaries' actions impacts the algorithm's capacity to accurately estimate the optimal action in response to adversarial strategies.

Under this context, we focus on the application and implementation of a more complex Markovian-based model together with RL algorithms to handle the proposed problem. Hence, we propose and describe the application of a Multi-agent Markov Decision Process (MMDP) [19]. This model considers that $M = |\phi \cup \Omega|$ agents are sharing the same environment and compounding the team Λ . The MMDP model contains a finite set of states $s \in \mathbf{S}$ with transition probability \mathcal{T} and expected reward equals to $\mathcal{R}(s, \mathbf{J})$ depending on the joint-actions of all agents $\mathbf{J} = \{a_1, \dots, a_M\}$, where each action is defined in the action space $a \in \mathbf{A}$. Therefore, the MMDP model is represented by the 6-tuple:

$$MMDP = (\mathbf{S}, \mathbf{A}, \mathbf{J}, \mathcal{T}, \mathcal{R}, \Lambda) \quad (2.3)$$

We use the MMDP to represent the problems studied and proposed for BAE (Chapter 6). Since it describes an adversarial context, we assume there is an impostor agent ψ which is disguised as a non-strategic agent among the team and tries to minimise the team’s performance by acting in the environment. Therefore, in order to enable the traditional MMDP to consider a potential adversarial agent in the environment, we assume that $\psi \in \Omega$ and propose its representation by a 7-tuple:

$$MMDP_\psi = (\mathbf{S}, \mathbf{A}, \mathbf{J}, \mathcal{T}, \mathcal{R}, \Lambda, \psi) \quad (2.4)$$

We highlight that, in our adversarial deduction context, ϕ and ψ are strategic agents, while all other agents ω are non-strategic [107]. That is, both ϕ and ψ are capable of modelling other agents in their reasoning process, while the other ω agents do not model teammates and typically follow fixed rules for planning. The details about the implementation of all agents and their decision-making process will be presented in Chapter 6, where we formalise the problem and evaluate our results.

2.2 Tree-Search Methods

The application of *tree-search-based algorithms* to tackle complex problems continues to offer relevant solutions in the literature by incorporating heuristics, leveraging complex spaces, and building hybrid models that combine their systematic exploration and exploration of the environment with the strengths of other approaches [7, 71, 73, 113]. Widely applied in AI contexts, especially when considering the proposal of RL solutions, their popularity stems from their versatility in modelling the world and their ability to perform planning and estimation in an online manner.

From the current state-of-the-art, MCTS-based methods stand out as relevant and effective algorithms for decision-making in complex and uncertain environments. Renowned for their simulation-based approach, these algorithms excel in games with

large state spaces, exemplified by their success in applications like the proposal of POMCP [93] and AlphaGo [94]. These algorithms adeptly balance exploration and exploitation, making them adaptable to unknown environments and suitable for real-time decision-making. These features demonstrate that these methods are scalable and effective enough to tackle this thesis’s context.

Aiming to guarantee the optimisation of our solutions, we employ different tree-search methods tailored to each of our proposals, as demonstrated in our approach to Markovian models. In this section, we will introduce and provide the necessary background to facilitate the understanding and implementation of our methodologies. In summary, our solutions implement the following MCTS-based methods:

- For IB-POMCP (Chapter 4), we will use the Partially Observable Monte-Carlo Planning (POMCP) algorithm, proposed by Silver and Veness (2010) [93]. We opted to use this method due to the aforementioned benefits and because it can easily model observations of the scenario in its planning procedure.
- For OEATE (Chapter 5), we propose the usage of both, the full observable and the partially observable Monte-Carlo tree search algorithms, MCTS and POMCP respectively. Each algorithm will be applied according to the target problem and intends to enhance our estimation method capability and the team’s performance. However, we highlight here that we will be using the *single-agent model* to tackle this problem.
- Finally, for BAE (Chapter 6), we employ a different MCTS-solution which is focused on *multiple agents* under an *adversarial* context: the Adversarial MCTS (A-MCTS) algorithm, which is an adapted version of traditional the 2-player MCTS’s structure [23] for our adversarial context. This algorithm directly models the adversary in the nodes of the tree in order to improve the reasoning capabilities of MCTS against adversaries in the environment.

This section commences by introducing the traditional MCTS algorithm (Section 2.2.1), which can be easily comprehended with the fundamental knowledge about tree

structures and their applications in decision-making [87]. In Section 2.2.2, we present the Partially Observable Monte-Carlo Planning (POMCP) algorithm, proposed by Silver and Veness (2010) [93], showcasing its role as an extension of the generic MCTS tailored for scenarios involving partial observability. In Section 2.2.3, we present the construction of our MCTS solution for MAS from a single-agent perspective, which models non-strategic agents as part of the environment. Finally, in Section 2.2.4, we introduce the A-MCTS algorithm, adept at accommodating multi-agent contexts and incorporating multiple perspectives within a unified tree search process.

2.2.1 Monte Carlo Tree-Search

MCTS algorithms aim to determine the optimal action a^* for a given state s by simulating the world steps within a tree structure. The MCTS's tree structure \mathbf{T} considers the existence of state nodes connected through actions, as illustrated in Figure 2.1. Each node in the tree is represented by $(s, \mathcal{V}, \mathcal{N})$, i.e., a 3-tuple with a state s , a value $\mathcal{V}(s, a)$, and a visitation count $\mathcal{N}(s, a)$ for each action $a \in \mathbf{A}$. The value of the node represents the expected cumulative reward for the simulated states. The number of visits to the state s is represented by $\mathcal{N}(s) = \sum_{a \in \mathbf{A}} \mathcal{N}(s, a)$.

Each state in the search tree is viewed as a multi-armed bandit taking actions usually chosen by Upper Confidence Bound (UCB1). *UCB1 is a well-known* algorithm that tries to increase the value of less-explored actions by attaching a bonus inversely proportional to the number of times each action is tried, following:

$$UCB1(s, a) := \mathcal{V}(s, a) + c \sqrt{\frac{\log(\mathcal{N}(s))}{\mathcal{N}(s, a)}} \quad (2.5)$$

The scalar c is the exploration constant responsible for weighting the exploration value, given by $\sqrt{\frac{\log(\mathcal{N}(s))}{\mathcal{N}(s, a)}}$, within the UCB1 function. If it is close to 0, the algorithm will act greedily, following only the exploitation value calculated in $\mathcal{V}(s, a)$. We can fit this constant to the target problem by considering the desired balance between exploiting close and future rewards.

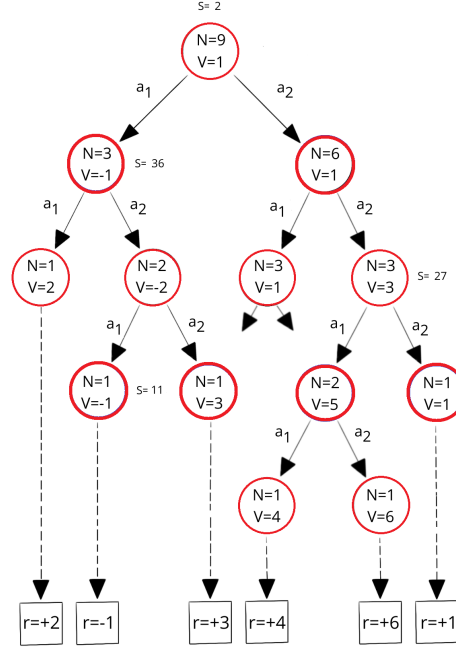


Figure 2.1: Illustration of MCTS’s tree structure. Each node represents a state s . Some states are specified in the figure to highlight this difference. Each node is connected to another via an action $A = \{a_1, a_2\}$. N represents the number of visits and V the current value estimated to each specific node. Each box at the bottom of the figure presents the reward r found in the last node.

Note that the $\mathcal{V}(s, a)$, presented in equation 2.5, is updated at each visit to the node, following the update function $\mathcal{V}(s, a) := \mathcal{V}(s, a) + \frac{(\mathcal{R}(s, a) - \mathcal{V}(s, a))}{\mathcal{N}(s, a)}$. The visit count is also updated at each iteration to the node, following $\mathcal{N}(s, a) := \mathcal{N}(s, a) + 1$. With these updates in hand, we found the following result: with a correct definition of c , the $\mathcal{V}(s, a)$ function converges in probability to its optimal value $\mathcal{V}^*(s, a)$, which can be mathematically represented by $\mathcal{V}(s, a) \xrightarrow{p} \mathcal{V}^*(s, a)$ [93]. In other words, if the standard updates, as defined above, are followed and a suitable value for c is found, then the $\mathcal{V}(s, a)$ values converge to the true (optimal) values $\mathcal{V}^*(s, a)$ for the given state s and action a . The idea behind this result is that, with an infinite number of iterations over a node, the variations in the value of $\mathcal{V}(s, a)$ will approximate 0 and, hence, the UCB1 can choose the “best” action according to the “estimated optimal

policy”. This is an important result, which motivates our work and supports further discussions of our contribution. Besides that, in this thesis, we focus on *UCB1* but acknowledge the existence of different options in the literature as action selection methods, such as the Thompson-Sampling and PUCT [9, 94].

Now, let us outline the traditional MCTS algorithm. The MCTS algorithm starts with the *initialisation of the tree* \mathbf{T} , where we create the root as a simple node $(s, \mathcal{V}, \mathcal{N})$. Initially, s is the current state of the world, $\mathcal{V}(s) = 0$ and $\mathcal{N}(s) = 0$, since we did not visit the node yet. Subsequently, our *search process starts*. It is an iterative process where the agent performs multiple rounds of reasoning, *simulating* steps in the world to evaluate which is the best sequence of actions that solves the problem. This strategy requires a *Monte-Carlo simulator* \mathcal{G} , which can sample the next state s' and its reward r given a state-action pair. Every time the agent chooses an action and simulates it within the tree, a new node is created and added to the tree – a process often referred to as the “tree expansion process”. In general lines, the search is categorised by two procedures: the *simulation* and the *rollout*.

The simulation process is responsible for expanding the tree from an already existing node. During the process, the UCB1 is responsible for selecting the actions that will expand the tree, which will be the action with the highest value according to Equation 2.5. In other words, the action a which is used to expand a node s is obtained from $a = \arg \max_{a' \in \mathbf{A}} \text{UCB1}(s, a')$. However, If some action a' was never chosen, then it will be preferably chosen instead of the action a with the highest UCB1 value. This approach guarantees that each action is tried at least once before expanding the tree further into promising branches. At the end of each simulation, all rewards are back-propagated to the root and the nodes’ values are updated. This process is known as *reward back-propagation*.

The rollout process, on the other hand, is responsible for expanding the tree from a non-existing node, hence, creating it. Overall, it works similarly to the simulation process but it does not create new nodes. The idea of the rollout process is to estimate an initial value (better than 0) for the new nodes created in the tree,

increasing the possibility of differentiating actions and, consequently, improving the planning process. Besides that, the rollout process does not rely on the calculations of UCB1, it applies a pre-defined rollout policy $\pi_{rollout}$, which will guide the rollout simulations. At the end of each rollout, all rewards are back-propagated from the *created node* to the root, updating the values in the process.

The usual stop conditions for the simulation and the rollout processes are the *maximum depth* and the *end of the problem* (when, while performing the search, the agent reaches a terminal state). If any of these conditions are reached, the *reward back-propagation* starts, even if no reward is found, following the equation:

$$\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \frac{R - \mathcal{V}(s)}{\mathcal{N}(s)} \quad (2.6)$$

The R in the right-hand of the equation is the back-propagated reward, which aggregates the instant reward r and future rewards R' in a single value through $R \leftarrow r + \gamma R'$. The instant reward r is the reward collected by the agent when acting at its current time and state. On the other hand, the historical rewards R' are the rewards collected in the future. In order to differentiate the rewards collected close to the present and far in the future, it is proposed the application of a historical discount factor γ to balance the values during the back-propagation.

After finishing the search process, i.e., finishing the multiple rounds of reasoning within the tree about possible paths to take, the *best action* is selected to be performed in the real world based on the estimated rewards, following:

$$a_{best} = \arg \max_{a \in \mathbf{A}} \mathcal{V}(s, a) \quad (2.7)$$

After taking the best action, the current state and the root of the MCTS are updated to the current step in the execution. Then, the MCTS search process restarts. Algorithm 1 presents the pseudo-code for the MCTS implementation.

Finally, we highlight here that the above explanation and introduction of our MCTS planning process considers the suggestion from the literature about the application of the the UCT-H [111], an enhanced version of UCT [58], to augment the

Algorithm 1 MCTS's Planning

<pre> 1: procedure SEARCH(s) 2: while Timeout() is False do 3: Simulate($s, 0$) 4: return $\arg \max_{a \in \mathbf{A}} \mathcal{V}(s, a)$ </pre>	<pre> 1: procedure SIMULATE(s, d) 2: if $d < depth_{max}$ then 3: return 0 4: else if is_terminal(s) then 5: return 0 6: if $s \notin \mathbf{T}$ then 7: Expand_Node(s) 8: return Rollout(s, d) 9: $a \leftarrow \arg \max_{a \in \mathbf{A}} \text{UCB1}(s, a)$ 10: $(s', r) \sim \mathcal{G}(s, a)$ 11: $R \leftarrow r + \gamma \text{Simulate}(s', d + 1)$ 12: $\mathcal{N}(s) \leftarrow \mathcal{N}(s) + 1$ 13: $\mathcal{N}(s') \leftarrow \mathcal{N}(s') + 1$ 14: $\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \frac{R - \mathcal{V}(s)}{\mathcal{N}(s)}$ 15: return R </pre>
<pre> 1: procedure EXPAND_NODE(s) 2: for $a \in \mathbf{A}$ do 3: $\mathbf{T}(sa) \leftarrow (\mathcal{T}(s, a), 0, 0)$ </pre>	
<pre> 1: procedure ROLLOUT(s, d) 2: if $d < depth_{max}$ then 3: return 0 4: $a \sim \pi_{rollout}(s, \cdot)$ 5: $(s', r) \sim \mathcal{G}(s, a)$ 6: return $r + \gamma \text{Rollout}(s', d + 1, \gamma)$ </pre>	

MCTS capabilities in exploring and exploiting the search space of an ad-hoc teamwork problem. Briefly, both the UCT and UCT-H algorithms are tree search methods that follow a similar search procedure to the MCTS explained above. Therefore, both methods are planning algorithms that employ the UCB1 equation while expanding a tree structure in search of the best action to take to solve the problem. The UCT considers the creation of a new node for every possible next state s' from s by taking action a and evaluates each one separately. On the other hand, UCT-H extends the UCT proposal by considering the creation of a single node from s, a , which represents and evaluates all possible next states s' together in this single node. This approach can summarize all potential outcomes from an action into a single node, saving memory and boosting the value estimation for actions while sacrificing the exact representation of a state for a node. For the ad-hoc teamwork context, this abstract representation of a state is beneficial because representing all possible states considering the actions of all agents in a team inflates the size of the state space

significantly, making it challenging to take an action even when considering one step in the world. Therefore, considering these characteristics, we employed the UCT-H in the MCTS implementation for this thesis.

2.2.2 Partially Observable Monte Carlo Tree-Search

POMCP is an extension of MCTS for partially observable problems that commonly apply a POMDP model together with the Partially Observable UCT (PO-UCT) to evaluate the world and build the tree [93]. The structure of this tree (illustrated in Figure 2.2) considers the existence of action and observation nodes, which make the agent aware of the uncertainty in the scenario. Each node is represented by a 4-tuple $(h, \mathcal{V}(h), \mathcal{N}(h), \mathcal{F}_h)$, where h is the history of the node, $\mathcal{V}(h)$ is the value of the node h , $\mathcal{N}(h)$ is the visit count of the node h and \mathcal{F}_h is the unweighted particle filter of the node h used to approximate the belief state at each node h .

Analogously to the *MCTS* procedure, the algorithm commonly uses the UCB1 function to select actions during the search process and is divided into the *simulation* and *rollout* stages, requiring then a *Monte-Carlo simulator* \mathcal{G} , which can sample a state s' , reward r and observation z given a state-action pair. However, both the simulation and rollout are developed using states sampled from the particle filter.

In detail, the POMCP algorithm starts with the *initialisation of the tree* \mathbf{T} by creating the root $(h, \mathcal{V}(h), \mathcal{N}(h), \mathcal{F}_h)$ with $\mathcal{V}(h) = 0$, $\mathcal{N}(h) = 0$ and $\mathcal{F}_h = \emptyset$. Subsequently, we *initialise the particle filter of our root node* \mathcal{F}_h , generating k possible states from which we can simulate the agents' actions and search for a solution to the target problem. Initially, the beliefs are generated through a uniform distribution \mathcal{U} but, if some prior knowledge about the problem's belief state distribution is available, a refined distribution can be used instead.

The search process is iterative, as in the MCTS's search procedure. Differently from MCTS, we need to sample a state from our current belief (approximated by the particle filter) to perform multiple rounds of reasoning. While choosing actions, POMCP will usually employ the UCB1 function and perform a similar simulation

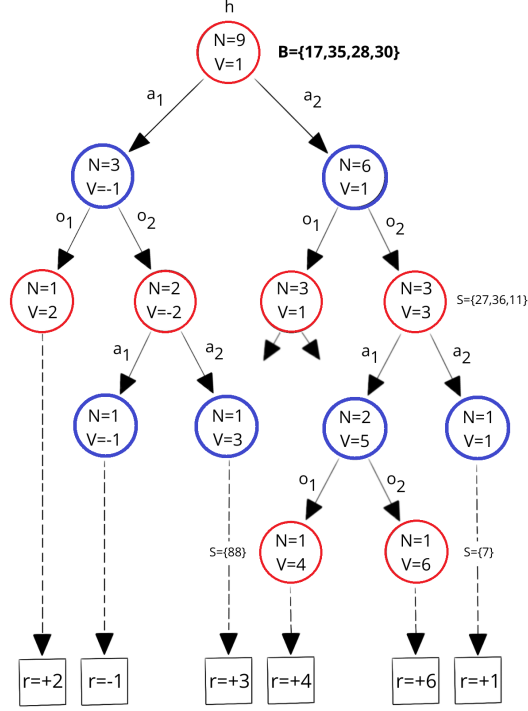


Figure 2.2: Illustration of POMCP’s tree structure. The red circles indicate action nodes and the blue circles indicate observation nodes. Action nodes are connected to observations nodes through actions and observation nodes are connected action nodes through observations. B represents the belief state set of the root node (each number represent a possible state). N represents the number of visits and V the current value estimated to each specific node. Each box at the bottom of the figure presents the reward r found in the last node.

and rollout process to MCTS. At the end of each simulation or rollout, it performs the *reward back-propagation*. However, when finishing a simulation, the current state s used in the simulation for the node h will be added to the particle filter \mathcal{F}_h of the node. This strategy maintains and updates the belief’s estimation at all levels in the tree. When we finish the search, we *select the best action* based on the estimated rewards (Equation 2.7) of each node to determine the most promising path to follow in the real environment.

After taking an action and receiving a new real observation from the environment,

we restart the POMCP algorithm. However, the initial steps are slightly changed to maintain the knowledge and update the current information, i.e., perform online planning. Unlike the first iteration, we now *update the root node* by traversing the tree based on the action taken and the most recent observation from the environment and, we *update the particle filter of our root node* by reinvigorating our belief, generating new states using the uniform distribution while maintaining promising states for simulation. Then, after performing these updates, we restart the search and repeat the whole process. Algorithm 2 illustrates the POMCP’s implementation.

Algorithm 2 POMCP’s Planning

<p>1: procedure SEARCH(h_τ)</p> <p>2: while Timeout() is False do</p> <p>3: if $\mathcal{F}_\tau = \emptyset$ then $s \sim \mathcal{U}_z$</p> <p>4: else $s \sim \mathcal{F}_\tau$</p> <p>5: Simulate($s, h_\tau, 0$)</p> <p>6: return $\arg \max_{a \in \mathbf{A}} \mathcal{V}(ha)$</p> <p>1: procedure EXPAND_NODE(h)</p> <p>2: for $a \in \mathbf{A}$ do</p> <p>3: $\mathbf{T}(ha) \leftarrow (ha, \mathcal{V}_{init}(ha), \mathcal{N}_{init}(ha), \emptyset)$</p> <p>1: procedure ROLLOUT(s, h, d)</p> <p>2: if $d < depth_{max}$ then return 0</p> <p>3: $a \sim \pi_{rollout}(h, \cdot)$</p> <p>4: $(s', z, r) \sim \mathcal{G}(s, a)$</p> <p>5: return $r + \gamma Rollout(s', haz, d + 1)$</p>	<p>1: procedure SIMULATE(s, h, d)</p> <p>2: if $d < depth_{max}$ then</p> <p>3: return 0</p> <p>4: else if is_terminal(s) then</p> <p>5: return 0</p> <p>6: if $h \notin \mathbf{T}$ then</p> <p>7: Expand_Node(h)</p> <p>8: return Rollout(s, h, d)</p> <p>9: $a \leftarrow \arg \max_{a \in \mathbf{A}} \text{UCB1}(h, a)$</p> <p>10: $(s', z, r) \sim \mathcal{G}(s, a)$</p> <p>11: $R \leftarrow r + \gamma \text{Simulate}(s', haz, d + 1)$</p> <p>12: $\mathcal{F}_h \leftarrow \mathcal{F}_h \cup \{s\}$</p> <p>13: $\mathcal{N}(h) \leftarrow \mathcal{N}(h) + 1$</p> <p>14: $\mathcal{N}(ha) \leftarrow \mathcal{N}(ha) + 1$</p> <p>15: $\mathcal{V}(ha) \leftarrow \mathcal{V}(ha) + \frac{R - \mathcal{V}(ha)}{\mathcal{N}(ha)}$</p> <p>16: return R</p>
--	---

2.2.3 Single-agent Model Approach for Ad-hoc Teamwork

As we discussed above, for the OEATE’s methodology [28] (which will be presented in details in Chapter 5), we will be using a single-agent Markovian model to represent the world. We consider that, given a state s , a non-strategic agent $\omega \in \Omega$ has an unknown

pdf across a set of actions defined by its internal features. This pdf is going to affect the probability of the next state because it defines each agent’s transition function. Therefore, we can say that the uncertainty in the Markovian model comes from the randomness of the actions of the ω agents, besides any additional stochasticity of the environment. Additionally, two main features define an agent’s pdf: its type $\theta \in \Theta$ and its parameters \mathbf{p} . In general lines, the type θ of an agent is a policy function π^θ that, given a state s and a vector of parameters \mathbf{p} , returns the actions probabilities for the agent. Mathematically, it can be described by $\pi^\theta(s, a, \mathbf{p}) = [0, 1]$.

Considering the application of this model in a Monte-Carlo tree search-based algorithm, at each node transition, ϕ samples ω agents’ actions from their (estimated) pdfs across types and parameters. The current outcome from the pdfs will determine the next state s' for the next node in the tree. Note that the agents’ types and parameters are not observable, but in our MDP model, that is not directly considered. Estimated types and parameters are used during online planning, creating an estimated transition function. The actual decisions made by the non-strategic agents are observable in real-world transitions without any direct information about type and parameters (i.e., information exchange between agents).

As mentioned earlier, in the OEATE’s task-based ad-hoc team, ϕ attempts to help the team get the highest possible reward. For this reason, ϕ needs to find the optimal value function, which maximises the expected sum of discounted rewards $E[\sum_{j=0}^{\infty} \gamma^j r_{t+j}]$, where t is the current time, r_{t+j} is the reward ϕ receives at j steps in the future, $\gamma \in (0, 1]$ is a discount factor. Also, we consider that we obtain the rewards by solving the tasks $\tau \in \mathbb{T}$. That is, we define ϕ ’s reward as $\sum r(\tau)$, where $r(\tau)$ is the reward obtained after the task τ completion. Note that the sum of rewards is not only across the tasks completed by ϕ , but all tasks completed by any set of agents in a given state. Furthermore, there might be some tasks in the system that cannot be completed without cooperation between the agents, so the number of required agents for finishing a task τ depends on each specific task and the set of agents that are jointly trying to complete it.

2.2.4 Adversarial Tree-search-based Approach

The A-MCTS, as the traditional MCTS, aims to find the optimal action a^* for a given state s and agent by simulating the world steps within a tree structure. However, besides considering the perspective of the main agent ϕ , which is running the algorithm and reasoning about the world, it also considers the perspective of an adversary ψ taking actions and expanding the tree similarly to the main agent. Overall, the A-MCTS adapts the traditional 2-player MCTS idea, while using its architecture, to enable the representation and simulation of different potential adversaries taking actions in our adversarial context and defined environment [23].

In our model, each node in the A-MCTS's tree \mathbf{T}_ψ is represented by $(s, \mathcal{V}, \mathcal{N}, \phi, \psi)$, i.e., a tuple with a *state* s , a *value* $\mathcal{V}(s, a)$, a *visitation count* $\mathcal{N}(s, a)$ for each action $a \in \mathbf{A}$, the agent ϕ which will define the perspective of the main agent's actions simulation within the tree and the ψ , which will define the adversarial perspective in the tree search process. As for the traditional MCTS, the *value* of the node represents the expected cumulative reward for the simulated states and the number of visits to a state s is represented by $\mathcal{N}(s) = \sum_{a \in \mathbf{A}} \mathcal{N}(s, a)$.

The inclusion of these two different perspectives in the tree differentiates the MCTS from the A-MCTS approach by the definition of a minimisation and maximisation (min-max) search process in the tree; with ϕ trying to maximise the performance of the team and ψ agent trying to minimise performance by disrupting the coordination of the team. In terms of tree structure, the A-MCTS is similar to the MCTS, where every node is connected by action to the next ones, but it presents a similar alternation of nodes as in the POMCP's tree. However, instead of alternating action and observation nodes, it alternates ϕ and ψ nodes.

As explained in Section 2.2.1, while performing simulations within an A-MCTS, each state in the search tree is also viewed as a multi-armed bandit. In a traditional maximisation problem, *UCB1* tries to increase the value of less-explored actions by attaching a bonus inversely proportional to the number of times each action is tried (Equation 2.5). Analogously, while solving a minimisation problem, the UCB1

function considers the subtraction of the exploration value to correctly balance the exploration and exploitation levels, as the following:

$$UCB1_{min}(s, a) := \mathcal{V}(s, a) - c\sqrt{\frac{\log(\mathcal{N}(s))}{\mathcal{N}(s, a)}} \quad (2.8)$$

Algorithm-wise, the A-MCTS follows the same procedure explained for the MCTS (Section 2.2.1), considering the initialisation of the tree, a search process with simulations and rollouts and, the selection of the best action before restarting the process. However, while performing the tree expansion, it considers the sequence of actions taken by ϕ and ψ to find the best path that maximises or minimises reward collection. Therefore, the path in the tree is the succession of action pairs taken by ϕ and ψ instead of a sequence of actions taken only by the ϕ agent, such as in the MCTS. Mathematically, the sequence can be represented by $\{a_0^\phi a_0^\psi, a_1^\phi a_1^\psi, \dots, a_D^\phi a_D^\psi\}$, where D represents the max depth of the tree.

2.3 Information Analysis

Having the ability to analyse available information delivered by the environment stands as a relevant feature for agents performing decision-making in partially observable scenarios. This capacity becomes particularly significant because it offers plenty of opportunities to enhance an agent’s performance, even when no clear path is available to achieve it. When we discuss the application of information analysis, our focus lies on the implementation of external or auxiliary methods integrated into online planning algorithms to support reasoning, mainly through the development of statistical approaches. In essence, this aligns with several works in the current state-of-the-art of computing, which focus on the *Data Science* development to provide valuable contributions to the community.

Therefore, this section is dedicated to a detailed exploration of the statistical approaches employed in this thesis, with a specific emphasis on enhancing online planning under uncertainty. We commence by introducing an entropy analysis

technique, presenting the application and development of Shannon’s Entropy solutions (Section 2.3.1), a key component in the IB-POMCP proposal. Subsequently, we present an estimation technique employed in BAE – the Bayesian inference approach (Section 2.3.2). Notably, OEATE is not present in this section, as its approach considers the application of a Genetic Algorithm-inspired approach, which we adapted to aggregate information and navigate the scenarios with uncertainty, mainly regarding the estimation of types and parameters for unknown teammates.

2.3.1 Shannon’s Entropy

The calculation of entropy in scenarios with uncertainty stands out as a powerful approach because it can provide a quantifiable measure to assess the randomness level in a decision-making process. In general lines, we can say that entropy can measure the “surprise” or “ambiguity” present in an environment. By quantifying uncertainty, calculating entropy enables decision-makers and algorithms to prioritise actions, allocate resources, and make informed choices in situations where clarity is elusive, hence, it can contribute significantly to the enhancement of performance in complex and dynamic environments. A traditional approach to measure uncertainty in stochastic processes is using the Shannon’s Entropy equation [97], described by:

$$\mathcal{H}(X) = - \sum_{i=1}^n P(x_i) \log(P(x_i)), \quad (2.9)$$

where X is a discrete random variable and x_i are the possible outcomes from X . Again, the core idea of this approach is to find a reliable measure for “information”, that is, to calculate the degree of *surprise* of an event given the available space of possibilities. If an event often occurs, the information inside the event is likely not “novel”. Hence, there is no “surprise” when it occurs. On the other hand, if an event rarely occurs, the “surprise” is greater. Comparing both situations, we can follow events with higher surprise to explore the space of possibilities, and safely follow events with lower surprise to exploit the environment, hence, we can adapt an agent’s behaviour according to the entropy of the current state.

For IB-POMCP, we adapt Shannon’s Entropy to retrieve information for the POMDP model’s observation space and show that it is possible to surpass the limitation of traditional reward-guided planning methods and perform an efficient decision-making process even when no reward is available in the reasoning horizon. All details about our modifications and implementation is presented in Chapter 4.

2.3.2 Bayesian Inference

Bayesian inference is a powerful tool that has shown diverse benefits when applied together with decision-making processes. By calculating and updating the probability of a hypothetical event, this statistical tool can approximate the uncertainty of the environment as more evidence or information becomes available, perfectly fitting online planning solutions. However, unlike traditional approaches, which solely rely on observed data, Bayesian inference incorporates prior knowledge or beliefs about a situation, allowing the building of a more flexible and robust analysis of the uncertainty. Mathematically, it is represented by the equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.10)$$

In the equation, A represents the hypothesis and B is the observation or evidence from the model. Hence, $P(A)$ is the *prior probability*, $P(B)$ the *marginal likelihood*, $P(B|A)$ the *likelihood* of observing B given A , and $P(A|B)$ our *posterior probability*. In this thesis, we suggest its application together with our adversarial estimation solution, BAE (Chapter 6). We develop its application to estimate the probability of an agent being an adversary (hypothesis), given the action performed by it in the real world (evidence). By adapting our tree search process together with the idea of roles in a Multi-Agent Reinforcement Learning (MARL) system, modelled as an MMDP, we enable the estimation of teammates’ and adversaries’ intentions.

2.4 Estimation Methods

In the literature, several different methods are proposed for the estimation or approximation of latent features of unknown agents in collaborative environments. As discussed before, learning these features and approximating their behaviour can enhance an ad-hoc agent capability to coordinate and cooperate with potential teammates. Therefore, in this section, we introduce two different type and parameters estimation methods used as baselines in this thesis: AGA and ABU. Both algorithms will be evaluated under different experimental settings in Chapter 5 and 6, where we introduce OEATE and BAE, respectively.

- **Algorithms Outline** – The Approximate Gradient Ascent (AGA), and the Approximate Bayesian Update (ABU) estimation methods are introduced by Albrecht and Stone (2017) [5]. In that work, the probability of taking the action a_ω^t at time step t , for agent ω , is defined as $P(a_\omega^t | H_\omega^t, \Theta_i, \mathbf{p})$, where $H_\omega^t = (s_i^0, \dots, s_i^t)$ is the ω agent’s history of observations at time step t , Θ_i is a type in Θ , and \mathbf{p} is the parameter vector which is estimated for type Θ_i . For the estimation methods, a function f is defined as $f(\mathbf{p}) = P(a_\omega^{t-1} | H_\omega^{t-1}, \Theta_i, \mathbf{p})$ where $f(\mathbf{p})$ represents the probability of the agents’ previous action a_ω^{t-1} , given the history of observations of agent ω in previous time step, H_ω^{t-1} , type Θ_i , and its corresponding parameter vector \mathbf{p} . After estimating the parameter \mathbf{p} for agent ω for the selected type Θ_i , the probability of having type Θ_i is updated following:

$$P(\Theta_i | H_\omega^t) \propto P(a_\omega^{t-1} | H_\omega^{t-1}, \Theta_i, \mathbf{p}) \times P(\Theta_i | H_\omega^{t-1}) \quad (2.11)$$

Iteratively, they showed that both methods are capable of approximate the type and parameters and improve the performance in the ad-hoc teamwork context.

- **AGA** – The main idea of this method is to update the estimated parameters of an agent ω by following the gradient of a type’s action probabilities based on its

parameter values. Algorithm 3 provides a summary of this method.

Algorithm 3 Approximate Gradient Ascent

- 1: **procedure** AGA ESTIMATION(\mathbf{p}^{t-1}, d)
 - 2: Collect samples $\mathbf{D} = (\mathbf{p}^{(l)}, f(\mathbf{p}^{(l)}))$
 - 3: Fit polynomial \hat{f} of degree d to \mathbf{D}
 - 4: Compute gradient $\nabla \hat{f}(\mathbf{p}^{t-1})$ and step size λ^t
 - 5: Update estimate \mathbf{p}^t
-

First of all, the method collects samples $(\mathbf{p}^{(l)}, f(\mathbf{p}^{(l)}))$, and stores them in a set \mathbf{D} (Line 2). The method for collection could be, for example, using a uniform grid over the parameter space that includes the boundary points. After collecting a set of samples, the algorithm, in Line 3, fits a polynomial \hat{f} of some specified degree d according to the collected samples. By fitting \hat{f} , the gradient $\nabla \hat{f}$ with some suitably chosen step size λ^t is calculated in the next Line 4. At the end, in Line 5, the estimated parameter is updated as presented in Equation 2.12.

$$\mathbf{p}^t = \mathbf{p}^{t-1} + \lambda^t \nabla \hat{f}(\mathbf{p}^{t-1}) \quad (2.12)$$

These steps describe AGA to estimate the agent's parameters and type iteratively. For further details, we recommend reading Albrecht and Stone (2017) [5].

- **ABU** – In this method, rather than using \hat{f} to perform gradient-based updates, Albrecht and Stone use \hat{f} to perform Bayesian updates that retain information from past updates. Hence, in addition to the belief $\mathbf{P}(\Theta_i | H_\omega^t)$, agent ϕ now also has a belief $\mathbf{P}(\mathbf{p} | H_\omega^t, \Theta_i)$ to quantify the relative likelihood of parameter values \mathbf{p} , for agent ω , when considering type Θ_i . This new belief is represented as a polynomial of the same degree d as \hat{f} . Algorithm 4 provides a summary of the ABU method.

Algorithm 4 Approximate Bayesian

- 1: **procedure** ABU ESTIMATION(\mathbf{p})
 - 2: Fit \hat{f} to f as in Algorithm 3
 - 3: Compute polynomial product $\hat{g} = \hat{f} \cdot \mathbf{P}(\mathbf{p}|H_\omega^{t-1}, \theta_i)$
 - 4: Collect samples $\mathbf{D} = (\mathbf{p}^{(l)}, \hat{g}(\mathbf{p}^{(l)}))$
 - 5: Fit new polynomial \hat{h} of degree d to \mathbf{D}
 - 6: Compute integral $I = \int_{\mathbf{p}_{min}}^{\mathbf{p}_{max}} \hat{h}(\mathbf{p}) d\mathbf{p}$
 - 7: Set new belief $\mathbf{P}(\mathbf{p}|H_\omega^t, \theta_i) = \hat{h}/I$
 - 8: Extract estimate \mathbf{p}^t from $\mathbf{P}(\mathbf{p}|H_\omega^t, \theta_i)$
-

After fitting \hat{f} (Line 2), the polynomial convolution of $\mathbf{P}(\mathbf{p}|H_\omega^{t-1}, \Theta_i)$ and \hat{f} results in a polynomial \hat{g} of degree greater than d (Line 3). Afterwards, in Line 4, a set of sample points is collected from the convolution \hat{g} in the same way that is done in Approximate Gradient Ascent, and a new polynomial \hat{h} of degree d is fitted to the collected set in Line 5. Finally, the integral of \hat{h} under the parameter space, and the division of \hat{h} by the integral is calculated, to obtain the new belief $\mathbf{P}(\mathbf{p}|H_\omega^t, \Theta_i)$. This new belief can then be used to obtain a parameter estimation, e.g., by finding the maximum of the polynomial or by sampling from the polynomial. For further details, we also recommend reading Albrecht and Stone (2017) [5].

Chapter 3

Related Works

In this chapter, we will carry out a comprehensive discussion about the state-of-the-art contributions and how different approaches have inspired this thesis and research projects. Intending to facilitate understanding and readability, we organised the chapter into sections and related contributions by groups. Each section categorises the major idea of each group and summarises the main strategy of those.

In Section 3.1, we review pertinent proposals in the literature for online planning under uncertainty, which considers the presentation of latent function approximation methods (Section 3.1.1), algorithms employing belief-dependent approaches for value approximation (Section 3.1.2), time-constrained planning algorithms (Section 3.1.3), entropy-based methods (Section 3.1.4), a discussion surrounding information-based sampling approaches (Section 3.1.5) and alternative options for online planning algorithms under partial observability, beyond POMCP (Section 3.1.6).

Moving on to Section 3.2, we delve into significant contributions from the teamwork and ad-hoc teamwork literature relevant to this thesis. This includes discussions on type-based parameter estimation algorithms for handling unknown teammates (Section 3.2.1), the application of complex models for learning environmental features, such as the transition function (Section 3.2.2), the development of task-oriented approaches to solving distributed task problems (Section 3.2.3), the representation of ad-hoc teamwork problems in a MRTA model (Section 3.2.4), and the application

of genetic algorithm strategies for type and parameter estimation (Section 3.2.5).

Finally, to conclude this chapter in Section 3.3, we address the Adversarial Identification problem and explore existing state-of-the-art solutions. We start the discussion with the application of task-based algorithms and teamwork models extended for the adversarial context and detection problem (3.3.1). Subsequently, we delve into the application of neural network solutions for the identification of adversaries in the environment (Section 3.3.2).

3.1 Online Planning under Uncertainty

In the current state-of-the-art, POMCP [93] remains a relevant solution for problems that require the application of intelligent agents to perform online planning and handle uncertainty at the same time. Due to its adaptability and problem-solving capability across the most diverse domains, POMCP is still inspiring different works in the literature and presenting fresh solutions for the community [7, 71, 73, 113]. However, many of these proposals are tailored to specific problems, leaving room for improvement, especially when handling sparse reward scenarios.

One cause of this constraint is the recurrent “reward-guided paradigm” in these methods’ planning. Most of the current solutions focus on planning solely through the collection of rewards to value actions. Consequently, if no reward is available within the planning horizon, making decisions becomes challenging and expensive.

The Upper Confidence Bound (UCB1) [63] is a relevant algorithm in the literature for evaluating and selecting actions in online planning solutions and is recurrently used inside Monte-Carlo decision-making frameworks [51, 86, 96]. This method, besides being useful, is still attached to the aforementioned “reward-guided paradigm” of planning. Note that, there are also other action selection methods available in the literature, such as the Thompson Sampling method [9] and the PUCT algorithm [94], and they also still present the same constraint.

In this thesis, we propose a novel approach that is capable of addressing this

gap by shifting POMCP and UCB1’s reward-guided planning approach to our “*information-guided planning strategy*”. Our new approach can leverage observation space entropy (information) and guide the agent to promising spots from where it easily solves such scenarios. IB-POMCP, together with our novel I-UCB function, present our advancement in this area, besides introducing a new perspective to the community in terms of developing online planning algorithms under partial observability. Moreover, our solution is lightweight and generic to tackle Partially Observable Markov Decision Problems (POMDPs). Our proposed *I-UCB*, differently from PUCT (*AlphaGo*) [94] for example, well-fits partially observable scenarios while enhancing POMCP planning capabilities through an entropy-based perspective, without relying on pre-trained models or a large amount of data.

3.1.1 Latent function approximation

One common strategy suggested by the literature to solve online planning problems while handling uncertainty is through the application of methods capable of approximating the latent world functions to handle the lack of knowledge [42, 45, 62, 96]. Katt, Oliehoek, and Amato (2019) [56], for example, propose the FBA-POMDP, a framework that can learn a compact model of the world’s dynamics of POMDPs. Through the planning process, they develop a belief tracking method based on the likelihood of belief that improves POMDP’s simulation quality, hence, the planning quality and algorithm performance. However, these approaches are directly bounded by the belief in space dimensionality, which can grow exponentially and hence require exponentially more resources as the complexity of the problem increases [55].

We propose the extraction of information using only the observations received from the simulations and the real world, improving the reasoning without requiring significantly more resources. In simpler terms, we intentionally designed our reasoning process to leverage the information gain generated from both simulated world scenarios in the agent’s mind and real-world observations. By the application of a statistical approach, we could retain the information value without demanding

additional data or previous training. Differently from the above solutions, our calculations are also well-designed to fit the online planning procedure, updating its value at each iteration without spending significantly more time.

3.1.2 Belief-dependent approaches

Some studies suggest the application of belief-dependent POMDP models to improve POMCP’s reasoning capabilities [8, 65]. Under certain assumptions, these models have shown great efficiency when solving problems with large action and observation spaces, since they are capable of boosting the decision-making process’s quality through the approximation of belief-dependent rewards to value actions for the agent during its planning and reasoning procedure.

A recent proposal that shows great advancement in this field is ρ -POMCP [99], an efficient online planner that boosts the POMCP’s search process by estimating belief-dependent rewards. This method proposes the propagation of more than one state during the simulation and the estimation of a single belief reward at the end of the process. This strategy has shown great capability in handling partially observable scenarios, mainly through the enhancement of the quality in estimating the true state of the problem. However, ρ -POMCP relies on explicit access to the observation and transition function to calculate its belief rewards. The application of these functions also leads the algorithm to struggle while reasoning under time constraints.

In order to handle time-constrained problems, they propose TB ρ -POMCP [99]. This version of ρ -POMCP requires the definition of a time budget (TB) to run, which will define the maximum time available for the algorithm to reason about the problem before choosing the best action. Although it solves the time constraint problem, it greatly penalises the algorithm’s efficacy in planning the best action.

Therefore, in this thesis, we show that it is possible to improve planning using less information and without significant impact on the reasoning time. Our solution, IB-POMCP, can perform planning faster and better than different baselines in several benchmarks without requiring knowledge about the observation and transition

functions to reason about the problem. This approach makes our solution generic since retrieving these world's functions can be challenging or even impossible depending on the target domains of analysis.

3.1.3 Time-constrained planning

Considering the above discussion, time-constrained problems are prevalent in the literature and often applied for the evaluation of online planning algorithms. In real-time strategy (RTS) games, a commonly suggested approach to tackle these problems considers the usage of macro and micro predefined coordination scripts to support the decision-making and handle multiple levels of coordination [74, 112]. Although solutions in RTS handle several levels of coordination, these methods still require the development of precise behavioural templates and complex policies to enable an efficient action planning procedure.

Note that, besides making decisions about the actions, RTS's methods also define a hierarchy to execute the plan considering the entire world state (i.e., macro and micro-states of the problem). This hierarchical model also appears in multi-agent swarm research, where they avoid developing a complex planning process to save time while estimating unknown features of the environment and reasoning about multiple levels of execution. Pelcner et al. (2020) [81], for example, suggest the application of a Flat Monte Carlo approach to achieve this objective.

In this thesis, we propose solutions where the execution of efficient planning under partial observability is possible from scratch for every problem without penalising the planning quality. That is, we do not require pre-training or the creation of auxiliary policy models to execute our reasoning process. IB-POMCP and BAE, for example, only require the definition of the Markovian model in order to start planning. On the other hand, OEATE requires templates to perform type estimation, however, we show that our approach is capable of improving planning even in the absence of good templates to approximate the agents' behaviour.

3.1.4 Entropy-based methods

Regarding the entropy application, Xiao et al. (2019) [108] propose the MENTS algorithm that augments the MCTS process using a maximum entropy policy optimisation, evaluating each search node by softmax values back-propagated from simulation. In a similar line of work and as a different version of MENTS, Kozakowski, Pacek, and Miloś (2022) [61] propose the ANTS algorithm which proposes an adaptive structure to the learning and planning process combining it with the maximum entropy paradigm for tree-search approaches. Moreover, to better perform the optimisation, ANTS suggests the application of Q-networks to initialise leaves and a pseudo reward shaping technique to handle the inflation and differences of backpropagated rewards in a first visit to a state. This technique mitigates the possibility of the planner to start deepening the same, single path, because it keeps receiving high values along it.

Both of these works under *fully observable models* and suggest the employment of entropy as a regulariser to augment the standard expected reward objective as in an optimisation problem. MENTS and ANTS, while running their optimisation, assume that their model provides the knowledge about the maximum number of steps at each episode, information which leads the algorithm to avoid the sparsity problem since it enables the agent to reason about the complete problem and the possible plans (sequence of actions) at every episode.

Trivially different from them, in the proposal of IB-POMCP (Chapter 4), we study the problem from the perspective of *partially observable models*. Instead of looking at the problem as a reward optimisation problem, we propose to augment the planning process by including the entropy value into consideration. In other words, we quantify the value of observations (delivered by a POMDP model) and add its value to the tree search process, avoiding working on the space of rewards and latent functions. Our adopted perspective also avoids problems related to the sparsity of rewards in the environment but relies on less information.

We also assume strong constraints to the model. Besides the partial observability, the agent must reason considering a limited number of steps (lower than the necessary

to solve the problem) and a limited number of times (in terms of the number of simulations performed during the search process). Additionally, we assume that every execution of the problem is performed from scratch and no knowledge is carried between executions, therefore, using trained networks or shaped initialisation surpasses our defined constraints.

3.1.5 Information-based Sampling Approaches

There is another line of research that aims at refining decision-making processes which explores the optimisation of agents' action selection approaches using entropy-based strategies, but for single-period problems, i.e., one action problems [49, 68, 104].

Russo and Roy (2014) [88] introduced the Information-Directed Sampling (IDS) method, which represents an action selection approach that leverages the mutual information between the true optimal action and the next observation in order to minimise the regret in single-period optimization problems. While IDS shares certain similarities with our proposed I-UCB approach, particularly in the application of an entropy-based technique for action selection, there are some key points that differentiates it from our work

For example, IDS requires a reliable approximation of transition function in order to optimise the action selection process. This requirement already distinguishes our work from IDS's proposal, since we assume that such knowledge may not be readily available to the agent before problem execution and/or can not be efficiently estimated during execution time. IDS utilises this knowledge to approximate the expected regret of its decision-making process within a finite time and subsequently optimise the search for the best action, a strategy akin to that proposed by Katt, Oliehoek, and Amato (2019) [56], albeit focusing on the action space as the optimisation target instead of the reward function. In contrast, I-UCB enhances the decision-making process without requiring any additional information about the problem besides the POMDP model.

Furthermore, while IDS shows some potential in its application to various problems

due to its generalised problem formulation for different multi-armed bandit settings, its adaptation to sequential decision-making problems may necessitate fine empirical and theoretical adjustments, particularly concerning the management of information for its execution under partial observability’s constraints. What we mean is, although IDS aims to minimise the expected regret over a finite simulation horizon for generic problems, performing the optimisation of its policy in sequential decision-making problems can be proved unreliable due to the size and nature of the belief state space in these domains, which can be non-stationary, and due to the aggregation of uncertainty across multiple steps, which can significantly increase the necessary time of making a decision. I-UCB was specifically designed to address these challenges within reasonable simulation and time horizons. Instead of directly modelling a latent function to be optimised in order to improve decision-making, IB-POMCP performs the approximation using its online sampling technique, which can handle the non-stationarity of the environment besides maintain the performance working under time constraints (i.e., small time windows to reason).

Finally, we recognise IDS as an important work, which can also contribute to improvements in IB-POMCP itself, and as a potential benchmark for evaluating the efficiency of our proposal in future works. However, since our current focus lies in studying sequential decision-making problems rather than single-period problems, and adapting IDS to our context would involve revisiting the method’s implementation besides its mathematical model, we opted not to include it in our study (Chapter 4).

3.1.6 Other options to POMCP

Finally, although we focus on improving POMCP-based methods, our contributions can also benefit other algorithms, such as DESPOT-based solutions [110].

Different from POMCP, DESPOT’s main focus is on filtering a large observation space to plan using a sparse approximation of the problem, hence, a smaller reasoning space. However, the algorithm still struggles in large action spaces, and/or sparse reward scenarios where the optimal policy could be long (i.e., requiring many steps

to be modelled). Our proposal addresses POMDPs featuring sparse rewards in an efficient manner while running the search process from scratch.

3.2 Teamwork and Ad-hoc Teamwork Literature

The literature introduces ad-hoc teamwork as a remarkable approach to model MAS [95, 2]. This approach presents the opportunity to achieve the objectives of multiple agents in a collaborative manner that surpasses the requirement of designing a communication channel for information exchange between the agents, building an application that does not rely on prior coordination protocols or the collection of previous data for training. Furthermore, these models enable the creation of algorithms capable of acting in an online fashion, dynamically adapting their behaviour according to the environment and current teammates.

3.2.1 Type-based parameter estimation

Type-based reasoning and *parameters learning* algorithms have also shown great capabilities in solving complex decision-making problems using fine-grained models, which evaluate the observations and estimate each agent's type and parameters to perform an online planning procedure [3, 4, 12, 14, 15]. These lines of work propose the approximation of agents' behaviour to a set of potential types in order to improve the ad-hoc agents' decision-making capabilities. Moreover, they also allow a quick online estimation of agents' strategies without requiring an expensive training process for learning their policies from scratch. However, if a set of potential types and the parameter space cannot be defined through domain knowledge, they would have to be learned from previous interactions [15].

Albrecht and Stone (2017) [5], in particular, introduced the *AGA* and *ABU* algorithms for *type-based* reasoning of teammates parameters in an on-line manner, which are the main inspirations for this work. Both methods sample sets of parameters (from a defined parameter space) to perform estimations via *gradient*

ascent and *Bayesian* updates, respectively. However, by focusing on decentralised task execution in ad-hoc teams, our *novel* method surpasses their parameter and type estimations when the number of teammates gets larger or more tasks are accomplished, consequently leading to better team performance. We have extended their work with OEATE, enabling the execution in partially observable environments, and BAE, handling unknown templates and adversaries.

On the other hand, Hayashi et al. (2020) [48], propose an enhanced particle reinvigorating process that leverages prior experiences encoded in a recurrent neural network (RNN), acting into a partial observable scenario in their ad-hoc team. However, they need thousands of previous experiences for training the RNN, while still requiring knowledge of the potential types. In this thesis, all our approaches can start from scratch at every single run without requiring pre-training.

Concerning problems with partial observability, POMCP is usually employed for online planning [93]. However, it was originally designed for a discrete state space, making it harder to apply POMCP for (continuous) parameter estimation. We apply POMCP in combination with our algorithm OEATE, which enables the decision-making on partial observable scenarios and improves the POMCP search space, given the OEATE’s estimation of the agents’ parameters. We also evaluate experimentally the performance of POMCP for our problem without the embedding of parameter estimation algorithms.

3.2.2 Complex models application

Guez, Silver, and Dayan (2013) [46] proposed a Bayesian MCTS that tries to directly learn a transition function by sampling different potential MDP models and evaluating it while planning under uncertainty. Our planning approach in OEATE (inspired by Albrecht and Stone (2017) [5] and Barrett et al. (2013) [14]) is similar, as we sample different agent models from our estimations. However, instead of directly working on the complex transition function space, we learn agents’ types and parameters, which would then translate to a certain transition probability for the current state.

Rabinowitz et al. (2018) [83] introduce a “Machine Theory of Mind” - or purely the Theory of Mind (ToM) approach –, where neural networks are trained in general populations to learn agent types, and the current agent behaviour is then estimated in an on-line manner. Similarly to learning policies from scratch, however, their general models require thousands (even millions) of observations to be trained. Besides, they used a small 11×11 grid in their experiments, while we scaled all the way to 45×45 to estimate the behaviour of several unknown and distinct teammates. On the other hand, if a set of potential types is not given by domain knowledge, then their work serves as another example of how types could be learned.

A different approach that enables the learning of teammates’ models and reasoning about their behaviour in planning is given by I-POMDP-based models [44, 36, 50, 30]. However, they are computationally expensive, assuming all agents are learning about others recursively and considering agents that receive individual rewards (processing estimations individually). All our proposed methods in this thesis represent lightweight solutions that do not require a large amount of data, memory or an exhaustive time to run.

Eck et al. (2020) [38] addressed this problem and recently proposed a scalable approach using the I-POMDP-Lite Framework in order to consider large open agent systems. In their approach, an agent considers a large population by modelling a representative set of neighbours. They focus on estimating how many agents perform a particular action, hence their approach is not applicable to the task-based problems that we consider in this work. Additionally, although they present a scalable approach in terms of team size, they still consider only small 3×3 scenarios. In this thesis, we show scalability regarding the team size, the dimensions of the map and the numbers of simultaneous tasks in the scenario for OEATE and BAE.

Rahman et al. (2020) [84] also handle open agent problems and propose the application of a Graph Neural Network (GNN) for estimating agents behaviours. Similarly to other neural network-based models, it needs a large amount of training, and their results are limited to a 10×10 grid world with 5 agents. Their agent

parametrisation is also more limited, with only 3 possible levels in the level-based foraging domain, which is directly given as input for each agent (instead of learned).

Overall, regarding mainly the proposal and development of OEATE and BAE, we propose lighter MDP/POMDP models, focused on decentralised task execution, with a single team reward, that allows us to tackle problems with a larger number of agents, and tasks in bigger scenarios in the partially observable domain. Also, we build a model for every single member of the team. On the other hand, open agent systems are not in the scope of our work, and we consider fixed team sizes.

3.2.3 Task-oriented approaches

Decentralised task execution problems in ad-hoc teamwork are one of the key ideas of this thesis. Chen et al. (2019) [31] present a related approach, where they focus on estimating tasks of teammates, instead of learning their model. While related, they focus on task inference in a model-free approach, considering that each task must be performed by one agent, and the ad-hoc agent goal changes to identifying tasks that are not yet allocated. Our work, on the other hand, combines task-based inference with model-based approaches and allows for tasks to require an arbitrary number of agents. Additionally, their experiments are on small 10×10 grids and consider a lower number of agents than us under this configuration.

There are also works that attempt to identify the task being executed by a team [72]; or an agent’s strategy for solving a repetitive task, enabling the learner to perform collaborative actions [102]. Our work, however, is fundamentally different, since we focus on a set of (known) tasks which must be completed by the team.

Another approach suggested in the literature for task-based problem optimisation is the Multi-Agent Markov Decision Problem (MMDP) models [33, 34]. These models allow agents to decide their target task autonomously and are focused on estimating teammates’ policies directly at specific times in the problem execution. Given knowledge of the MMDP model, those approaches compute the best response policy (at the current time) for the other agents and use those models while planning.

However, they do not consider learning a probability distribution over potential types and estimating agents' parameters like in our approach. OEATE is capable of using a set of potential types and space of parameters to learn the probabilities of each type-parameter set up for each teammate in an online fashion.

Concerning task allocation, MDP-based models are commonly applied [75, 76] in the ad-hoc teamwork context. For instance, it can be framed as a multi-agent team decision problem [89], where a global planner calculates local policies for each agent. Auction-based approaches are also common, assigning tasks based on bids received from each agent [70]. These approaches, however, require pre-programmed coordination strategies, while we employ online learning and planning for ad-hoc teamwork in decentralised task execution, enabling agents to choose their tasks without relying on previous knowledge of the other team members, and without requiring an allocation by centralised planners/controllers.

3.2.4 MRTA model for ad-hoc teamwork

Multi-Robot Task Allocation (MRTA) models also represent an alternative approach to solving problems in the ad-hoc teamwork context [66, 106]. Intending to maximise the collective completion of tasks, these models employ decentralised task execution strategies that work in an online manner without a central learning agent. Each agent develops its own strategy based on the received observations. Similarly to our proposal, MRTA models implement a task-based perspective to deliver solutions where agents know and seek tasks distributed in an environment while reasoning. However, MRTA models assume knowledge about the teammates' types and the tasks that they are pursuing. Furthermore, this assumption holds because they consider this information to be available in the environment, where agents can get it through observation (e.g., agents choosing tasks of different colours) or reliable communication channels for information exchange between the agents. As we mentioned earlier, there are circumstances where communication channels are unreliable, and agents cannot fully trust them to send or receive information. OEATE predicts their teammates'

targets while learning their types and parameters, besides handling problems where these assumptions are not secured. BAE is also capable of estimating the true adversary without relying on communication channels.

3.2.5 Genetic algorithms' inspiration for OEATE

OEATE is inspired by Genetic Algorithms (GA) [52] since our main idea is to keep a set of *estimators*, generating new ones either randomly or using information from previously selected *estimators*. However, GAs evaluate all individuals simultaneously at each generation, and usually, they are selected to stay in the new population or for elimination according to its fitness function. Our *estimators*, on the other hand, are evaluated per agent at every task completion, and survive according to the success rate. The proportion of survived *estimators* are then used for type estimation, and new ones are generated using a similar approach to the usual GA mutation/crossover. Moreover, we choose the application of GA concepts in the works considering our empirical and theoretical results. As an empirical result, the employment of the GA approach showed better results in comparison with the Bayesian Updates (considering the performance of AGA and ABU against OEATE). As a theoretical result, our solution does not depend on finite-dimensional representations for parameter-action relationships and can provide a more robust way to explore the whole parameter space by using multiple estimators, which mutate to form even better estimators.

3.3 Adversarial Identification among a Team

The estimation of the latent features of the environment, including features from other agents sharing the same scenario, presents noteworthy contributions to the Artificial Intelligence (AI) community [2, 48]. Overall, these solutions improve coordination by boosting the intelligent agent's knowledge of their teammates and the surrounding world, as discussed in Section 3.2. However, they typically fail when their models for types and world representation do not adequately fit the target problem, especially

when facing agents running strategic models (such as an adversary). Note that, when we say “strategic”, we refer to models where agents are capable of modelling other agents in their reasoning process while “non-strategic” agents do not model teammates and typically follow fixed rules for planning [107].

3.3.1 Task-based algorithms and teamwork models in the adversarial detection context

AGA and ABU base their estimation on sampling and testing a set of parameters that approximate the type probabilities for each agent in the environment [5] using *gradient ascent* or *bayesian* updates for estimation, respectively. However, these methods are constrained by the quality of their templates (which are usually reactive model templates working in limited parameter spaces) because they apply this type-based approach to handle the problem. If there is no good template to estimate an agent’s behaviour (e.g. an adversary), the method might fail. Note that, this limitation also affects our proposal, OEATE (Chapter 5), which runs a similar approach to AGA and ABU under the task-oriented teamwork model. Noticing this constraint, we develop our research further and propose BAE with a new approach for estimating adversaries without relying on template models, coined Q-valued Bayesian Estimation (QvBE). We suggest the application of this approach inside BAE, requiring only the MDP model to identify adversaries and, consequently, improve planning.

Another method developed from a previous version of OEATE, the Online Estimation Ad-hoc Teamwork Allocation (OEATA) algorithm [91], is OEATA with Adversary (OEATA-A), which is a task-oriented estimation method that not only considers type and parameter estimation using predefined templates but also focuses on identifying “adversarial” agents within a team [90]. While the paper reports promising results, it presents three crucial problems: **(i)** Its adversarial detection strategy does not directly impact or improve the ad-hoc agent’s planning process because it represents a parallel process to planning, **(ii)** It often misclassifies idle or less capable agents as adversaries due to its task-oriented estimation approach,

i.e., agents that rarely accomplish tasks are classified as adversaries, even if their intention is to benefit the team, and; **(iii)** OEATA-A relies on hard assumptions to ensure its estimation method works in ad-hoc estimation scenarios, for example, assuming that every template will fail to estimate the actions of adversarial agents in the team. In order to avoid these problems, we propose an algorithm capable of integrating planning and estimation results in order to improve performance in an online manner and based on its current world knowledge. We show empirically that BAE is capable of correctly spotting the impostor among the teammates, if it exists, without relying on hard assumptions about the teammates and under different teamwork settings.

3.3.2 Neural network solutions for adversary detection

Detecting adversaries is also relevant when researching neural network solutions [67, 57]. In these works, a common strategy is to use action distribution information to validate the integrity of their decision-making and prediction processes. They proposed this validation using training data and previous knowledge about the features of their adversaries. Hence, if there is a lack of knowledge or training data for retrieving a robust set for validation, these methods may fail. Our method can run estimations from scratch; therefore, it does not rely on previous data or information. Furthermore, we use the action distribution for adversarial estimation, but we do not require a comparison to previously known templates. In particular, BAE detects the adversary by considering the difference between each agent’s adversarial probability across its teammates.

Kopparapu et al. (2022) [60] propose the “Hidden Agenda” game, inspired by the popular multiplayer deduction game “Among Us”. This N-player reinforcement learning (RL) environment pits Crewmates against Impostors. Crewmates aim to complete tasks as quickly as possible, while Impostors aim to prevent them from achieving this objective. Crewmates are unaware of others’ roles, whereas Impostors possess full knowledge. Prior solutions for this game leverage robust

RL neural networks to enable task completion, adversary deduction, and accurate voting. However, the solution’s efficacy depends on the quality of the training data. Unfortunately, the “Hidden Agenda” game is not publicly available for the community. However, our Level-based Foraging environment is similar to the game in terms of action and observation space size, besides the adversarial behaviour. The main difference is the existence of a voting phase where the agents actively vote to expel an agent from the scenario — an ability out of this research’s focus —, besides our assumption that agents have no prior information or data and must learn only with the data available while making decisions on the fly. Since our objective is to provide a solution that can find good actions despite the presence of a hidden adversary, we do not perform experiments on the “Hidden Agenda”.

Carminati et al. (2023) [24] have initiated the formal study of hidden-role games (for example, the Mafia/Werewolf family games, Avalon and Hidden Agenda) from a game-theoretic perspective. Mathematically, they define a notion of equilibrium and computation efficiency of algorithms in these games. Empirically, they show how their mathematical fit real-world instances of Avalon and show solutions that consider the application of a parallelised version of the PCFR+ algorithm [39] and the implementation of a simplex algorithm. These solutions, despite providing good mathematical guarantees and reliable estimation for equilibrium, require the application of large computational resources in order to solve the problem (CPU compute cluster with 64 CPUs and 480 GB RAM). Our proposal solves our proposed hidden-role problem, implemented in a Level-based Foraging Environment, with fewer mathematical guarantees but using significantly fewer computational resources to solve it. Moreover, we focus on providing an online planning solution while they propose an equilibrium solution.

Chapter 4

Information-based Partially Observable Monte-Carlo Planning

In this chapter, we present the technical details of IB-POMCP, our proposed online planning algorithm to handle partially observable scenarios with sparse rewards. This work was published at NeurIPS 2023 in the paper titled “Information-guided Planning: An Online Approach for Partially Observable Problems” [27].

4.1 Introduction

Decision-making agents are increasingly being used under uncertainty in nontrivial systems [55]. Based on the available information, such agents must evaluate potential decisions in the current environment to build a robust plan that either accomplishes the task or leads to a better situation. Consider, for instance, a scenario in which drones are deployed to rescue people in a hazardous environment. Drones need to quickly identify critical locations where they can offer support and save lives. If no clear target is found, they must rapidly formulate a plan based on their current knowledge and observable information to find an area in need of assistance from their current position.

The above context describes a partial observability problem that is recurrent in

the literature [5, 15, 43]. The Partially Observable Monte-Carlo Planning (POMCP) [93] algorithm is commonly suggested as a method to address these problems because it enables agents to perform planning in an online manner while handling uncertainty [10, 99, 109]. However, several state-of-the-art solutions that rely on POMCP often struggle when rewards are delivered sparsely in time or outside their reasoning horizon. For example, in foraging, there may be no clear target to spot from the current observation and/or estimated map so far. Here, we find a fundamental academic challenge: How can we improve agent performance when rewards are delivered sparsely in time or out of their reasoning horizon?

Every time an agent retrieves an observation from the world, it gains information that can be used to improve its internal planning process. Based on this, some works of the state-of-the-art suggest the improvement of traditional planning process by handling the lack of prior information using the approximation of the world’s dynamic models [54, 80], embedding supportive planning methods within the tree [5, 15], extracting non-explicit information from the observations [10, 54], enhancing the method’s inner calculations [1, 32, 77] or employing neural networks-based techniques to improve the search quality [61, 108].

Overall, these strategies can improve search quality within a defined simulation horizon by using extensions of traditional partially observable models and enabling the integration of additional knowledge into planning. However, such knowledge may not be available to agents beforehand (e.g., conditional observation probabilities), or may demand significantly more computational resources to perform the task (e.g., time and memory). While it is important to note that incorporating more knowledge benefits efficiency in terms of performance, it does not resolve challenges in sparse-reward scenarios, as most of the solutions remain reward-driven.

Hence, we present *Information-based POMCP (IB-POMCP)*, an online planning algorithm that uses a novel *information framework* to boost the decision-making process under partial observability even when no rewards are available in the reasoning horizon. Our framework refines the traditional UCB1 action selection strategy by

implementing our proposed *Information-guided UCB (I-UCB)* function, which is capable of leveraging entropy and estimating information gain, using observations delivered from both the real-world and from the simulations within the tree to identify promising states in the search process that leads the agent to quickly accomplish its objective. Its application together with our new particle filter reinvigoration strategy, which considers the current system’s entropy to calibrate the reinvigoration, indicates that this approach may lead the agent to act hybridly when solving a partially observable problem by better weighing the exploration, exploitation, and information levels within the tree. We ran experiments across five benchmarks and compared them with state-of-the-art baselines, obtaining significantly higher average rewards (up to 10 times) while performing faster reasoning (up to 93%).

4.2 Information-based Partially Observable Monte Carlo Planning

In this section, we present IB-POMCP, describing its details while discussing our rationale. We follow a similar presentation to Silver and Veness (2010) [93].

4.2.1 Algorithm Outline

IB-POMCP is a tree search algorithm designed for solving partially observable problems, which performs what we denominate as an *information-guided planning*, and enable better decision-making when dealing with uncertainty. It builds upon the POMCP algorithm but modifies its traditional procedure by incorporating into the PO-UCT algorithm our novel I-UCB function.

Our algorithm starts with the *(i) initialisation of our tree structure*, where we create our root and calculate the probability of stepping into this node given our current knowledge. Since the initial knowledge about the problem is assumed to be none, we can not calculate this probability, hence, we initialise it as 0. Subsequently,

we *(ii) initialise the particle filter of our root node*, generating possible states from which we can simulate the agents' actions and search for a solution to the target problem. Initially, the beliefs are generated through a uniform distribution.

Our search process is iterative, as commonly found in the literature [5, 93, 111], where we sample a state from our current belief to perform multiple rounds of simulation. However, unlike the typical approach, we propose the implementation of an adaptive exploration coefficient and of our novel I-UCB function instead of the usual UCB1. Therefore, at every iteration of our search (before starting a simulation), we first perform our *(iii) exploration coefficient adaptation*, adjusting it using the entropy (level of information) estimated over the set of observations collected through the simulations. Then, with the updated coefficient in hands, we start our *(iv) simulation with the I-UCB function*. While choosing actions, our proposal searches for solutions and takes actions based on observations' entropy besides the collection of rewards. At the end of each simulation, we re-update the exploration coefficient based on the information gained during the path traversal. When we finish the search, we *(v) select the best action* based on the estimated rewards, entropy, and the number of visits of each action's node to determine the most promising path to follow in the real environment.

After taking an action and receiving a new real observation from the environment, we restart the IB-POMCP algorithm. However, the initial steps are slightly changed to maintain the knowledge and update the current information, i.e., perform online planning. Unlike the first iteration, we now *(i) update the root node* by traversing the tree based on the action taken and the most recent observation from the environment. Consequently, we recalculate the probability of stepping into the new root node using the information available in the tree. With the probability in our hands, we now *(ii) update the particle filter of our root node* by reinvigorating our belief, generating new states using the uniform distribution while maintaining promising states for simulation. Then, after performing these updates, we restart the search and repeat the whole process. Now, let us discuss these steps in detail.

4.2.2 Root Initialisation and Update

This step, represented by (i) in the algorithm outline, is responsible for maintaining the tree structure \mathbf{T} coherent with the world and the information gained while performing online planning. Consider \mathbf{r} as the tree \mathbf{T} 's root node, $h_{\mathbf{r}}$ as the history of the root \mathbf{r} (our current history), a as the action taken by the agent in the real world and z as the most recent observation received from the real world (after the agent's action). **As in POMCP**, this step considers three possible procedures:

- **Initialisation:** *if the tree \mathbf{T} does not exist*, we initialise the tree by creating a root node \mathbf{r} using the current history $h_{\mathbf{r}}$, else;
- **Update:** *if \mathbf{T} exists and the nodes $h_{\mathbf{r}}a$ and $h_{\mathbf{r}}az$ also exist*, we walk in the existing tree, traversing the nodes that correspond to the actions taken by the agent a and the most recent world observation z , and update the root, i.e., the node $h_{\mathbf{r}}az$ becomes the new \mathbf{r} , hence, $h_{\mathbf{r}} = h_{\mathbf{r}}az$, else;
- **Re-initialisation:** *if \mathbf{T} exists but, node $h_{\mathbf{r}}a$ or $h_{\mathbf{r}}az$ does not exist*, the tree is re-initialised. That is, we create a new node $(h, \mathcal{V}(h), \mathcal{N}(h), \mathcal{F}_h)$ with h as the current history, $\mathcal{V}(h) = 0$, $\mathcal{N}(h) = 0$ and \mathcal{F}_h is empty. Afterwards, we assign this new node as our new root \mathbf{r} of \mathbf{T} , which means that all other simulations already made in the agent's head are discarded and we restart the algorithm.

In contrast to POMCP, we *estimate the current probability* $P(z|h_{\mathbf{r}}a)$, in this step, when finding the observation z after taking the action a considering our history $h_{\mathbf{r}}$. Overall, we aim to use this probability to enhance the Particle Filter Update (see step (ii)) procedure by weighting its diversification and reinvigoration levels according to the agent's uncertainty when stepping into the new root – which would be the same as calculating the agent's “surprise” on finding the received observation in the real world after taking the chosen action. However, we consider that a compact representation of transition \mathcal{T} or observation probabilities \mathcal{Z} may not be available for complex problems. Hence, we propose the approximation of this probability using the

knowledge within our particle filter \mathcal{F}_τ (without requiring a POMDP explicit model) by $P(z|h_\tau a) \approx \tilde{P}(z|h_\tau a) = \frac{\mathcal{N}(h_\tau a z)}{\mathcal{N}(h_\tau a)}$, where $\mathcal{N}(h_\tau a z)$ and $\mathcal{N}(h_\tau a)$ are the number of visits of node $h_\tau a z$ and $h_\tau a$, respectively (the new root node and its parent). For clarity, we use a different representation for \tilde{P} to indicate when we are using an estimated probability function in the calculations instead of the true probability P . If h_τ is empty or no new root node is found, $\tilde{P}(z|h_\tau a) = 0$. Note also that $\mathcal{N}(h a z) \leq \mathcal{N}(h a), \forall h \in \mathbf{T}$ and, consequently, $\mathcal{N}(h a z)/\mathcal{N}(h a) \in [0, 1]$.

4.2.3 Particle Filter Initialisation and Update

This process, represented by **(ii)** in the algorithm outline, is responsible for initialising the particle filter of the root node \mathcal{F}_τ (if it is empty or does not exist) or performing the particle reinvigoration process based on the probability $\tilde{P}(z|h_\tau a)$ calculated in the Root Update process (Section 4.2.2, step (i)).

Directly, the initialisation is made through the sampling of k particles (which generate the observation z) from the uniform distribution \mathcal{U}_z . On the other hand, the update considers the $\tilde{P}(z|h_\tau a)$ as the weight that balances the particle reinvigorating process over \mathcal{F}_τ . The idea is to diversify (or boost) the new root's particle filter as a reliable approximation of the belief state function. If the probability of stepping into this new root is high, i.e., $\tilde{P}(z|h_\tau a)$ is high, we assume that the particles in the new root's particle filter will well approximate the real world, since through the simulations we recurrently found the observation z after taking the action a from the last root node h . In contrast, when this probability is low, we diversify the new root's particle filter \mathcal{F}_τ by uniformly generating particles that may better represent the real world using \mathcal{U}_z instead of \mathcal{F}_τ . Consequently, coherent with the above rationale, we reinvigorate the new root's particle filter \mathcal{F}_τ by maintaining $\lfloor k\tilde{P}(z|h_\tau a) \rfloor$ particles sampled from itself, i.e., $\mathcal{F}_{h a z}$, and sampling new $\lfloor k(1 - \tilde{P}(z|h_\tau a)) \rfloor$ particles from the uniform distribution \mathcal{U}_z . This update on the new root's particle filter may offer a better start for the algorithm when performing the search process since the sampling of belief states may consider an enhanced set of particles through our

uncertainty-guided reinvigoration process. Algorithm 5 presents the pseudo-code for implementation of our proposed IB-POMCP’s particle filter reinvigoration strategy. We kindly refer the reader to Section 4.5 for further discussion about the novelty about this proposed enhancement and how it affects other components of IB-POMCP.

Algorithm 5 Particle Filter Update. The starred lines (with a light grey background) highlight the differences between POMCP and our proposal, IB-POMCP. *Union considers repetition here.*

```

1: procedure PARTICLEFILTERUPDATE( $\mathcal{F}_\tau, \tilde{P}(z|ha), k$ )
2:    $\hat{\mathbf{B}} \leftarrow \mathcal{F}_\tau, \mathcal{F}_\tau \leftarrow \{\}$  ▷  $\hat{\mathbf{B}}$  is a copy of  $\mathcal{F}_\tau$ 
*3:   while  $|\mathcal{F}_\tau| < \tilde{P}(z|ha)k$  do
*4:      $b \sim \hat{\mathbf{B}}, \mathcal{F}_\tau \leftarrow \mathcal{F}_\tau \cup b$  ▷ Adding  $\tilde{P}(z|ha)k$  particles to the new  $\mathcal{F}_\tau$ 
*5:   while  $|\mathcal{F}_\tau| < k$  do
*6:      $b \sim \mathcal{U}_z, \mathcal{F}_\tau \leftarrow \mathcal{F}_\tau \cup b$  ▷ Adding  $(1 - \tilde{P}(z|ha))k$  particles to the new  $\mathcal{F}_\tau$ 
*7:   return  $\mathcal{F}_\tau$ 

```

4.2.4 Updating the Tree Exploration Coefficient

To explain how we update our tree exploration coefficient, we first introduce the (a) *adaptation made to the traditional exploration coefficient*, then we present our (b) *modified entropy function*, which is used in the update, we explain how to calculate it and, in the end, we show our (c) *strategy to normalise the entropy* in the online planning context. This step is represented by (iii) in our algorithm outline.

(a) Exploration Coefficient Adaptation – Before the actual tree simulation process, IB-POMCP first adjusts the value of our tree’s exploration parameter based on the estimated entropy for the current system (the tree) in the root level \mathbf{r} with history h_τ . Directly, our approach considers the replacement of the traditional UCB1’s c constant (Equation 2.5) by the function $(1 - \alpha(h_\tau)) \in [0, 1]$, which we define as:

$$\alpha(h_{\tau}) := \frac{e \ln(\mathcal{N}(h_{\tau}))}{\mathcal{N}(h_{\tau})} \frac{\sum_{i=1}^{\mathcal{N}(h_{\tau})} \mathcal{H}_i(h_{\tau})}{\mathcal{N}(h_{\tau}) \max_{i=1}^{\mathcal{N}(h_{\tau})} \mathcal{H}_i(h_{\tau})} \quad (4.1)$$

Our insight is to use α to augment the current uncertainty level in the tree’s search policy. On the left-hand side of the multiplication, we design a function that represents the chance of finding new information by exploring a node, which decreases as the number of visits to the node increases. e is the Euler’s constant, which will be used as our amortisation factor for $\ln(\mathcal{N}(h_{\tau}))/\mathcal{N}(h_{\tau})$. Applying both together, we can describe a function that slowly decreases and maintains, in the infinity and under some assumptions, theoretical guarantees for belief approximation (see Section 4.3). The right-hand side of the multiplication expresses the “*general surprise trend*”, which is calculated through the division between the actual cumulative entropy $\sum_{i=1}^{\mathcal{N}(h_{\tau})} \mathcal{H}_i(h_{\tau})$ and the estimated maximum cumulative entropy $\mathcal{N}(h_{\tau}) \max_{i=1}^{\mathcal{N}(h_{\tau})} \mathcal{H}_i(h_{\tau})$, which is the multiplication of the total number of visits to the node and the maximum entropy. By multiplying both sides of the equation, we can estimate the current uncertainty of our system, in this case, of our tree. In addition to improving reasoning, we discard the need for prior knowledge about the problem of adjusting and fixing the tree constant c . Note that we adjust $\alpha(h_{\tau})$ for each traversal on the tree, i.e., for each simulation we perform from the root τ to some leaf in the tree.

(b) Entropy Calculation – We adapt Shannon’s Entropy (Equation 2.9) to measure the level of information in the IB-POMCP’s search process based on the collection of observations of our agent while performing Simulations (see step (iv)), which is designed as $\mathcal{H}(h) = - \sum_{z \in \tilde{\mathbf{Z}}_h} P_h(z) \ln(P_h(z))$, where $P_h(z)$ is the probability of finding the observation $z \in \tilde{\mathbf{Z}}_h$ by simulating actions from the node with history h . We use $\tilde{\mathbf{Z}}_h$, which is an estimated set of observations for node h , in the calculation of the entropy because we consider that a compact representation of the full observations space \mathbf{Z} (the true observation set for the problem) may not be available. Therefore, in order to build $\tilde{\mathbf{Z}}_h$ as a reliable estimation of \mathbf{Z} , we collect all the observations found during *all* traversals within the tree and save them *first* as the *multiset* $\tilde{\mathbf{Z}}_h^m$ –

avoiding losing information, e.g., the frequency of the observations – and then we translate it as the set $\tilde{\mathbf{Z}}_h$ when necessary.

Therefore, each node has its own estimated multiset of observation $\tilde{\mathbf{Z}}_h^m$. Every time we visit a node h , we update the $\tilde{\mathbf{Z}}_h^m$ using the collected observation information, which is saved and back-propagated as $\tilde{\mathbf{Z}}_t^m$. On the other hand, $\tilde{\mathbf{Z}}_t^m$ is the multiset that saves the observations from a single traversal starting from the node at the tree level t to the maximum length of the path D . Therefore, each node h has its own multiset that saves all possible observations to be found by performing a simulation from it. Formally, we can define $\tilde{\mathbf{Z}}_h^m = \bigcup_{i=1}^{\mathcal{N}(h)} \tilde{\mathbf{Z}}_{t,i}^m$, where $\tilde{\mathbf{Z}}_{t,i}^m$ is the back-propagated $\tilde{\mathbf{Z}}_t^m$ at i -th visit to the node, and $\tilde{\mathbf{Z}}_{t,i}^m = \tilde{\mathbf{Z}}_{t+1,i}^m \cup z_t, \forall t = 0, 1, \dots, D$. Note that $\tilde{\mathbf{Z}}_{t,i}^m = \emptyset, \forall t > D$. Let's put it as an example:

Consider $h = h_0$ as the root node, $D = 3$ and a single traversal in the tree. Under the back-propagation perspective, we start our update from the last node visited, related to $\tilde{\mathbf{Z}}_{t+3}^m$. Following our definition, $\tilde{\mathbf{Z}}_{t+3}^m = \{z_{t+3}\}$ and, since it is a leaf node, only z_{t+3} will be included in $\tilde{\mathbf{Z}}_{h_3}^m$ as a new possible observation to be found from node h_3 . Now, in $\tilde{\mathbf{Z}}_{t+2}^m$, we will add the z_{t+2} to our back-propagation multiset $\tilde{\mathbf{Z}}_{t+3}^m$ and, consequently, we will add $\{z_{t+3}, z_{t+2}\}$ to $\tilde{\mathbf{Z}}_{h_2}^m$. We repeat this process until we reach h_0 , where we include all found observations $\tilde{\mathbf{Z}}_t^m = \tilde{\mathbf{Z}}_{t+1}^m \cup z_0 = \{z_3, z_2, z_1, z_0\}$ to the root multiset $\tilde{\mathbf{Z}}_h^m$. Figure 4.1 illustrates this example and our proposed observation back-propagation procedure. Figures 4.1a and 4.1d present a high-level perspective of the process, showing only the observation nodes to facilitate visualisation. Figures 4.1b and 4.1c present a closer perspective of the process as a one-step execution. The red particles are states and the blue ones are observations.

With $\tilde{\mathbf{Z}}_h^m$ in hands, we now can approximate $P_h(z)$ by calculating the frequency of the observation z in $\tilde{\mathbf{Z}}_h^m$, following $P_h(z) \approx \tilde{P}_h(z) = \frac{1}{|\tilde{\mathbf{Z}}_h^m|} \sum_{z' \in \tilde{\mathbf{Z}}_h^m} 1_{\{z'=z\}}$ and define our final entropy function as $\mathcal{H}(h) = - \sum_{z \in \tilde{\mathbf{Z}}_h} \tilde{P}_h(z) \ln(\tilde{P}_h(z))$. Note, however, that we perform the entropy summation across elements using the set of observations $\tilde{\mathbf{Z}}_h$ instead of the multiset $\tilde{\mathbf{Z}}_h^m$, preventing the inclusion of the same element multiple times in the calculation. Besides that, since IB-POMCP plans in an online manner

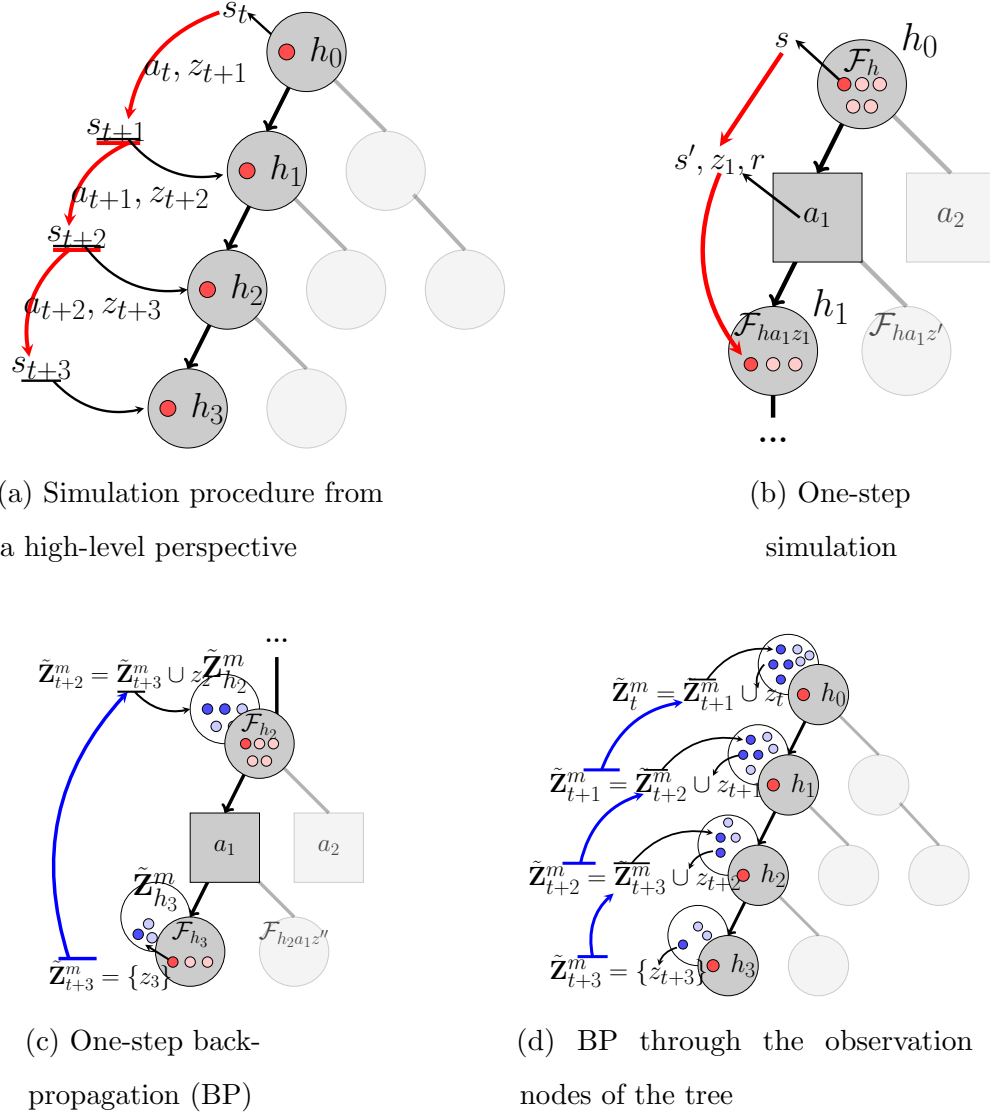


Figure 4.1: Illustration of IB-POMCP’s search process. The red circles represent states stored in the particle filter of a node. The blue circles represent observations stored in the observation set of a node. In the simulation step, we add states to the particle filter of nodes in the path. In the back-propagation step, we add the observations to each node in the path.

and the $\tilde{\mathbf{Z}}_h^m$ is updated at each visit to the node, we propose the estimation of our tree's entropy using the update-equation:

$$\mathcal{H}_{\mathcal{N}(h)}(h) := \mathcal{H}_{\mathcal{N}(h)-1}(h) + \frac{(\mathcal{H}(h) - \mathcal{H}_{\mathcal{N}(h)-1}(h))}{\mathcal{N}(h)} \quad (4.2)$$

where $\mathcal{N}(h)$ is the number of visits to the node h , $\mathcal{H}(h)$ is the above defined entropy calculation, and $\mathcal{H}_i(h), i = 1, 2, \dots, \mathcal{N}(h)$ is the estimated entropy at the visit i . Note that, by following this final definition for the entropy, we can also calculate α (Equation 4.1) and update it in an online fashion.

(c) Entropy Normalisation – Another issue that arises is that, since we may not have access to the true observation distribution in advance, our current approach will result in entropy calculations that are not standardised under the same image function. This problem occurs because the size of the set $\tilde{\mathbf{Z}}$ grows every time an observation never found before is delivered by the environment. As a consequence, the maximum value for the entropy also grows, and nodes that receive a higher number of different observations will calculate higher values for $\mathcal{H}(h)$, which can create bias by including these overestimated values in the decision-making process. Therefore, we propose its normalisation *a posteriori*, following: $\hat{\mathcal{H}}(h) := \frac{\mathcal{H}_{\mathcal{N}(h)}(h)}{\max_{j=1}^{\mathcal{N}(h)} \mathcal{H}_j(h)}$, where $\mathcal{H}_j(h)$ represents the entropy calculated at the j -th visit to the node h . This approach guarantees $\hat{\mathcal{H}}(h) \in [0, 1], \forall h$, i.e., for all nodes. Moreover, $\mathcal{H}(h) = 1$ when $\mathcal{N}(h) = 0$ by definition, since when there is no information, the uncertainty is maximised, hence, the entropy is also maximised.

4.2.5 Simulation

After adjusting our α function, we start the simulation process (step *(iv)*). While expanding the tree, we propose a modified version of UCB1 to guide the tree search process based on the information level of each node, the *I-UCB* function:

$$\text{I-UCB}(h, a, \alpha) := \mathcal{V}(ha) + (1 - \alpha)\sqrt{\ln(\mathcal{N}(h))/\mathcal{N}(ha)} + \alpha\hat{\mathcal{H}}(ha) \quad (4.3)$$

Our proposal intends to guide the expansion and planning process based on entropy updates that occur within the tree, defining what we denote as the *information-guided planning process*. In this type of planning, a trade-off between exploration and exploitation still exists, but now we also bring information value to the discussion. Our main novelty here moves past the traditional idea, where the more we explore, the better we can exploit the rewards, thereby improving the agent’s performance. Instead, we expand the discussion and enhance the algorithm capabilities by introducing an information-gain perspective to the tree-planning process using entropy calculations.

Intuitively, our action selection function considers that **(a)** if the system uncertainty is high (hence, α value is high), we weight the action selection based on the advantage of information gain (which may expand the tree in depth), trying to decrease the entropy value and accumulate knowledge, or; **(b)** if the system uncertainty is low (hence, α value is low), we weight the action selection in the advantage of the exploration gain (which may expand the tree in breadth), trying to increase the entropy value and increase the number of possibilities to reason over. Therefore, I-UCB performs adaptive planning that also depends on the system entropy, besides the upper confidence estimations.

Another direct advantage of I-UCB application instead of UCB1 is that our method rarely fails to deliver an action justified by metrics, that is, it rarely delivers a randomly chosen action as the best one. Usually, this issue arises in problems where rewards are sparsely distributed over a long horizon of actions, which can lead to several ties in the Q-values (e.g., at zero). Proposing a solution that works under these conditions requires **(a)** to increase the reasoning horizon, which usually leads to the expenditure of significantly more computational resources, or **(b)** the capability to handle it within the available reasoning horizon, which is a non-trivial task. IB-POMCP solves this problem by following a non-trivial solution considering **(a)** the inclusion of a novel metric for evaluation – the system entropy, which frequently is

non-zero since the observation distribution is diverse and; **(b)** the adaptive value of alpha, which often promotes an action branch to find (at least) a non-zero value (reward or entropy) or to be frequently visited during the reasoning process.

4.2.6 Best action selection

We decide the best action by calculating $a_{best} = \underset{a \in \mathbf{A}}{\operatorname{argmax}} (1 - \alpha)\mathcal{V}(ha) + \alpha\hat{\mathcal{H}}(ha)$ (step **(v)**). If there is a tie, we break it using the number of visits for each possible node. If it persists, we run a random policy to select the “best” action. Because of the proposed entropy calculation (and in contrast to POMCP), IB-POMCP has the ability to choose actions even without experiencing non-zero rewards while planning.

4.2.7 Algorithm and Practical Enhancement

Algorithm/Pseudo-code – Intending to highlight the difference between POMCP and IB-POMCP, we present the complete pseudocode of our proposal in Algorithm 6. Additionally, IB-POMCP’s code is publicly available on GitHub¹.

Practical Enhancement – Lastely, considering the above discussion on adapting and inserting α inside I-UCB, we consider scaling α value to be within an interval $[q, 1 - q] \subset [0, 1], 0 < q < 0.5$, as a *practical* enhancement. This strategy allows our method to avoid ignoring part of the I-UCB result by multiplying one term by zero. This step will not be considered in the theoretical analysis.

4.3 Theoretical Analysis

In this section, we offer a comprehensive theoretical analysis of our proposed method, IB-POMCP. Therefore, we analyse, theoretically, IB-POMCP’s capability to plan optimally in partially observable finite horizon problems and ϵ -optimally for partially observable infinite horizon problems.

¹IB-POMCP’s GitHub: <https://github.com/lsmcolab/ib-pomcp/>

Algorithm 6 IB-POMCP’s Planning. The starred lines in a light grey background highlight the difference between POMCP and our proposal. We suggest the reader to Silver and Veness (2010) [93] for further details. γ is the historical discount factor, $depth_{max}$ is the maximum depth for the tree, and z_h represents the last observation found (associated with node h).

<p>1: procedure SEARCH(h_τ)</p> <p>2: while Timeout() is False do</p> <p>3: if $\mathcal{F}_\tau = \emptyset$ then $s \sim \mathcal{U}_z$</p> <p>4: else $s \sim \mathcal{F}_\tau$</p> <p>*5: $\alpha := \frac{e \ln(\mathcal{N}(h_\tau))}{\mathcal{N}(h_\tau)} \frac{\sum_{i=1}^{\mathcal{N}(h_\tau)} \mathcal{H}_i(h_\tau)}{\mathcal{N}(h_\tau) \max_{i=1}^{\mathcal{N}(h_\tau)} \mathcal{H}_i(h_\tau)}$</p> <p>*6: Simulate($s, h_\tau, 0, \alpha$)</p> <p>*7: return $\arg \max_{a \in \mathbf{A}} (1 - \alpha) \mathcal{V}(ha) + \alpha \hat{\mathcal{H}}(ha)$</p> <p>1: procedure EXPAND_NODE(h)</p> <p>2: for $a \in \mathbf{A}$ do</p> <p>3: $\mathbf{T}(ha) \leftarrow (ha, \mathcal{V}_{init}(ha), \mathcal{N}_{init}(ha), \emptyset)$</p> <p>1: procedure ROLLOUT(s, h, d, γ)</p> <p>2: if $d < depth_{max}$ then return 0</p> <p>3: $a \sim \pi_{rollout}(h, \cdot)$</p> <p>4: $(s', z, r) \sim \mathcal{G}(s, a)$</p> <p>5: return $r + \gamma \text{Rollout}(s', haz, d + 1)$</p>	<p>1: procedure SIMULATE(s, h, d, α)</p> <p>2: if $d < depth_{max}$ then</p> <p>*3: return 0, $\{z_h\}$</p> <p>4: if $h \notin \mathbf{T}$ then \triangleright If node is not in the tree</p> <p>5: Expand_Node(h)</p> <p>*6: return Rollout(s, h, d), $\{z\}$</p> <p>*7: $a \leftarrow \arg \max_{a \in \mathbf{A}} \text{I-UCB}(h, a, \alpha)$</p> <p>8: $(s', z, r) \sim \mathcal{G}(s, a)$</p> <p>*9: $r', \tilde{\mathbf{Z}}_{t+1}^m \leftarrow \text{Simulate}(s', haz, d + 1, \alpha)$</p> <p>10: $\mathcal{F}_h \leftarrow \mathcal{F}_h \cup \{s\}$</p> <p>11: $\mathcal{N}(h) \leftarrow \mathcal{N}(h) + 1$</p> <p>12: $\mathcal{N}(ha) \leftarrow \mathcal{N}(ha) + 1$</p> <p>13: $\mathcal{V}(ha) \leftarrow \mathcal{V}(ha) + \frac{R - \mathcal{V}(ha)}{\mathcal{N}(ha)}$</p> <p>*14: $\mathcal{H}_i(h) := \mathcal{H}_{i-1}(h) + \frac{(\mathcal{H}_i(h) - \mathcal{H}_{i-1}(h))}{\mathcal{N}(h)}$</p> <p>*15: $R \leftarrow r + \gamma r'$</p> <p>*16: $\tilde{\mathbf{Z}}_h^m \leftarrow \tilde{\mathbf{Z}}_h^m \cup \tilde{\mathbf{Z}}_{t+1}^m$</p> <p>*17: return $R, \tilde{\mathbf{Z}}_{t+1}^m \cup z$</p>
--	---

► Before delving into IB-POCMP’s planning analysis, we need the following definition and assumption for the proof:

Definition 4.1 An estimated $\hat{\mathcal{V}}(s)$ is said to be ϵ -optimal if $|\hat{\mathcal{V}}(s) - \mathcal{V}^*(s)| \leq \epsilon$. From the literature, actions taken from the result of ϵ -optimal value function estimations are known as ϵ -optimal actions.

Assumption 4.1 Given a generic POMDP represented by the 7-tuple:

$$(\mathbf{S}, \mathbf{A}, \mathcal{R}, \mathcal{T}, \mathbf{Z}, \mathcal{Z}, \mathbf{H}),$$

the \mathcal{R} is bounded and there exists a positive maximum reward value $r_{max} \in \mathcal{R}$.

► Under this simple assumption and to establish the IB-POMCP’s planning capabilities, we must examine our action selection procedure, specifically through the application of I-UCB. Consequently, we first analyse how the α coefficient behaves and impacts the IB-POMCP’s search process:

Lemma 4.1 $\alpha(h) \in [0, 1]$ converges to 0 as the number of visits $\mathcal{N}(h)$ approach the infinity. Mathematically, $\lim_{\mathcal{N}(h) \rightarrow \infty} \alpha(h) = 0$.

Proof: Considers our equation for a root node with history h :

$$\alpha(h) = \frac{e \ln(\mathcal{N}(h))}{\mathcal{N}(h)} \frac{\sum_{i=1}^{\mathcal{N}(h)} \mathcal{H}_i(h)}{\mathcal{N}(h) \max_{i=1}^{\mathcal{N}(h)} \mathcal{H}_i(h)}$$

We state that, as the number of visits to a root node $\mathcal{N}(h)$ approaches infinity, $\alpha(h)$ converges to *zero*. This proof is trivial since the second term of our equation is $\frac{\sum_{i=1}^{\mathcal{N}(h)} \mathcal{H}_i(h)}{\mathcal{N}(h) \max_{i=1}^{\mathcal{N}(h)} \mathcal{H}_i(h)} < 1, \forall \mathcal{N}(h) = 1, 2, 3, \dots$, and the first term $\frac{e \ln(\mathcal{N}(h))}{\mathcal{N}(h)} \rightarrow 0, \mathcal{N}(h) \rightarrow \infty$. Therefore:

$$\lim_{\mathcal{N}(h) \rightarrow \infty} \alpha(h) = \lim_{\mathcal{N}(h) \rightarrow \infty} \frac{e \ln(\mathcal{N}(h))}{\mathcal{N}(h)} \frac{\sum_{i=1}^{\mathcal{N}(h)} \mathcal{H}_i(h)}{\mathcal{N}(h) \max_{i=1}^{\mathcal{N}(h)} \mathcal{H}_i(h)} = 0 \quad \blacksquare$$

► Now, we introduce Lemma 2, which will be used as a key result that supports Theorem 1. In other words, we will use Lemma 2's proof to reach a contradiction later in order to prove Theorem 1.

Lemma 4.2 *Assume an action a_i that is taken a finite number of times and a_j that is taken infinitely during IB-POMCP's search for any node h . There $\exists t'' > t'$ such that $I\text{-UCB}_{t''}(ha_i) \geq I\text{-UCB}_{t''}(ha_j)$, where t' is a finite iteration number after which a_i is never taken.*

Proof: Let us assume that there exists an iteration time t that is greater than t' where a_i is never taken again for simulation. Consider the expression $I\text{-UCB}_t(ha_i) - I\text{-UCB}_t(ha_j)$, which is equal to:

$$\begin{aligned} I\text{-UCB}_t(ha_i) - I\text{-UCB}_t(ha_j) &= \left[\mathcal{V}_t(ha_i) - \mathcal{V}_t(ha_j) \right] + \\ & (1 - \alpha_t(h)) \sqrt{\ln(\mathcal{N}_t(h))} \left[\frac{1}{\sqrt{\mathcal{N}_t(ha_i)}} - \frac{1}{\sqrt{\mathcal{N}_t(ha_j)}} \right] + \alpha_t(h) \left[\hat{\mathcal{H}}_t(ha_j) - \hat{\mathcal{H}}_t(ha_i) \right] \end{aligned}$$

Trivially, $\left(\frac{1}{\sqrt{\mathcal{N}_t(ha_i)}} - \frac{1}{\sqrt{\mathcal{N}_t(ha_j)}} \right) \rightarrow \frac{1}{\sqrt{\mathcal{N}_t(ha_i)}}$ as $\mathcal{N}_t(ha_j) \rightarrow \infty$, since $\frac{1}{\sqrt{\mathcal{N}_t(ha_j)}} \rightarrow 0$ and $\mathcal{N}_t(ha_i)$ will be constant under our assumption. Note also that $\frac{1}{\sqrt{\mathcal{N}_t(ha_i)}} - \frac{1}{\sqrt{\mathcal{N}_t(ha_j)}}$ is increasing and will become positive at some time t^* when $\mathcal{N}_{t^*}(ha_j) > \mathcal{N}_{t^*}(ha_i)$. After then (i.e. $t > t^*$), the term $\sqrt{\ln(\mathcal{N}_t(h))} \left(\frac{1}{\sqrt{\mathcal{N}_t(ha_i)}} - \frac{1}{\sqrt{\mathcal{N}_t(ha_j)}} \right)$ will diverge to ∞ as $t \rightarrow \infty$, $\ln(\mathcal{N}_t(h)) \rightarrow \infty$. Further, note that, in a single time step, the maximum difference in rewards is bounded by r_{max} , therefore:

$$\mathcal{V}_t(ha_i) - \mathcal{V}_t(ha_j) \geq - \sum_{k=0}^{\infty} \gamma^k r_{max} = \frac{-r_{max}}{1 - \gamma}$$

This is because $|\mathcal{V}_t(ha_i) - \mathcal{V}_t(ha_j)| \leq \frac{r_{max}}{1 - \gamma}$, as shown in Theorem 3 and this implies that $\frac{-r_{max}}{1 - \gamma} \leq \mathcal{V}_t(ha_i) - \mathcal{V}_t(ha_j) \leq \frac{r_{max}}{1 - \gamma}$. Moreover, we find that $\alpha_t(h)(\hat{\mathcal{H}}_t(ha_j) - \hat{\mathcal{H}}_t(ha_i)) \geq -1$ as $0 \leq \alpha_t(h) \leq 1$ and $0 \leq \hat{\mathcal{H}}_t(h) \leq 1$. Therefore, $\exists t''$ where $I\text{-UCB}_{t''}(ha_i) - I\text{-UCB}_{t''}(ha_j) \geq 0$. ■

Theorem 4.1 For any non-empty history $h \neq \emptyset \in \mathbf{H}$ and $\forall a \in \mathbf{A}$, as $\mathcal{N}(h) \rightarrow \infty$, $\mathcal{N}(ha) \rightarrow \infty$, we have that all states b in $\mathcal{B}(h, b)$ which $P(s = b \mid h) > 0$ will be visited infinitely many times.

Proof: Let us assume that *not* all actions are taken infinitely many times. There must exist *at least* one action $a_i \in \mathbf{A}$ that is taken a finite number of times and, hence, $\exists t'$ after which the action a_i will never be taken. Now, consider any action a_j that is taken infinitely many times as $\mathcal{N}(h) \rightarrow \infty$. In Lemma 2, we show that $\exists t'' > t'$ such that:

$$\text{I-UCB}_{t''}(ha_i) > \text{I-UCB}_{t''}(ha_j)$$

We know that the second term in I-UCB grows faster for a_i than a_j , as seen from the I-UCB function, since $\frac{1}{\mathcal{N}_t(ha_i)}$ is constant for $t > t'$. Consequently, we reach a conclusion that a_i is chosen over a_j again after t' . This further means that for all actions a_k 's, we can find a such a finite time instant after which a_i is preferred over a_k . Therefore, at some point, the action a_i has to be chosen over all others. This is a contradiction to our original claim. Therefore, all action $a \in \mathbf{A}$ are taken infinitely many times and, hence, all states will be visited infinitely many times. ■

► Using the previous results, we can show the convergence for a finite horizon:

Theorem 4.2 IB-POMCP's nodes converge to the optimal value for a fixed horizon D as the number of search iterations goes to infinite (i.e., $\mathcal{N}(h_\tau) \rightarrow \infty$).

Proof: We adapt the proof from the UCT convergence in Shah, Xie, and Xu (2020) [92] and POMCP convergence in Silver and Veness (2010) [93]. We induct over the depth of the tree, starting from the leaf up to the root. From Theorem 1, the leaf nodes will have an unbiased estimation of the expected reward $r_h^a = \mathbb{E}_{b \in \mathcal{B}(h)}(\mathcal{R}(b, a))$, $\forall a \in \mathbf{A}$. Note that we are using r_h^a as the expected reward given a h , which is different from an immediate reward $r_b^a = R(b, a)$ given a state b . Similarly, any state at level $D - 1$ has a simple UCB-like problem since each state is visited infinitely many times and the one-step rewards have an unbiased estimation. In

detail, consider the leaves at level $|h| = D$, as $\mathcal{N}(h_\tau) \rightarrow \infty \Rightarrow \mathcal{N}(h) \rightarrow \infty$, $\alpha \rightarrow 0$, and in the absence of children's entropy $\mathcal{H}(ha)$ (as it is the leaf node and has no children), r_h^a converges unbiasedly $\forall h \in \mathbf{H}, a \in \mathbf{A}$. Consequently, $\mathcal{V}(h) \rightarrow \mathcal{V}^*(h)$ since it is a single decision process, akin to multi-armed bandits. Now, we assume convergence for all levels from the leaves up to $|h'| = |h| + 1$, and as we know $r_h^a = \sum_{s \in \mathbf{S}} P(s_t = b|h)r_b^a$. As we know, the value function satisfies:

$$\mathcal{V}^*(h) = \max_{a \in \mathbf{A}} \left[r_h^a + \gamma \sum P(h_{t+1} = h' | h_t a_t = ha) \mathcal{V}^*(h') \right]$$

From our hypothesis, the $\mathcal{V}^*(h')$ must converge and moreover, through the sampling process, the value function must converge for level $|h|$ [93]. Consequently, by backward induction, the whole tree's values must converge. ■

► We now move on to the infinite horizon case. Consider $\mathcal{V}^*(h_\tau)$ as the optimal value function for the root τ and $\mathcal{V}_D^*(h_\tau)$ as the optimal value function for the root τ for a finite horizon D . We find that:

Theorem 4.3 *Given any $\epsilon > 0$, there is a finite horizon D for the problem, such that $|\mathcal{V}^*(h_\tau) - \mathcal{V}_D^*(h_\tau)| \leq \epsilon$, where h_τ represents the root of the tree.*

Proof: Consider π^* and π_D^* to be the optimal policies for the infinite and finite horizon problem, respectively. Also, assume that ϵ is an arbitrary, positive real number. Now, let the value functions for these problems be $\mathcal{V}^*(h_\tau)$ and $\mathcal{V}_D^*(h_\tau)$. For the root τ , we can find that:

$$|\mathcal{V}^*(h_\tau) - \mathcal{V}_D^*(h_\tau)| \leq \sum_{k=0}^{\infty} \gamma^k r_{max} = \frac{\gamma^D}{1 - \gamma} r_{max} \leq \epsilon$$

since we can calculate $D = \log_\gamma \frac{\epsilon(1-\gamma)}{r_{max}}$, we can also ensure that the relation holds. ■

► Given the previous results, we can prove the following for the convergence in the infinite horizon:

Corollary 4.1 *IB-POMCP converges to ϵ -optimal actions at the root (h_τ) in a γ -discounted infinite horizon scenario.*

Proof: From Theorem 6, we can always find a depth D such that the value function of the finite-horizon problem is arbitrarily close to the infinite-horizon problem. From Theorem 5, we know that we can converge to the optimal value in a finite horizon POMDP. Therefore, since $\alpha \rightarrow 0$ as $\mathcal{N}(h_\tau) \rightarrow \infty$, IB-POMCP will converge to ϵ -optimal actions by choosing D appropriately. ■

4.4 Results

4.4.1 Evaluation Settings

Benchmarks – We define five well-known domains as our benchmarks. The Tiger problem is a well-known standard problem [99]. For the Maze, we based our design on Thomas, Hutin, and Buffet (2020) [99]. For the RockSample problem, we designed our own scenarios but based the implementation on Thomas, Hutin, and Buffet (2020) [99]. For Tag and LaserTag, we used the scenarios proposed by Ye et al. (2017) [110]. For Foraging [5], we proposed our own configurations for each scenario.

- **The Tiger (T0)** problem is a classical benchmark where an agent *must choose between two doors: one hides a treasure; the other, a tiger*. The objective is to find out which door hides the treasure. The agent can decide which door to open or wait and listen to the tiger (with a certain probability of mishearing it behind one of the doors) before making the final decision.

The agent has a 15% probability of mishearing the tiger. A maximum of 20 actions are allowed per experiment. Choosing the right door gives the agent a reward of +0.1, the wrong door -1 , and the listening action penalises the agent with a -0.01 reward. Thus, each agent could try to listen to the tiger 19 times before making the final decision to open either the left or right door.

• **The Maze (M0-M3)** environment [99] is an *Active Self-Localisation Problem* where an agent navigates through a toroidal grid and *tries to localise its own position* by collecting information from the environment. The reward is based on the entropy of the current belief. There is a 15% chance to miss observe the colour of the cell. The belief follows a Bayesian Update process. Figure 4.2 illustrates the scenarios.

(M0) Maze Cross: This is a 5×5 scenario in which black cells are positioned to ease the agent’s localisation problem, resembling a cross in the toroidal world. Figure 4.2a illustrates the scenario’s configuration.

(M1) Maze Holes: This is an 8×8 scenario, which requires the agent to reason one step in advance to search for the missing black cell in this regular configuration. Figure 4.2b illustrates the scenario.

(M2) Maze Dots: This is a 6×6 scenario, identical to Maze Hole except that black cells are in fewer number and separated by several white cells. Figure 4.2c illustrates the scenario.

(M3) Maze Grid-X: This is a 3×3 scenario presenting a pattern of triangles in the grid for the self-localisation: an upper white triangle and a bottom black triangle. Figure 4.2c illustrates the scenario.

• **The RockSample (R0-R3)** problem [99] considers a rover exploring an unknown planet. The rover’s objective is to earn rewards by both sampling rocks in the environment and leaving the planet with the samples. While the positions of the rover and the rocks are known, not all rocks hold scientific value, which are referred to as “good” rocks. Given the cost associated with rock sampling, the rover is equipped with a noisy long-range sensor, enabling it to assess a rock’s potential scientific value before deciding whether to approach and sample it. The environment is a grid map of size $N \times N$ with k rocks. The POMDP model for *RockSample*(N, k) follows: The state space is a combination of $k + 1$ features,

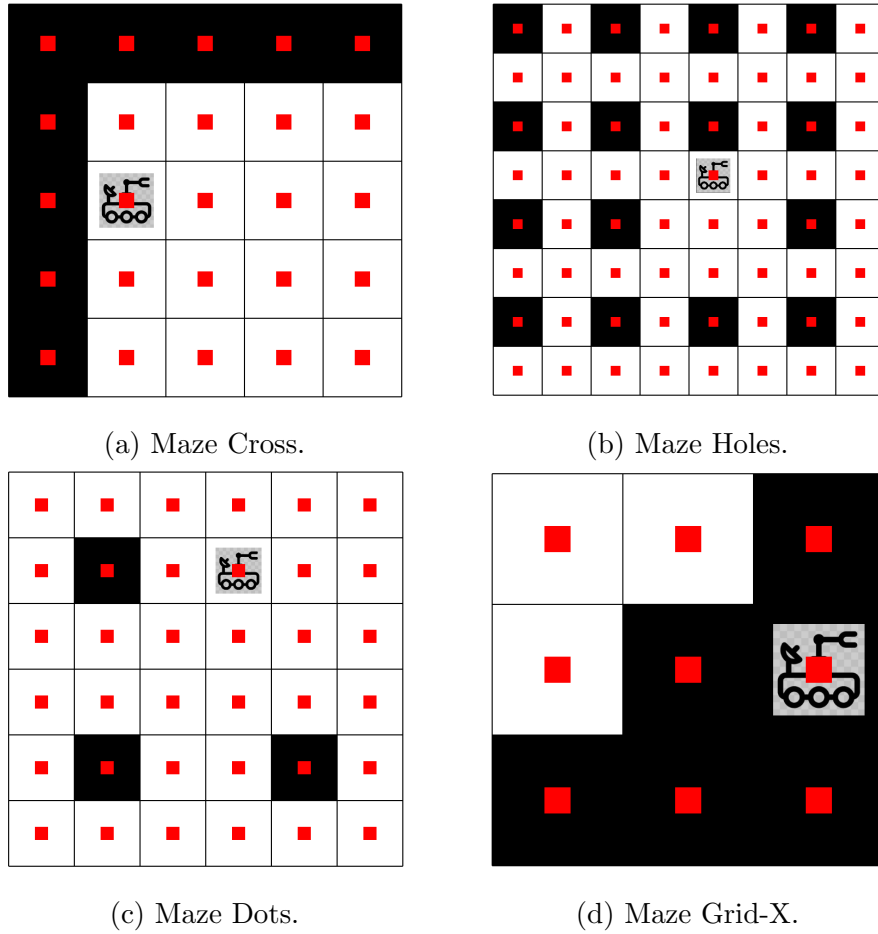


Figure 4.2: Maze environment scenario's configuration.

where *Positions* represent the rover's location $\{(1, 1), (1, 2), \dots, (N, N)\}$, and k binary features, $RockType_i$, indicating whether each rock is "Good" or "Bad". There is a terminal state (portal) at the top-right corner of the map. The rover has a choice of $k + 5$ actions: $\{North, South, East, West, Sample, Check_1, \dots, Check_k\}$. The first four are deterministic single-step motion actions. The *Sample* action involves sampling the rock at the rover's current location. If the rock is determined to be "Good", the rover receives a reward of +1, and the rock transitions to "Bad", signifying no further benefit from sampling. If the rock is "Bad", a penalty of -1 is delivered to the agent. Moving into the exit portal yields a reward of +0.0001. All other actions have no associated cost or reward. Each $Check_i$ action employs the rover's long-range sensor to observe $Rock_i$, yielding a noisy observation of either "Good" or

“Bad”. The noise in the long-range sensor reading is influenced by the efficiency η , which decreases exponentially as a function of Euclidean distance from the target, following $\eta = \exp(-0.2 \text{ EuclideanDistance}(\text{rover}, \text{Rock}_i))$. Initially, every rock is assumed to have an equal probability of being “Good” or “Bad”. Figure 4.3 shows the four scenario configurations used.

- (R0) RockSample22:** This scenario is set on a 5x5 grid, featuring 2 good rocks and 2 bad rocks. The rover initiates its mission from the middle of these rocks and must locate each specific good rock among the bad rocks. Figure 4.3a illustrates the scenario’s configuration.
- (R1) RockSample40:** In this 5x5 scenario, there are 4 good rocks initially placed in the environment, with no bad rocks present. In contrast to RockSample22, the rover’s goal here is to collect more rewards as there are more good rocks available. The rover starts its journey from the centre of these rocks. Figure 4.3b illustrates the scenario’s configuration.
- (R2) RockSample44:** This 10x10 scenario features 4 good rocks and 4 bad rocks. The rover’s starting point is the bottom-left corner of the map. Similar to RockSample22 but on a larger scale, the rover’s challenge is to locate each specific good rock among the bad rocks to optimize its reward. Figure 4.3c illustrates the scenario’s configuration.
- (R3) RockSample17:** In this 10x10 scenario, there’s only 1 good rock and 7 bad rocks, and the rover begins its journey in the bottom-left corner of the map. Unlike RockSample44, the expectation here is for the rover to collect fewer rewards. However, if the rover finds the only good rock available, it can significantly improve its performance in terms of reward collection. Figure 4.3c illustrates the scenario’s configuration.

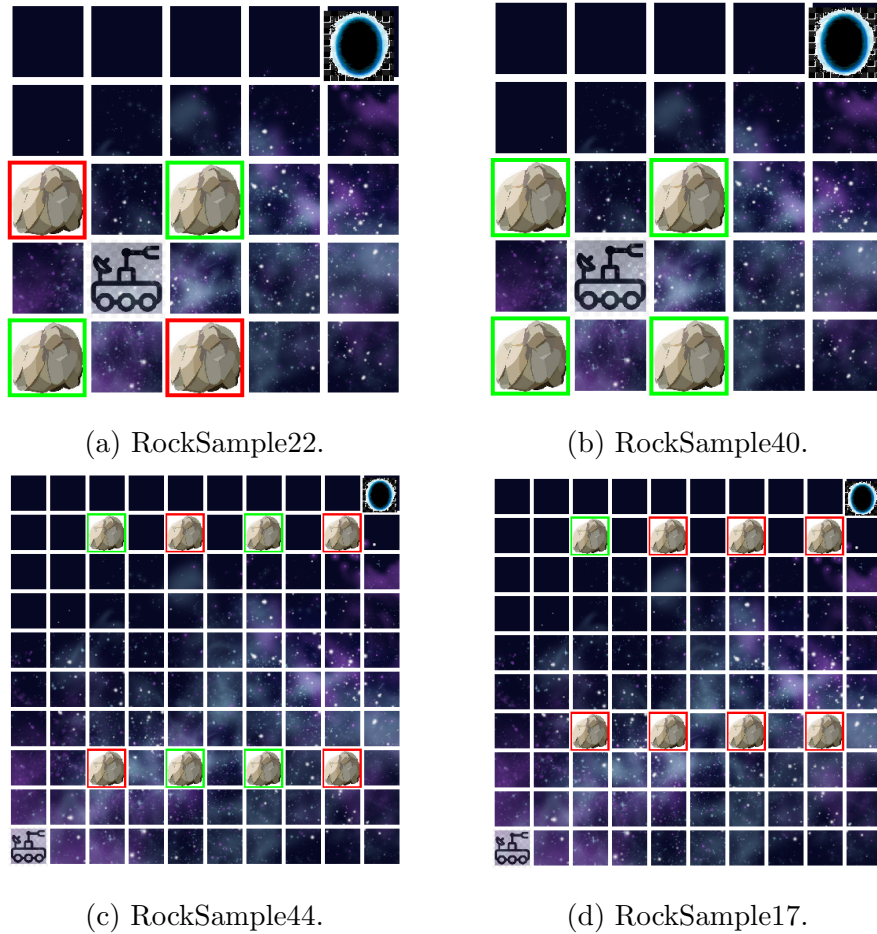


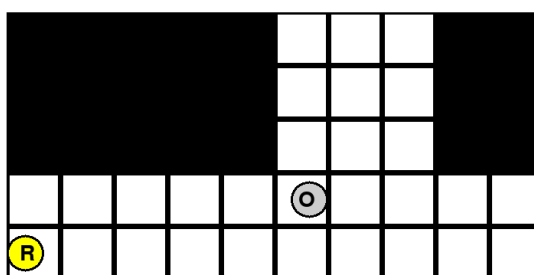
Figure 4.3: RockSample environment scenario's configuration.

- **The Tag/LaserTag (LT0-LT1)** [110], where *an agent is trying to find and tag a target opponent agent that intentionally moves away.*

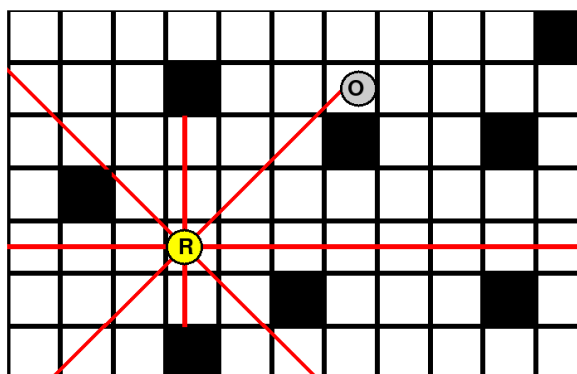
In the Tag scenario (Figure 4.4a), the agent's primary objective is to locate and tag a target that actively moves away. Both the agent and the target navigate within a grid featuring 29 possible positions. While the agent is aware of its own location, it can only observe the target's position when they occupy the same spot. The agent has the option to either stay in its current position or move to any of the four adjacent positions, incurring a cost of -0.1 for each move. Additionally, it can opt to execute the tag action, wherein it receives a reward of $+1$ for a successful tag but receives a penalty of -1 if it fails to tag the target.

On the other hand, the LaserTag (Figure 4.4b) is an augmented version of Tag, where an agent is acting in a 7x11 grid world containing randomly placed obstacles. The agent's behaviour and that of its opponent align with the rules established in Tag. However, in LaserTag, there's a key distinction: the agent possesses prior knowledge of its initial location. Additionally, the agent is equipped with a laser system capable of providing distance estimations (in cells' units) in 8 directions. The laser readings are generated from a normal distribution centred around the agent's true distance from the nearest obstacle in each direction, with a standard deviation of 2.5 units. These readings are then discretised into whole units, resulting in an observation comprising a set of 8 integers.

Figure 4.4 illustrates the scenarios.



(a) Tag.



(b) LaserTag.

Figure 4.4: Tag and LaserTag scenarios' configuration.

• **The Foraging (F0-F4)** problem is a common problem used for evaluating online planning algorithms [5, 15]. This domain presents *an agent that must collect boxes displaced in a rectangular grid-world*. The problem is defined over partial observability and the agent does not know how the tasks are distributed. The problem *ends when the agent collects all boxes*. A reward of +1 is delivered to the agent every time a box is collected. The radius and the angle of vision (integer numbers) for the agents are, respectively, 20% of the diagonal dimension in the environment in cells' unit and 90° – for example, if we have a 10×10 environment, so our vision radius will be $radius = 0.2\sqrt{x_{dim}^2 + y_{dim}^2} = 0.2\sqrt{100 + 100} = 0.2\sqrt{200} = 2$. Additionally, agents have memory, i.e., after seeing (in the real world) a state or position, they will sample the next possible world configuration considering this information. Moreover, obstacles block the agent's vision. Figure 4.5 illustrates the scenarios.

(F0) The Corridor – This is a simple $(20, 2)$ scenario, where the agent starts in the left part of a corridor with a box at its side and another at the end of the corridor. The idea here is to test the agent's capability to find isolated reward spots. Figure 4.5a illustrates the scenario.

(F1) U-Shaped – A complex version of The Corridor, the U-shaped is a 15×15 scenario, where the agent starts at the tip of a U-shaped corridor with a box at the beginning, one in the middle and another at the end. The idea here is to test the agent's capability to perform simple planning, but facing sparse rewards collection. Figure 4.5b illustrates the scenario.

(F2) U-Obstacles – A complex 20×10 scenario, where the agent needs to collect the boxes distributed in a room, but U-shaped walls block its vision. This scenario is commonly used to study local minima issues that emerge in robotic navigation problems [69]. The idea here is to test the agent's capability to perform planning with sparse reward collections, and the vision is blocked by obstacles. Figure 4.5c illustrates the scenario.

(F3) The Warehouse – Our largest in terms of dimensions, 20×20 . The agent’s objective is to locate specific reward spots (groups of boxes). These reward clusters are situated at considerable distances from one another. Consequently, the agent cannot observe multiple reward clusters from a single position, making planning more challenging. Figure 4.5d illustrates the scenario.

(F4) The Office – Our most complex 15×10 scenario, where the agent needs to collect the boxes distributed in an office. The idea here is to test the agent’s capability to perform complex planning when tasks are distributed in different rooms and its vision is blocked by walls. The agent needs to enter each room to finish the problem. Figure 4.5e illustrates the scenario.

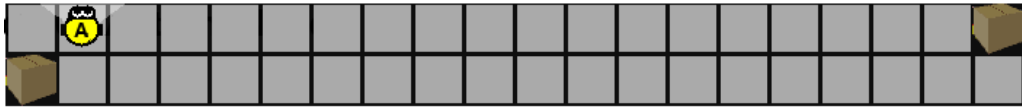
All environments were implemented using *AdLeap-MAS*, our developed simulator (see Appendix A). Each run was performed in a single node of a high-performance cluster containing 16 cores of Intel Ivy Bridge processors and 64 GB RAM.

Baselines – In this thesis, we compare IB-POMCP against 3 relevant methods from the state-of-the art:

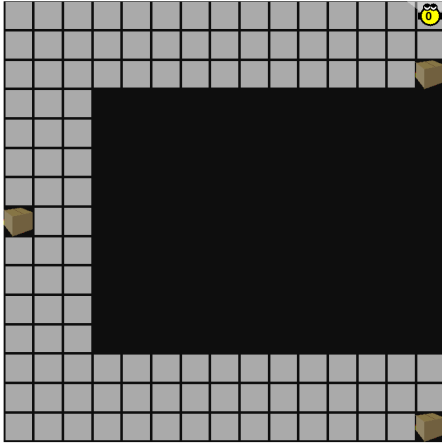
- (i) **POMCP** proposed by [93], since it is a relevant state-of-art proposal and represents the basis of this work;
- (ii) ρ -**POMCP** proposed by [99], representing our main competitor in terms of using information theory to perform the online planning procedure, and;
- (iii) **TB ρ -POMCP** [99], as a faster alternative to ρ -POMCP that employs an explicit POMDP model.

Metrics – Two different evaluation metrics were used for the analysis:

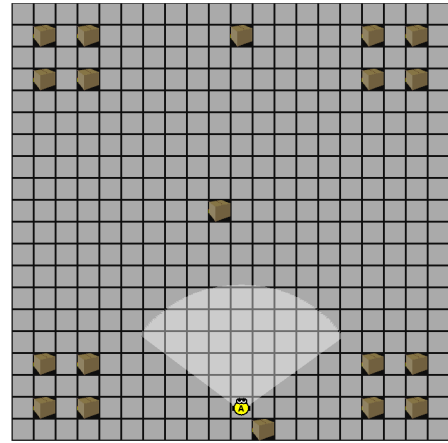
- (i) **the average reward (\mathbf{R})** across the experiment’s mean reward, and;
- (ii) **the average planning time (\mathbf{t})** spent by the agent to plan the next actions.



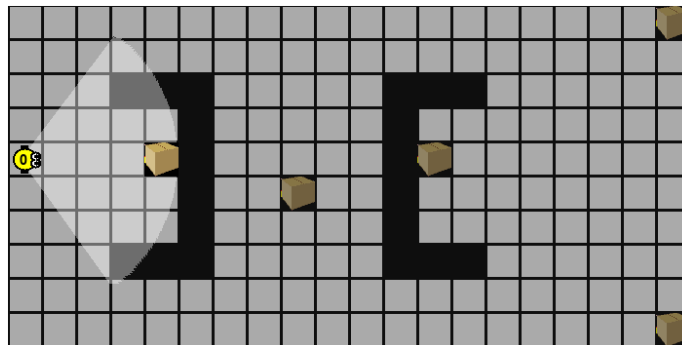
(a) The Corridor.



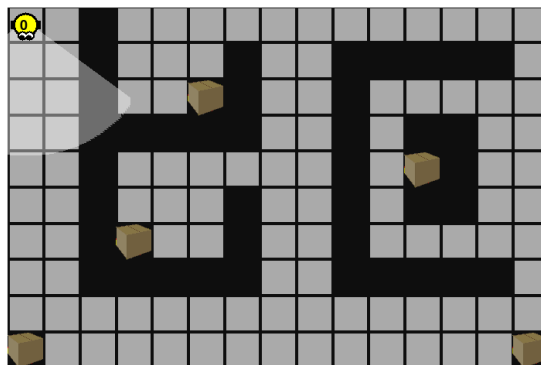
(b) U-shaped.



(d) The Warehouse.



(c) U-Obstacles.



(e) The Office.

Figure 4.5: Foraging environment scenario's configuration.

Mean results were calculated across *50 executions*. Every experiment ran independently; thus, no knowledge was carried from one execution to another. The calculated errors ($\pm Err$) represent the 95% confidence interval of a two-sample t-test. In other words, we label the result as “significant” if it is statistically significant considering $\rho \leq 0.05$ unless otherwise stated. Note that our results and baselines are separated into two categories:

- (i) **Quick Planning**: grouping methods that perform the decision-making process within a reasonable time window, and;
- (ii) **Long Planning**: presenting the ρ -POMCP’s results (without time constraints). We separated them to make their understanding easier.

Hyperparameters – We used a single hyperparameter set for all Monte-Carlo Tree Search-based methods:

- Historical weight/Discount factor: $\gamma = 0.95$
- Maximum depth for the tree: 20
- Maximum number of simulations: 250 (per search)

The value for γ was determined based on the literature [93, 111, 99]. The maximum depth and the maximum number of simulations were selected with the intention of constraining the problem’s execution by limiting the reasoning horizon.

For ρ -POMCP and TB ρ -POMCP specific hyperparameters, based on the best setting found in Thomas, Hutin, and Buffet (2020) [99], we consider:

- Small bag size: $|\mathfrak{B}| = 10$
- TB ρ -POMCP runs within the same time window of IB-POMCP’s average planning time.

Finally, we applied $q = 0.2$ for the IB-POMCP’s experiments, hence $\alpha \in [0.2, 0.8]$.

4.4.2 Benchmarks study

All results are presented in Table 4.1 and 4.2, where Table 4.1 presents the summarised results for the average reward and Table 4.2 for the average planning time.

In the text, we highlight the pros and cons of our method, analysing its limitations while discussing the outcome for each baseline in each benchmark study.

Table 4.1: Average reward result for the baselines in benchmark problems. The bold highlighted values in the Quick Planning category highlight the best result for the respective problem and metric with statistical significance across all Quick Planning baselines. The rewards order are $\times 10^{-2}$, except for the Foraging scenarios (F0-4).

Problem	Quick Planning			Long Planning
	POMCP	TB ρ -POMCP	IB-POMCP	ρ -POMCP
	$R \pm Err$	$R \pm Err$	$R \pm Err$	$R \pm Err$
Tiger (T0)	-4.25 ± 0.80	-0.16 ± 0.03	-0.52 ± 0.15	-0.21 ± 0.03
MazeCross (M0)	1.11 ± 0.08	1.12 ± 0.07	1.23 ± 0.04	1.19 ± 0.07
MazeHoles (M1)	1.56 ± 0.50	1.32 ± 0.46	3.80 ± 0.86	2.34 ± 0.44
MazeDots (M2)	1.20 ± 0.28	0.79 ± 0.15	2.99 ± 0.85	1.25 ± 0.20
MazeGridX (M3)	2.80 ± 0.60	0.99 ± 0.15	0.96 ± 0.02	1.37 ± 0.30
RockSample22 (R0)	0.6 ± 0.1	0.3 ± 0.1	0.7 ± 0.1	0.3 ± 0.1
RockSample40 (R1)	1.2 ± 0.1	1.1 ± 0.2	1.6 ± 0.1	1.0 ± 0.1
RockSample44 (R2)	0.7 ± 0.1	0.6 ± 0.1	0.9 ± 0.2	0.4 ± 0.2
RockSample17 (R3)	-0.4 ± 0.2	-0.6 ± 0.2	0.0 ± 0.1	-0.9 ± 0.2
Tag (LT0)	-4.7 ± 0.9	-6.5 ± 0.9	-3.6 ± 0.7	-6.5 ± 1.0
LaserTag (LT1)	-9.0 ± 0.6	-8.6 ± 0.9	-9.2 ± 0.6	-9.6 ± 5.1
TheCorridor (F0)	4.29 ± 0.51	4.36 ± 0.38	6.89 ± 0.28	5.15 ± 0.18
U-shaped (F1)	0.70 ± 0.34	0.53 ± 0.03	5.10 ± 0.14	5.30 ± 0.31
U-obstacles (F2)	1.17 ± 0.34	1.67 ± 0.23	5.50 ± 0.34	3.75 ± 0.45
Warehouse (F3)	5.20 ± 0.45	4.42 ± 1.03	10.73 ± 0.41	8.99 ± 0.96
TheOffice (F4)	0.43 ± 0.21	0.72 ± 0.14	2.34 ± 0.27	1.61 ± 0.17

Table 4.2: Average time result for the baselines in benchmark problems. The bold highlighted values in the Quick Planning category highlight the best result for the respective problem and metric with statistical significance across all Quick Planning baselines. The time is expressed in seconds.

Problem	Quick Planning			Long Planning
	POMCP	TB ρ -POMCP	IB-POMCP	ρ -POMCP
	$t \pm Err$ (sec)	$t \pm Err$ (sec)	$t \pm Err$ (sec)	$t \pm Err$ (sec)
Tiger (T0)	<u>0.09 \pm 0.01</u>	0.20 \pm 0.00	0.11 \pm 0.01	1.45 \pm 0.04
MazeCross (M0)	2.47 \pm 0.02	3.00 \pm 0.00	<u>2.43 \pm 0.01</u>	28.16 \pm 0.68
MazeHoles (M1)	3.72 \pm 0.03	4.00 \pm 0.00	<u>3.60 \pm 0.04</u>	48.41 \pm 2.82
MazeDots (M2)	2.59 \pm 0.02	3.00 \pm 0.00	<u>2.53 \pm 0.02</u>	36.71 \pm 1.39
MazeGridX (M3)	1.86 \pm 0.02	2.00 \pm 0.00	<u>1.82 \pm 0.01</u>	19.79 \pm 0.32
RockSample22 (R0)	<u>2.08 \pm 0.03</u>	5.00 \pm 0.00	2.33 \pm 0.04	2.70 \pm 0.43
RockSample40 (R1)	<u>2.08 \pm 0.03</u>	5.00 \pm 0.00	2.40 \pm 0.04	2.77 \pm 0.32
RockSample44 (R2)	<u>4.67 \pm 0.09</u>	5.00 \pm 0.00	5.10 \pm 0.06	23.18 \pm 3.10
RockSample17 (R3)	<u>4.72 \pm 0.09</u>	5.00 \pm 0.00	5.11 \pm 0.06	25.49 \pm 3.28
Tag (LT0)	1.22 \pm 0.20	3.00 \pm 0.00	<u>0.86 \pm 0.14</u>	11.23 \pm 1.52
LaserTag (LT1)	4.72 \pm 0.09	3.00 \pm 0.00	5.11 \pm 0.06	19.41 \pm 1.02
TheCorridor (F0)	0.96 \pm 0.06	1.00 \pm 0.00	0.87 \pm 0.10	8.18 \pm 0.71
U-shaped (F1)	3.67 \pm 0.02	3.00 \pm 0.00	<u>2.99 \pm 0.19</u>	37.36 \pm 1.29
U-obstacles (F2)	1.90 \pm 0.03	2.00 \pm 0.00	1.88 \pm 0.08	15.44 \pm 0.37
Warehouse (F3)	3.02 \pm 0.11	3.00 \pm 0.00	2.91 \pm 0.25	20.40 \pm 0.67
TheOffice (F4)	1.90 \pm 0.03	2.00 \pm 0.00	1.88 \pm 0.08	15.44 \pm 0.37

In the *Tiger domain*, TB ρ -POMCP presents the best average reward among the Quick Planning methods ($\rho < 0.01$). IB-POMCP still significantly outperforms POMCP in terms of reward collection ($\rho < 0.01$). For this specific problem, since the action “listen” generates variation in the entropy, IB-POMCP keeps performing it repeatedly until other action value outcomes the “listen” value in the best action selection procedure (Section 4.2, step (v)), a circumstance which leads our method

to reduce its average reward collection. Hence, when facing problems where seeking spots of high uncertainty produces small penalties, IB-POMCP may collect penalties until it significantly decreases the uncertainty and chooses another path.

In *the Maze domain*, we observe that IB-POMCP presents a significantly better average reward in 3 out of 4 proposed scenarios ($\rho \leq 0.015$), except for M3, for which POMCP presents a better result. Note that reducing uncertainty leads to increasing reward; i.e. the faster an agent can access areas with high uncertainty, the higher its received reward. IB-POMCP’s results match our expectations (given the developed rationale throughout our methodology) since we build it to, besides tracking the rewards available in the scenario, often seek paths that lead to spots with high entropy in order to decrease uncertainty, what increases reward in this scenario. In terms of time, we present significantly faster reasoning in all the Maze’s scenarios ($\rho < 0.01$). Investigating why IB-POMCP runs faster than POMCP, we found that our information-guided planning leads the algorithm to perform more transitions during the rollout phase than while performing the simulation inside our actual search tree, with a rate $\frac{\#rollout}{\#simulation} = 1.61$, whereas POMCP presents a rate of 1.28. Because rollout transitions run faster than simulation transitions inside the tree, we can save time by performing them frequently.

In *the Rock Sample problem*, we can see that IB-POMCP shows a significant improvement in terms of reward collection ($\rho \leq 0.02$) in 3 out of 4 scenarios – except for the simplest scenario RockSample22 (R0), which presents a p-value of $\rho \leq 0.11$. In terms of reasoning time, POMCP is slightly faster than all Quick Planning methods in 3 out of 4 scenarios ($\rho \leq 0.01$), except for RockSample17 (R3).

In *Tag*, IB-POMCP presents significant improvement in terms of reward collection and reasoning time against all baselines ($\rho \leq 0.04$). As in the Maze, we believe that IB-POMCP is faster because it simulates transitions in rollouts more often (in a ratio of $\frac{\#rollout}{\#simulation} = 1.7$ against 1.61 for POMCP). In *LaserTag*, we have a tie between all methods, with no statistically significant difference spotted across metrics ($\rho \geq 0.32$ for reward collection and $\rho \geq 0.06$ for reasoning time).

Finally, in the *Foraging domain*, our proposed method significantly outperformed all the baselines ($\rho < 0.01$). The Foraging problem represents our most complex scenario and empirically shows how IB-POMCP can overcome the necessity of adjusting the reasoning horizon to perform planning in settings that deliver rewards sparsely. The only reward available in these scenarios comes from the collection of tasks. Consequently, while attempting to solve a task, the number of actions that the agent needs to plan and execute in sequence may approach or exceed the reasoning horizon size. In this case, the probability of experiencing this reward is low, and algorithms that only follow the reward in planning will rarely obtain it in their simulations; hence, they fail to plan effectively.

Overall, we experimentally demonstrated that our novel proposed method can significantly improve the performance of our agent and its planning capabilities without penalising the reasoning time. We also demonstrated that our proposal can improve the decision-making process by using only the information generated during the tree search process and, in contrast to ρ -POMCP, without relying on the explicit representation of latent functions (e.g., the observation function).

4.4.3 Ablation study

To evaluate the impact of our proposed modifications and enhancements, we performed an ablation study over our method. We consider 2 different variations of IB-POMCP that partially implement its key points in this experiment:

- (a) **Information-guided Particle Reinvigoration POMCP (IPR-POMCP)**, which implements the modifications to the particle filter reinvigoration process (explained in Section 4.2.2 and 4.2.3, steps (i) and (ii), respectively); and
- (b) **Information-guided UCB POMCP (I-UCB POMCP)**, which implements the proposed modifications to the search and simulation process (explained in Section 4.2.4 and 4.2.5, steps (iii) and (iv), respectively).

We consider 4 different scenarios to run our experiments, which are Tiger (T0), MazeDots (M2), U-obstacles (F2) and TheOffice (F4). The results are depicted in Figure 4.6. Both additional methods proposed for the study are available in our GitHub² repository together with the complete code of this work.

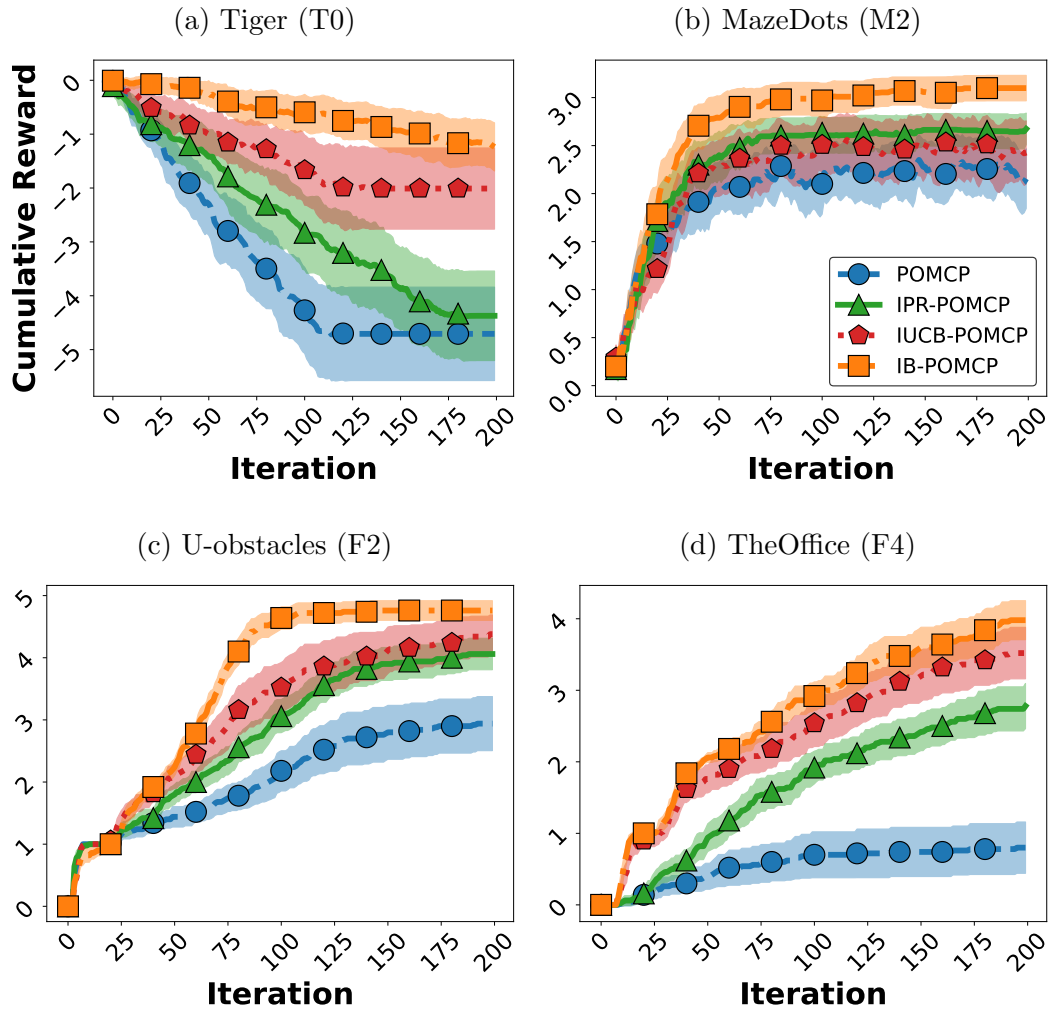


Figure 4.6: Ablation study of IB-POMCP in four different scenarios.

By analysing the graphs, it is clear that the I-UCB and implementation of an information-guided particle reinvigoration process for the Tiger, Maze, and Foraging problems directly enhance the reasoning capabilities of our planning method, which is translated here in terms of reward collection. For the Tiger and Foraging problems,

²IB-POMCP’s GitHub page: <https://github.com/lsmcolab/ib-pomcp/>

diversifying the set of particles (I-UCB POMCP) alone presents improvements for POMCP; however, it has less impact than including entropy in the search process (IPR-POMCP). On the other hand, in the Maze environment, both approaches present similar improvements for the POMCP algorithm. However, when combined, they significantly improved upon the baseline results. Our intuition behind these results is that the IPR-POMCP proposal is responsible for delivering a better set of particles and a better initial estimation of the current state to the agent before simulating actions, whereas I-UCB continually guides and affects the planning process from the beginning of the simulations until the decision of the best action.

4.5 Further Details and Discussion

We save some space here with the intention of assuring the reader’s understanding of the novelty that lies on our particle filter update proposal. Our method employs a dynamic threshold derived from the dynamic value of $\tilde{P}(z|ha)$ (or specifically, $1 - \tilde{P}(z|ha)$ that determines the ratio of uniform samples in the reinvigorated particle filter). Unlike some traditional approaches which use a fixed value for that threshold, or some other state-of-the-art contributions, which mostly rely on additional knowledge and expensive calculations [55, 40, 48], IB-POMCP does not need an explicit true distribution of observations, transition function or training data in this process by using our proposed $\tilde{P}(z|h_{\tau}a)$ probability.

It is important to note that the action selection process within the planning phase has a direct impact on the $\tilde{P}(z|h_{\tau}a)$ value. Therefore, the strategy used to choose actions for simulation during planning will affect the particle filter, since it affects the dynamic threshold $\tilde{P}(z|h_{\tau}a)$. Hence, note that the I-UCB equation used in the search tree also has an impact on belief tracking.

Our suggested algorithm to investigate the impact of this proposal (Ablation study in Section 4.4.3), IPR-POMCP, employs our dynamic threshold and shows significant improvement to the conventional POMCP, which follows a standard

particle reinvigoration paradigm (with a fixed parameter for reinvigoration). Hence, just using the dynamic threshold $\tilde{P}(z|h_\tau a)$ improves results. Furthermore, the ablation study indicates that combining the dynamic threshold with the I-UCB equation yields even more favourable results and that using solely the I-UCB equation without the dynamic threshold (IUCB-POMCP baseline) results in worse outcomes than the full IB-POMCP approach. Hence, both the dynamic threshold and the I-UCB equation are important to improve results, and their simultaneous application produces even greater enhancements than when employed individually.

What is perhaps still left open is whether the I-UCB equation leads to better $\tilde{P}(z|h_\tau a)$ values than the UCB equation. Since our approach takes the estimated entropy into consideration for action selection we believe that I-UCB may lead to better threshold values. However, further research is still necessary on this subject to provide a complete theoretical analysis and prove this impact.

For example, in cases where a node possesses high entropy and no reward in the reasoning horizon, the I-UCB equation could prioritise visiting it more often than the UCB equation would, increasing the $\tilde{P}(z|h_\tau a)$ value for that particular node (note that it will not imply in worse probability estimation because the entropy is based on the estimation of observations and the particle filter gives the probabilities of states). The high entropy indicates that the node accommodates a more diverse set of particles in the particle filter and hence sampling particles from a uniform distribution during particle reinvigoration (to increase diversity) becomes less useful. Hence, the implications of the I-UCB equation appear favourable as it leads to fewer particle samples from the uniform distribution in this case (as we use the ratio $1 - \tilde{P}(z|h_\tau a)$). On the other hand, if a node has low entropy, the particle filter may easily get biased to a small set of states. In this case, I-UCB would lead to a higher number of particles sampled from the uniform distribution than the UCB equation in the particle reinvigoration, since it would have a lower $\tilde{P}(z|h_\tau a)$ value in I-UCB. Therefore, we believe that I-UCB should be better than UCB for defining the dynamic threshold $\tilde{P}(z|h_\tau a)$, and hence better in belief tracking.

4.6 Chapter Conclusion

In this chapter, we presented *Information-based POMCP* (*IB-POMCP*), a novel algorithm for planning under uncertainty that is capable of aggregating information entropy into a decision-making algorithm using our modified version of the UCB function, *I-UCB*. We present the theoretical properties for convergence under certain assumptions, which are supported by empirical results collected in five different domains. Overall, we increase the reward collection by up to 10 times in comparison with TB ρ -POMCP in the U-shaped scenario (F1), in addition to reducing the reasoning time by up to 93% compared to ρ -POMCP in the MazeHoles (M1) scenario. We also kindly refer the reader to Chapter 7.2.1 for a more comprehensive discussion about IB-POMCP's contributions for this thesis' purposes.

Chapter 5

Online Estimators for Ad-hoc Task Execution

In this chapter, we present the technical details of OEATE, our proposed type and parameter estimation method, which runs together with an online planning algorithm and intends to improve performance in teamwork context. This work was published as a journal at AAMAS 2022 and as a short paper in AAMAS 2023, titled as “On-line Estimators for Ad-hoc Task Execution: Learning Types and Parameters of Teammates for Effective Teamwork” [26, 28].

5.1 Introduction

Autonomous agents are usually designed to pursue a specific strategy and accomplish a single or set of tasks. Intending to improve their performance, these agents often follow specified coordination and communication protocols to enable the collection of valuable information from the environment components or even from other reliable agents. However, employing these methods is challenging due to environmental and technological constraints. There are circumstances where communication channels are unreliable, and agents cannot fully trust them to send or receive information. Moreover, particular situations require the design of agents from various parties

aiming to solve a problem urgently, but constructing and testing communication and coordination protocols for all different agents can be unfeasible given the time constraints. For example, consider a natural disaster or a hazardous situation where institutions may urgently ship robots from different parts of the world for handling the problem. In these scenarios, avoiding delays and unnecessary funding usage would save lives and mitigate the caused damages.

One possible solution is to offer a centralised mechanism to allocate tasks to each agent in the environment in an efficient manner. However, we may face scenarios where there is no centralised mechanism available to manage the agents' actions. Considering large scale problems, it is even easier to imagine situations where environmental or time constraints also derail this solution. Hence, agents need to decide, autonomously, which task to pursue [17] – defining what we will denominate a decentralised execution scenario. The *decentralised* execution is quite natural in ad-hoc teamwork, as we cannot assume that other agents follow the same centralised controller. Therefore, allowing agents to reason about the surrounding environment and create partnerships with other agents can support the accomplishment of missions that are hard to deal with individually, reducing the necessary time to achieve all tasks and minimising the costs related to the process.

For many relevant domains, these decentralised execution problems can be modelled focusing on the set of tasks that need to be accomplished in a distributed fashion (e.g., victims to be rescued from a hazard, letters to be quickly delivered to different locations, etc). Note this kind of design presents a task-based perspective to solve the problem, where agents must reason about their teammates' targets to improve the coordination, hence the team's performance. In this way, the agents must approximate the teammates' behaviours (or their main features) in order to deliver this improvement while solving the problem.

As our first goal, we will address the problem where agents are supposed to complete several tasks cooperatively in an environment where there is no prior information, reliable communication channel or standard coordination protocol to

support the problem completion. We will denominate this ad-hoc team situation as a *Task-based Ad-hoc Teamwork* problem, a decentralised distributed system where agents decide their tasks autonomously, without previous knowledge of each other, in an environment full of uncertainties.

Instead of developing algorithms that are able to learn any possible policy from scratch, a common approach in the ad-hoc teamwork literature is to consider a set of possible agent types and parameters, thereby reducing the problem of estimating those [4, 5, 12]. This approach is more applicable, as it does not require a large number of observations, thus allows the learning and acting to happen simultaneously in an on-line fashion, i.e., in a single execution. Types could be built based on previous experiences [14, 15] or derived from the domain [3]. Moreover, the introduction of parameters for each type allows more fine-grained models [4]. However, previous works that learn types and parameters in ad-hoc teamwork are not specifically designed for decentralised task execution, missing an opportunity to obtain better performances in these relevant MAS scenarios.

Other lines of work focus on neural network-based models and learn the policies of other agents after thousands (even millions) of observations [48, 83]. These applications, however, would be costly, especially when domains get larger and more complicated. Similarly, I-POMDP based models [30, 36, 44, 50] could be applied for reasoning about the model of other agents, but its application is non-trivial considering larger problems. On the other hand, some approaches in the literature have also tested task-based designs, inferring about agents pursuing tasks to predict their behaviour [31]. Although we share some similarities, they have not yet handled learning types and parameters of agents in ad-hoc teamwork systems where multiple agents may need to cooperate to complete common tasks.

Therefore, as our main contribution, we present *Online Estimators for Ad-hoc Task Execution* (OEATE), a *novel algorithm* for estimating teammates types and parameters in decentralised task execution. Our algorithm is light-weighted, running estimations *from scratch* at every single run, instead of employing pre-trained models,

or carrying knowledge between executions. Under some assumptions, we show theoretically that our algorithm converges to a perfect estimation when the number of tasks to be performed gets larger. Additionally, we run experiments for two collaborative domains: (i) a level-based foraging domain, where agents collaborate to collect “heavy” boxes together, and; (ii) a capture the prey domain, where agents must collaborate to surround preys and capture them. We also tested the performance of our method in full and partial observable scenarios. We show that we can obtain a lower error in parameter and type estimations in comparison with the state-of-the-art, leading to significantly better performance in task execution for some of the studied cases. We also run a range of different scenarios, considering situations where the number of agents, scenario sizes, and the number of items gets larger. Furthermore, we evaluate the impact of increasing the number of possible types. Finally, we run experiments where our ad-hoc agent does not have the true type of the other agents in its pool of possible agent types. In such challenging situations, our parameter estimation outstands the competitors and, our type estimation and performance is similar or better than the state-of-the-art in several cases considering the results’ confidence interval.

5.2 The Task-based Ad-hoc Teamwork Problem

Before delving into OEATE, we introduce our Task-based Ad-hoc Teamwork model implemented in our solution. We extend the discussion presented in the Background (Chapter 2) but now focusing on OEATE’s model and our specific problem.

Ad-hoc teamwork defines domains where agents intend to cooperate with their teammates and coordinate their actions to reach common goals. Moreover, agents in the domains do not have any prior communication or coordination protocols to enable the exchange of information between them, so learning and reasoning about the current context are mandatory to improve the team’s performance as a unit. However, if agents are aware of some potential pre-existing standards for

coordination and communication, they can try to learn their teammates features [15]. As a result of such intelligent coordination in the ad-hoc teams, they can improve their decision-making process and hence, accomplish shared goals more efficiently. This fundamental model can be extended to fit distinct problems and scenarios.

In OEATE, we extend the Ad-hoc Teamwork model to a **Task-based Teamwork Model**, enabling a better representation of our world as presented in previous state-of-the-art works [2, 14, 111]. In this model, one *learning agent* ϕ , acts in the same environment as a set of *non-learning agents* $\omega \in \Omega$, $\phi \notin \Omega$. In the ad-hoc team $\phi \cup \Omega$, the objective of ϕ (as the learning agent) is to maximise the performance (e.g., increase the number of tasks accomplished or decrease the necessary time to finish them all). However, all non-learning agents' models are unknown to ϕ , and there is no communication channel available. Hence, ϕ must estimate and understand their models as time progresses, by observing the scenario. In other words, the learning agent must improve its decision-making process by approximating the teammates' behaviour in an on-line manner and facing a lack of information.

Besides, there is a set of tasks \mathbb{T} which all agents in the team endeavour to accomplish autonomously. A task $\tau \in \mathbb{T}$ may require multiple agents to perform it successfully and multiple time steps to be completed. For instance, in a foraging problem, a heavy item may require two or more robots to be collected, and the robots would need to move towards the task location to accomplish it, taking multiple time steps to move from their initial position.

The learning agent ϕ must minimise the time to accomplish all tasks. Hence, playing this role requires the support of a method that integrates the estimation and the decision-making process while performing and improving the planning. On the other hand, all non-learning agents aim to finish the tasks in the environment autonomously. However, choosing and completing a task τ by any ω is dependent on its internal algorithm and its capabilities. Nonetheless, ω 's algorithm can be one of the potential algorithms defined in the system, which might be learned from previous interactions with other agents [14].

Therefore, following the model of non-Learning agents defined in previous works [4, 111], there is a set of potential algorithms, which compose a set of possible types Θ for all $\omega \in \Omega$. The assumption is that all these algorithms make decisions based on a vector of *parameters*. Hence, the types are all *parameterised*, which affects agents' behaviour and actions. Considering the existence of these types' parameters allows ϕ to use more fine-grained models when handling new unknown agents.

According to these assumptions, each $\omega \in \Omega$ will be represented by a tuple (θ, \mathbf{p}) , where $\theta \in \Theta$ is ω 's type and \mathbf{p} represents its parameters, which is a vector $\mathbf{p} = \langle p_1, p_2, \dots, p_n \rangle$. Also, each element p_i in the vector \mathbf{p} is defined in a fixed range $[p_i^{min}, p_i^{max}]$ [4]. So, the whole parameter space can be represented as $\mathbf{p} \subset \mathbb{R}^n$. These parameters can be the abilities and skills of an agent. For instance, a robot can be quite different depending on its hardware – for a robot, it can be vision radius, the maximum battery level or the maximum velocity. The parameters could also be hyper-parameters of the algorithm itself. Consequently, each $\omega \in \Omega$, based on its type θ and parameters \mathbf{p} , will choose a target task. The process of choosing a new task can happen at any time and any state, depending on the agents' parameters and type. We denominate these decision states as *Choose Target States* $\mathfrak{s} \in \mathbf{S}$.

In the Task-based Ad-hoc Teamwork context, a precise estimation of tasks also depends on estimating the *Choose Target State*. Our method presents a solution to this problem by considering an *information-based* perspective, which does its evaluation by giving different weights to the *information* derived from observations made by the agent ϕ , instead of directly estimating the choose target state.

5.3 The Markovian Extension for Task-based Ad-hoc Teamwork Problems

Before introducing our Markovian model applied to OEATE, we need to understand what is a *Stochastic Bayesian Game* (SBG) and its importance for the ad-hoc teamwork context. Overall, a SGB describes a well-suited solution towards the

representation of ad-hoc teamwork problems that combine the Bayesian games with the concept of stochastic games and provide a descriptive model to the context [14, 72]. In this section, we will define an SBG-based model for our specific setting. We refer the reader to Melo and Sardinha (2016) [72] for a more generic formulation.

Our model consists of a discrete state space \mathbf{S} , a set of players ($\phi \cup \Omega$), a state transition function \mathcal{T} and a type distribution Δ . Each agent $\omega \in \Omega$ has a type $\theta_i \in \Theta$ and a parameter space \mathbf{p} . Each parameter is a vector $\mathbf{p} = \langle p_1, p_2 \dots p_n \rangle$ and each $p_i \in [p_i^{min}, p_i^{max}]$, for all agents. The set $[p_1^{min}, p_1^{max}] \times \dots \times [p_n^{min}, p_n^{max}] = \mathbf{p} \subset \mathbb{R}^n$ is the parameter space for each agent. Each type could have a different parameter space, but we define a single parameter space here for simplicity of notation. Furthermore, we assume that the types of the agents are fixed throughout the process (a pure and static type distribution). Moreover, each player is associated with a set of actions, an individual payoff function and a strategy. Considering that at each time step, agents $\omega_i \in \Omega$ are fixed tuples (θ_i, \mathbf{p}_i) , where $\theta_i \in \Theta$ and $\mathbf{p}_i \in \mathbf{p}$, we extend the SBG model in order to describe the following situation:

Problem: Consider a set of players $\phi \cup \Omega$ that share the same environment. Each player acts according to its type θ_i , parameters \mathbf{p}_i and own strategy π_i . They do not know the others' types or parameters. At each time step t , given the state s^t and a joint action $a^t = (a_\phi^t, a_1^t, a_2^t, \dots, a_{|\Omega|}^t)$, the game transitions accordingly to the transition probability \mathcal{T} and each player receives an *individual payoff* r_i until the end of the game. How can we maximise the *overall payoff* in this situation?

Therefore, by using the SBG model, we can represent our problem and the necessary components in it. However, we consider in this work a fully cooperative problem, under the point of view of agent ϕ . Hence, within the task-based ad-hoc teamwork context, we want to model the problem employing a single-player abstraction under ϕ 's point of view. Using a Markov Decision Process Model (MDP), we can abstract all the environment components as part of the state (including teammates in Ω), as explained in Section 2.2.3. This approach enables the aggregation of individual

rewards from the SBG model into a single global reward and allows us to use single-player Monte Carlo Tree Search techniques, as previous works did [81, 111, 2].

The OEATE’s *MDP* consists of a mathematical framework to model stochastic processes in a discrete time flow, as presented in Section 2.1.1. Although there are multiple agents and perspectives in the team, we will define the model *considering the point of view of an agent ϕ* and apply a *single agent MDP model*, as in previous works [81, 2, 111] that represent other agents as part of the environment.

All ω in Ω are modelled as the *environment*, as their actions indirectly affect the next state and the obtained reward. Therefore, they are abstracted in the transition function. That is, in the *actual* problem, the next state depends on the actions of all agents, however, ϕ is unsure about the non-learning agents next action. For this reason, we consider that given a state s , an agent $\omega \in \Omega$ has a (unknown) probability distribution (pdf) across a set of actions \mathbf{A}_ω , which is given by ω ’s internal algorithm (θ, \mathbf{p}) . This pdf is going to affect the probability of the next state. Therefore, we can say that the uncertainty in the MDP model comes from the randomness of the actions of the ω agents, besides any additional stochasticity of the environment.

This model allows us to employ single-agent on-line planning techniques, like UCT Monte Carlo Tree Search [58]. In the tree search process, the pdf of each agent defines the transition function. At each node transition, ϕ samples ω agents’ actions from their (estimated) pdfs, and that will determine the next state s' for the next node. However, in traditional UCT Monte Carlo Tree Search, the search tree increases exponentially with the number of agents. Hence, we use a history-based version of UCT Monte Carlo Tree Search called *UCT-H*, which employs a more compact representation than the original algorithm, and helps to trace the tree in larger teams in a faster fashion [111]. We refer the reader to our background material (at the end of Section 2.2.1) for more details about UCT-H.

As mentioned earlier, in this task-based ad-hoc team, ϕ attempts to help the team to get the highest possible reward. For this reason, ϕ needs to find the optimal value function, which maximises the expected sum of discounted rewards $E[\sum_{j=0}^{\infty} \gamma^j r_{t+j}]$,

where t is the current time, r_{t+j} is the reward ϕ receives at j steps in the future, $\gamma \in (0, 1]$ is a discount factor. Also, we consider that we obtain the rewards by solving the tasks $\tau \in \mathbb{T}$. That is, we define ϕ 's reward as $\sum r(\tau)$, where $r(\tau)$ is the reward obtained after the task τ completion. Note that the sum of rewards is not only across the tasks completed by ϕ , but all tasks completed by any set of agents in a given state. Furthermore, there might be some tasks in the system that cannot be completed without cooperation between the agents, so the number of required agents for finishing a task τ depends on each specific task and the set of agents that are jointly trying to complete it.

Note that the agents' types and parameters are actually not observable, but in our MDP model that is not directly considered also. Estimated types and parameters are used during on-line planning, creating an estimated transition function. The actual decisions made by the non-learning agents are observable in the real world transitions without any direct information about type and parameters. More details are available in the next section.

5.4 The Estimation Problem

Considering the problem described by the MDP model in Section 5.3 and its details presented in Section 2.1.1, we will describe the general workflow of an estimation process and discuss how we integrated planning and estimation in this work.

- **Estimations process** – Initially, since agent ϕ does not have information about each agent ω 's true type θ^* and true parameters \mathbf{p}^* , it will not know how they may behave at each state, hence, must reason about all possibilities for type and parameters from distribution Δ . So, ϕ must consider, for each $\omega \in \Omega$, an uniform distribution for initialising the probability of having each type $\theta \in \Theta$, as well as randomly initialising each parameter in the parameter vector \mathbf{p} based on their corresponding value ranges. However, given some domain knowledge, it could be sampled from a different distribution both for types and for parameters.

After each estimation iteration, we expect that ϕ will have a better estimation for θ and \mathbf{p} of each non-learning agent in order to improve its decision-making and the team's performance. Hence, ϕ must learn a probability for each type, and for each type, it must present a corresponding estimated parameter vector. In further steps, as agent ϕ observes the behaviour of all $\omega \in \Omega$ and notices their actions and the tasks that they accomplish, it keeps updating all the estimated parameter vectors \mathbf{p} , and the probability of each type $P(\theta)_\omega$, based on the current state. The way these estimations are updated depends on which on-line learning algorithm is employed.

This described process aims to improve the quality of ϕ 's decision-making based on the quality of the result delivered by the estimation method. Therefore, we will perform experiments using three different methods from the literature for type and parameter estimation: Approximate Gradient Ascent (AGA), Approximate Bayesian Update (ABU) [4] and POMCP [93], which are explained in detail in Section 2.4. Moreover, these methods will represent our baselines for comparison against our novel algorithm, denominated *Online Estimators for Ad-hoc Task Execution* (OEATE), for parameter and type estimation in decentralised task execution, which will be described in detail in Section 5.6.

A question that may arise here is whether IB-POMCP could serve as a planning and estimation method, thus acting as a baseline for OEATE. The short answer is yes; IB-POMCP could indeed be utilized as a baseline for OEATE if adapted accordingly. Similar to the adaptation we will discuss for POMCP in Section 5.5, it is feasible to leverage IB-POMCP for estimating types and parameters in this context. However, the mere act of adapting POMCP for estimation within this task-oriented domain already hints at the potential of IB-POMCP, comparing their performance to proper task-based estimation methods. Therefore, while we acknowledge this possibility, we opt to conduct experiments solely using the POMCP adaptation.

• **Planning and Estimations** – The current estimated models of the non-learning agents are used for on-line planning, allowing agent ϕ to estimate its best actions. In this work, we employ UCT-H for agent ϕ 's decision-making. UCT-H is similar

to UCT, but using a history-based compact representation. This modification was shown to be better in ad-hoc teamwork problems [111]. Therefore, as in previous works [4, 111], we sample a type $\theta \in \Theta$ for each non-learning agent from the estimated type probabilities each time we re-visit the root node during the tree search process. We use the newly estimated parameters \mathbf{p} for the corresponding agent and sampled type, which will impact the estimated transition function, as described in our MDP model. Consequently, the higher the quality of the type and parameter estimations, the better will be the result of the tree search process. As a result, agent ϕ makes a decision concerning which action to take.

Note that the actual ω agents may be using different algorithms than the ones available in our set of types Θ . Nonetheless, agent ϕ would still be able to estimate the best type θ and parameters \mathbf{p} to approximate agent ω 's behaviour. Additionally, ω agents may or may not run algorithms that explicitly model the problem as decentralised task execution or over a task-based perspective. However, using the *single-agent MDP*, we only need agent ϕ to be able to model the problem as such.

5.5 POMCP as an Estimation Method

Although in the MDP model agent ϕ has full observation of the environment, it cannot observe the type and parameters of its teammates. Therefore, we can employ POMCP [93], a state-of-the-art on-line planning algorithm for POMDPs [53]. POMCP stores a particle filter at each node of a Monte Carlo Search Tree. In this case, like the environment, apart from the types and parameters of the other agents, is fully observable, the particles are defined as different combinations of the types and parameters for all agents in Ω . I.e., $[(\theta_1, \mathbf{p}_1), (\theta_2, \mathbf{p}_2), \dots, (\theta_n, \mathbf{p}_n)]$, where each (θ, \mathbf{p}) corresponds to one non-learning agent.

In the very first root, when the particles are created, we randomly assign types and parameters for each agent at each particle. Therefore, at every iteration, we sample a particle from the particle filter of the root, and hence change the estimated

type and parameters of the agents. As in the POMCP algorithm, the root gets updated once a real action is taken, and a real observation is received. Therefore, for having a type probability $P(\theta)_\omega$ for a certain agent ω , we calculate the frequency that the type θ is assigned to ω in the current root's particle filter. Additionally, for the parameter estimation, we will consider the average across the particle filter (for each type and agent combination). For further explanations, we recommend revisiting our Background (Section 2.2.2) or the reading of Silver and Veness (2010)'s work [93].

5.6 Online Estimators for Ad-hoc Task Execution

In this section, we introduce our novel algorithm, *Online Estimators for Ad-hoc Task Execution* (OEATE), which helps the ad-hoc agent ϕ to learn the parameters and types of non-learning teammates autonomously. The main idea of the algorithm is to observe each non-learning agent ($\omega \in \Omega$) and record all tasks ($\tau \in \mathbb{T}$) that any one of the agents accomplishes, in order to compare them with the predictions of sets of *estimators*. In OEATE, there are some fundamental concepts applied during the process of estimating parameters and types. Therefore, we introduce the concepts first and, then, explain the algorithm in detail.

5.6.1 OEATE Fundamentals

- **Sets of Estimators** – In OEATE, there are sets of *estimators* \mathbf{E}_ω^θ for each type θ and each agent ω that the agent ϕ reasons about (Figure 5.1). Moreover, each set \mathbf{E}_ω^θ has a fixed number of N *estimators* $e \in \mathbf{E}_\omega^\theta$. Therefore, the total number of sets of *estimators* for all agents are $|\Omega| \times |\Theta|$. Figure 5.1 presents this idea, relating agent, types and *estimators*.

An *estimator* e of \mathbf{E}_ω^θ is a tuple: $\{\mathbf{p}_e, c_e, f_e, \tau_e\}$, where:

- \mathbf{p}_e is the vector of estimated parameters for ω , and each element of the parameter vector is defined in the corresponding element range.

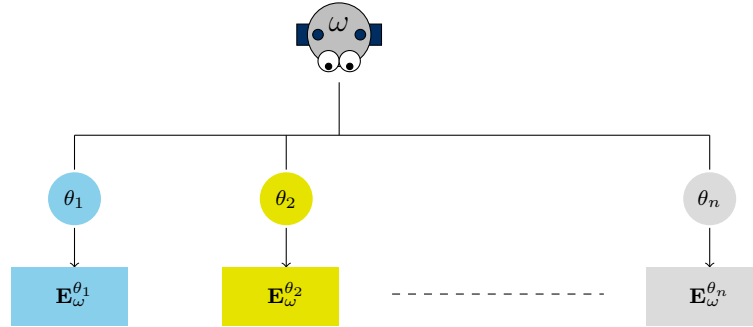


Figure 5.1: Illustration of OEATE’s set of estimators. Note that, for each ω agent there is a set of *estimators* \mathbf{E}_ω^θ for each type.

- c_e holds the success score of each estimator e in predicting tasks.
- f_e holds the failures score of each estimator e in predicting tasks.
- τ_e is the task that ω would try to complete, assuming type θ and parameters \mathbf{p}_e . By having estimated parameters \mathbf{p}_e and type θ , we assume it is easy to predict ω ’s target task at any state.

Note that when we say *it is easy to predict ω ’s target task at any state, given a vector of parameters \mathbf{p}_e and a type θ* , it is due to our assumption that our ad-hoc agent operates under full observability and the templates utilised to simulate teammates in the environment are parameterised. Consequently, we can accurately discern the information other agents possess and predict their target task with ease. However, in scenarios where these characteristics do not hold, this assumption may be unreliable. The success and failure scores (c_e and f_e , respectively) will be further explained in the *Evaluation* step of OEATE presentation.

All *estimators* are initialised in the beginning of the process and evaluated whenever a task is done (by the ω agent alone or cooperatively). The *estimators* that are not being able to make good predictions after some trials are removed and replaced by *estimators* that are created using successful ones, or purely random, in a fashion inspired by GA [52].

• **Bags of successful parameters** – Given the vector of parameters $\mathbf{p}_e = \langle p_1, p_2, \dots, p_n \rangle$, if any *estimator* e succeeds in task prediction, we keep each element of the parameter vector \mathbf{p}_e in bags of successful parameters to use them in the future during new parameter vector creation. Accordingly, there is a bag of parameters \mathbf{B}_ω^θ for each type $\theta \in \Theta$ as there is an estimator set \mathbf{E}_ω^θ for each type. These bags are not erased between iterations, hence, their size may increase at each iteration. There is no limit size for the bags. We will provide more details in Section 5.6.2. Figure 5.2 presents this idea, relating agent, types and *estimators* to the addition of estimators in the bags.

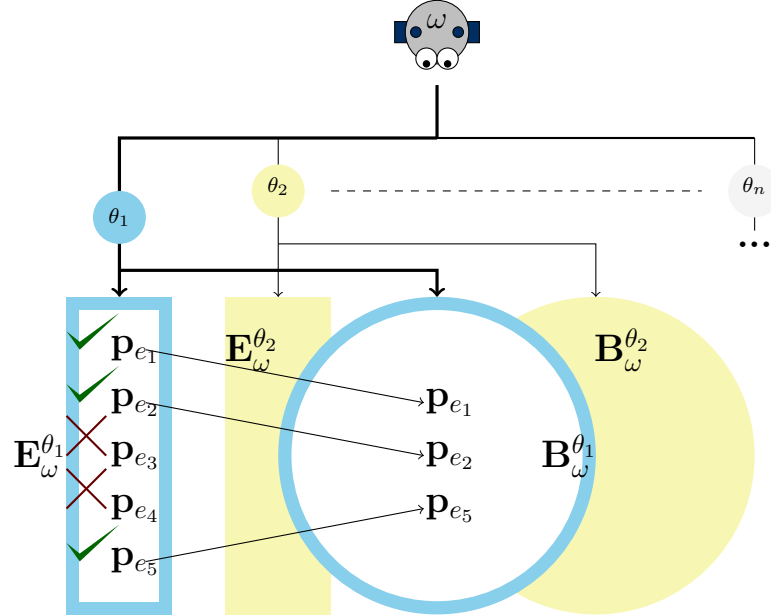


Figure 5.2: Illustration of OEATE’s bag of successful parameters. For each ω agent and each possible type $\theta \in \Theta$, there is a bag of successful estimators. Successful estimators are copied to the bag of estimators of their respective type, in order to later generate new combinations of their elements. The check (green) indicates success in predicting the task and the cross (red) indicates failure.

• **Choose Target State** – In the presented task-based ad-hoc teamwork context, besides estimation of type and parameter for each non-learning agent ($\omega \in \Omega$), ϕ must be able to estimate the *Choose Target State* (\mathbf{s}_e) of each ω . The *Choose Target*

State of an ω agent can be any $s \in S$ or, in other words, a non-learning agent ω can choose a new task $\tau \in \mathbb{T}$ to pursue at any time t or state s . This can happen in many situations, for example, when ω notices that its target does not exist anymore (if it was completed by other agents), it would choose a new target, and the *Choose Target State* would not be the same state as when the last task was done by ω . Hence, a task-based estimation algorithm must be able to identify these moments where a possible task decision happened, to correctly predict the target.

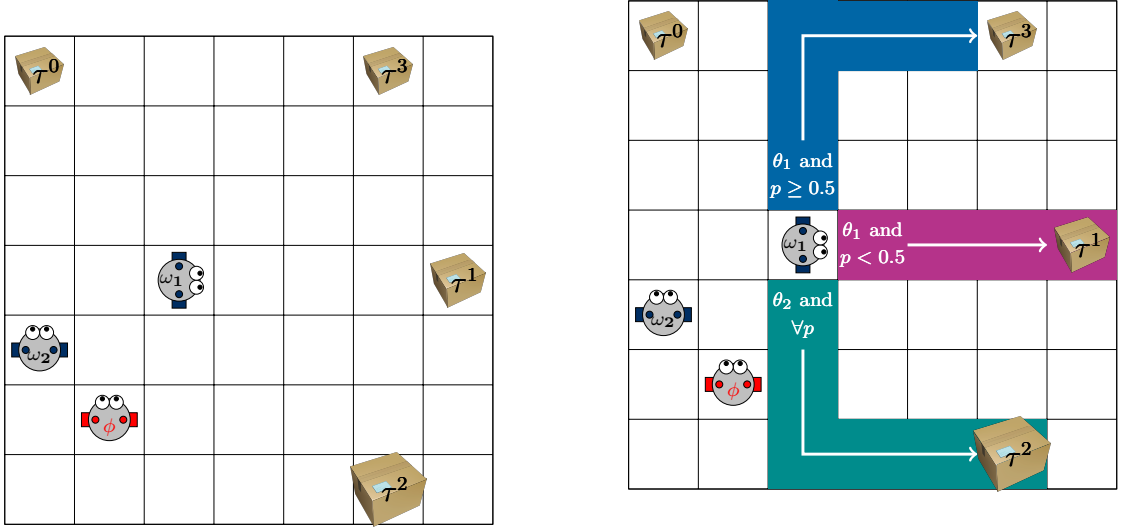
Example – For a better understanding of our method’s fundamentals, we will present a simple example. Let us consider a foraging domain [5, 111], in which there is a set of agents in a grid-world environment as well as some items. Agents in this domain are supposed to collect items located in the environment.

We show a simple scenario in Figure 5.3, in which there are two non-learning agents ω_1, ω_2 , one learning agent ϕ , and four items which are in two sizes. As in all foraging problems, each task is defined as collecting a particular item, so in this scenario there are four tasks τ^i . In addition, we have two types θ_1 and θ_2 , and two parameters (p_1, p_2) , where $p_1, p_2 \in [0, 1]$.

To keep the example simple, we consider that only p_1 affects ω_1 ’s decision-making at each state, and its behaviour follows the rules:

- If θ_1 , and $p_1 \geq 0.5$, then ω_1 goes towards small and furthest item (τ^3).
- If θ_1 , and $p_1 < 0.5$, then ω_1 goes towards small and closest item (τ^1).
- If $\theta_2, \forall p_1 \in [0, 1]$, ω_1 goes towards big and closest item (τ^2).

Therefore, in the example scenario, there are four sets of *estimators*, two for each ω agent : $\mathbf{E}_{\omega_1}^{\theta_1}, \mathbf{E}_{\omega_1}^{\theta_2}, \mathbf{E}_{\omega_2}^{\theta_1}, \mathbf{E}_{\omega_2}^{\theta_2}$. We assume that the total number of *estimators* in each set is 5 ($N = 5$). Furthermore, we maintain 4 bags of estimators : $\mathbf{B}_{\omega_1}^{\theta_1}, \mathbf{B}_{\omega_1}^{\theta_2}, \mathbf{B}_{\omega_2}^{\theta_1}, \mathbf{B}_{\omega_2}^{\theta_2}$. We assume that the true type of agent ω_1 is θ_1 , and the true parameter vector is $(0.2, 0.5)$. At this point, we will focus on the set of *estimators* for agent ω_1 . Moreover, we will continue to use this example to explain further details of OEATE.



(a) Current state where ϕ must reason about ω agents' behaviour (teammates).

(b) ϕ reasoning about ω agents' behaviour. In the illustration, ϕ considers three possible decisions for the agent ω_1 .

Figure 5.3: Example of ϕ thinking about ω 's behaviour, when performing foraging.

5.6.2 Process of Estimation

After presenting the fundamental elements of OEATE, we will explain how we define the process of estimating the parameters and type for each non-learning agent. Simultaneously, we will also demonstrate how OEATE evolves in various steps, using our above example. The algorithm is divided into five steps, which is executed for all agents in Ω at every iteration:

- (i) **Initialisation:** responsible for initialising the estimator set and the bags of successful estimators for each agent $\omega \in \Omega$.
- (ii) **Evaluation:** step where OEATE will increase the failure or the success score of each estimator, for all initialised estimator sets, based on the correct prediction of the ω 's target task. If the estimator successfully predicts the task, it will be added to its respective bag. Otherwise, it will be up for elimination.
- (iii) **Generation:** step where our method replaces the estimators removed in the evaluation process for new ones.

- (iv) **Estimation:** process of calculating the types' probabilities and expected parameters' value for each existing estimators set. The calculation is based on the success rate of each set.
- (v) **Update:** responsible for analysing the integrity of each estimator e and its respective chosen target τ_e given the current world state. If it finds some inconsistency, a new prediction is made considering ω 's perspective.

5.6.2.1 Initialisation

At the very first step, for each identified teammate in the environment, we initialise its *estimation set* and the *bag* for each possible type. Therefore, ϕ needs to create N *estimators* for each type $\theta \in \Theta$ and each $\omega \in \Omega$. If there is a lack of prior information, the parameter vectors \mathbf{p}_e of each *estimator* can be initialised with a random value from the uniform distribution \mathcal{U} , in each parameter's range. Since each *estimator* has a certain type θ and a certain parameter vector \mathbf{p}_e , it allows ϕ to estimate ω 's task choosing process. A task will be estimated and assigned to τ_e when, in a given state $s \in S$ at the time t , the prediction return a valid task. In the case where there is no valid task at the state s and time t , τ_e receives "None" and will be updated in later iterations (process carried out by the *Update* step – Section 5.6.2.5). Finally, both c_e and f_e are initialised to zero.

The Algorithm 7 illustrates the initialisation process.

- **Initialisation Example** – Returning back to our example, in *Initialisation* step, we start by creating random *estimators*, as shown in Table 5.1.

To make the example simple, we define the *state* as only the position of agent ω_1 . Therefore, we set each \mathbf{s}_e (Choose Target State) with the initial position of ω_1 , which is (3, 4), and then we create the parameter vectors \mathbf{p}_e by randomly sampling from the uniform distribution, which should be done separately for both p_1 and p_2 . Agent ϕ simulates ω_1 's task decision-making process for each *estimator* in the sets

Algorithm 7 OEATE Initialisation Process

```

1: procedure INITIALISATION( $\Omega, \Theta, N, \mathbf{p}_{ranges}, s^t$ )
2:    $EstimatorSets \leftarrow \emptyset$ 
3:    $EstimatorBags \leftarrow \emptyset$ 
4:   for each  $\omega \in \Omega$  do
5:     for each  $\theta \in \Theta$  do
6:        $EstimatorSets \leftarrow EstimatorSets \cup \mathbf{E}_\omega^\theta$ 
7:       while  $|\mathbf{E}_\omega^\theta| < N$  do
8:          $\mathbf{p}_e \leftarrow \mathcal{U}_e(\mathbf{p}_{min}, \mathbf{p}_{max}, \theta)$  ▷ Generating the estimator
9:         from uniform distribution  $\mathcal{U}$ 
10:         $c_e, f_e \leftarrow 0, 0$ 
11:         $\mathfrak{s}_e \leftarrow s^t$ 
12:         $\tau_e \leftarrow predict_\omega(s^t, \theta, \mathbf{p}_e)$ 
13:         $\mathbf{E}_\omega^\theta \leftarrow \mathbf{E}_\omega^\theta \cup e$ 
    
```

$\mathbf{E}_{\omega_1}^{\theta_1}$ and $\mathbf{E}_{\omega_1}^{\theta_2}$, and obtains the corresponding target task τ_e based on the type and parameter of each *estimator*. In addition, all f_e and c_e will be initialised as zero. All initial *estimators* for both sets are shown in Table 5.1.

$\mathbf{p}_e(p_1, p_2)$	\mathfrak{s}_e	τ_e	c_e	f_e	$\mathbf{p}_e(p_1, p_2)$	\mathfrak{s}_e	τ_e	c_e	f_e
(0.4, 0.6)	(3, 4)	τ^1	0	0	(0.1, 0.3)	(3, 4)	τ^2	0	0
(0.5, 0.3)	(3, 4)	τ^3	0	0	(0.8, 0.7)	(3, 4)	τ^2	0	0
(0.6, 0.2)	(3, 4)	τ^3	0	0	(0.3, 0.5)	(3, 4)	τ^2	0	0
(0.2, 0.5)	(3, 4)	τ^1	0	0	(0.6, 0.9)	(3, 4)	τ^2	0	0
(0.9, 0.8)	(3, 4)	τ^3	0	0	(0.2, 0.1)	(3, 4)	τ^2	0	0

(a) Initial *estimators* for type θ_1 (b) Initial *estimators* for type θ_2

Table 5.1: OEATE’s Initialisation example. *Estimator* sets $\mathbf{E}_{\omega_1}^{\theta_1}$ and $\mathbf{E}_{\omega_1}^{\theta_2}$ obtained from the *Initialisation* step.

5.6.2.2 Evaluation

The evaluation of all sets of *estimators* \mathbf{E}_ω^θ for a certain agent ω starts when it completes a task τ_ω . The objective of this step is to find the *estimators* that could estimate ω 's just completed real task τ_ω correctly. Therefore, we present the Algorithm 8 to facilitate the understanding of the evaluation process.

Algorithm 8 OEATE Evaluating Estimators

```

1: procedure EVALUATION( $\tau_\omega, \omega, s^t$ )
2:   for each  $\theta \in \Theta$  do
3:     for each  $e \in \mathbf{E}_\omega^\theta$  do
4:       if  $\tau_\omega = \tau_e$  then
5:          $\mathbf{B}_\omega^\theta \leftarrow \mathbf{B}_\omega^\theta \cup \mathbf{p}_e$            ▷ Parameters are added with repetition.
6:          $c_e \leftarrow c_e + \text{score}(e)$ 
7:       else
8:          $f_e \leftarrow f_e + \text{score}(e)$ ;
9:       if  $c_e / (c_e + f_e) < \xi$  then           ▷ Checking if estimator success rate >  $\xi$ 
10:         $\mathbf{E}_\omega^\theta \leftarrow \mathbf{E}_\omega^\theta \setminus e$            ▷ Removing  $e$  from  $\mathbf{E}_\omega^\theta$ 
11:      else
12:         $\mathfrak{s}_e \leftarrow s^t$ 
13:         $\tau_e \leftarrow \text{predict}_\omega(s^t, \theta, \mathbf{p}_e)$            ▷ Assigning new task
14:                                                                to survived estimators

```

As there are sets of *estimators* for each type $\theta \in \Theta$, then for every e in \mathbf{E}_ω^θ , we check if the τ_e (estimated task by assuming \mathbf{p}_e to be ω 's parameters with type θ) is equal to τ_ω (the real completed task). If they are equal, we consider them as successful parameters and save the \mathbf{p}_e vector in the respective bag \mathbf{B}_ω^θ (Line 5). The union between bag and parameter, which is applied in the equation, means that new parameters would be added to the bag with repetition, and if a parameter succeeds many times, it will appear in the bag with the same numbers of successes, so the chance of selecting it would be higher.

If the estimated task τ_e is equal to the real task τ_ω , we will increase the c_e following $c_e \leftarrow c_e + \text{score}(e)$. The $\text{score}(e)$ value denotes the information-level score for the prediction made by estimator e . The information-level score is used to represent the weighting given to certain task completions over others. For example: If a task prediction occurs many steps before the task completion, it was likely made by a correct estimator than by random chance. Furthermore, this function can be tweaked in a domain-specific way.

If the estimated task τ_e is not equal to the real task τ_ω , we will increase the f_e score following $f_e \leftarrow f_e + \text{score}(e)$. Note from the algorithm that we will only remove an estimator e if its success rate is lower than ξ (Line 9). We define the *threshold* ξ as a success threshold aiming to improve our estimator set, by removing the estimators that do not make good predictions and keeping the ones that do (more detail in the Generation explanation – Section 5.6.2.3).

Note that, by using this approach, any generated estimator e has a chance to be eliminated at the first iteration of estimation. Hence, some estimators, which may potentially approximate well the actual parameters, can be removed after performing their first estimation wrongly, $\forall \xi \in [0, 1]$. However, even if these particles fail at the beginning of the estimation, other estimators may also likely fail in the subsequent iterations of OEATE, enabling the regeneration of the removed potentially correct estimator through the bags or by sampling it again from the uniform distribution. As we will show in Section 6.3, OEATE estimates the correct parameter for all agents as the number of completed tasks grows and under some assumptions. Finally, the *Choose target State* (\mathfrak{s}_e) of the successful estimators is updated and a new task (τ_e) is predicted using the type and parameters of the estimator. The evaluation process ends and the removed estimators will be replaced by new ones in the Generation.

- **Evaluation Example** – From the previous example, after the initialisation, the agents move towards their respective targets. Based on the true type and parameters of the agent ω_1 , after some iterations, the agent (ω_1) gets the item that corresponds

to the task τ^1 . For this example, and throughout our experimentation, we will use the number of steps required between predicting the task and completing the next task as the score (*information-level*) for the estimator for that prediction. Let us assume that the number of steps required by the agent ω_1 is 4 (3 for moving and 1 for completing). From Figure 5.4, the agent ω_1 's new position will be (6,4). We will use this value as the score for the estimators. Note that here, since all estimators chose the task at the same time, they will get the same score.

Whenever a task is done by an agent, the process of evaluation will start. Now, we carry out the next step of our process. In *Evaluation*, all *estimators* of the two sets $\mathbf{E}_{\omega_1}^{\theta_1}$, $\mathbf{E}_{\omega_1}^{\theta_2}$ will be evaluated. If the task τ of any *estimator* e equals to τ^1 , then its success counter c_e increases by $score(e)$, otherwise it remains the same. Also, in failing cases, the counter of failures f_e increases by $score(e)$. The updated values of the estimators are shown in Table 5.2.

$\mathbf{p}_e(p_1, p_2)$	\mathfrak{s}_e	τ_e	c_e	f_e	$\frac{c_e}{c_e+f_e}$	$\mathbf{p}_e(p_1, p_2)$	\mathfrak{s}_e	τ_e	c_e	f_e	$\frac{c_e}{c_e+f_e}$
(0.4, 0.6)	(3, 4)	τ^1	4	0	1	(0.1, 0.3)	(3, 4)	τ^2	0	4	0
(0.5, 0.3)	(3, 4)	τ^3	0	4	0	(0.8, 0.7)	(3, 4)	τ^2	0	4	0
(0.6, 0.2)	(3, 4)	τ^3	0	4	0	(0.3, 0.5)	(3, 4)	τ^2	0	4	0
(0.2, 0.5)	(3, 4)	τ^1	4	0	1	(0.6, 0.9)	(3, 4)	τ^2	0	4	0
(0.9, 0.8)	(3, 4)	τ^3	0	4	0	(0.2, 0.1)	(3, 4)	τ^2	0	4	0

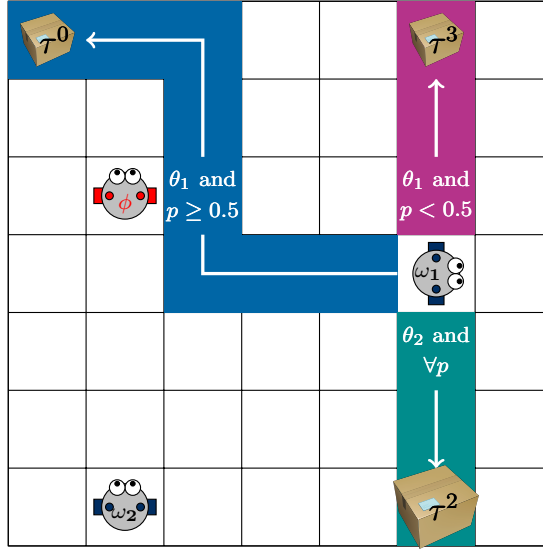
 (a) *Estimators* for type θ_1

 (b) *Estimators* for type θ_2

 Table 5.2: Evaluation example. *Estimator* sets $\mathbf{E}_{\omega_1}^{\theta_1}$, $\mathbf{E}_{\omega_1}^{\theta_2}$ after updating c_e and f_e .

If we suppose that the threshold for removing *estimators* is equal to 0.5 ($\xi = 0.5$), then we will have two surviving *estimators* ($\frac{c_e}{c_e+f_e} \geq \xi$) at $\mathbf{E}_{\omega_1}^{\theta_1}$ and none in $\mathbf{E}_{\omega_1}^{\theta_2}$. Hence, the bag for θ_1 are: $\mathbf{B}_{\omega_1}^{\theta_1} = \{(0.4, 0.6), (0.2, 0.5)\}$ and the bag for θ_2 is empty. Further, the new *Choose Target State* will be (6,4), which is used to find the new task (τ_e) for the surviving estimators. The new estimator sets are represented in Table 5.3 and, the new choose target state is illustrated by Figure 5.4.

$\mathbf{P}_e(p_1, p_2)$	\mathbf{s}_e	τ_e	c_e	f_e	$\frac{c_e}{c_e+f_e}$	$\mathbf{P}_e(p_1, p_2)$	\mathbf{s}_e	τ_e	c_e	f_e	$\frac{c_e}{c_e+f_e}$
(0.4, 0.6)	(6, 4)	τ^3	4	0	1	---	---	---	---	---	---
(0.2, 0.5)	(6, 4)	τ^3	4	0	1	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---	---

(a) Estimators for type θ_1 (b) Estimators for type θ_2
 Table 5.3: Evaluation example. Estimator sets $\mathbf{E}_{\omega_1}^{\theta_1}$, $\mathbf{E}_{\omega_1}^{\theta_2}$ after Evaluation step.

 Figure 5.4: New Choose Target state after ω_1 completing τ^1 . At this step, ω_1 will try to find a new task to pursue.

5.6.2.3 Generation

The generation process of new estimators occurs after every evaluation process and only over the removed estimators. In this step, the objective is to generate new *estimators*, in order to maintain the size of the $\mathbf{E}_{\omega}^{\theta} = N$.

Unlike the *Initialisation* step, we do not only create random parameters for new *estimators*, but generate a proportion of them using previously successful parameters from the *bags* \mathbf{B}_ω^θ . Therefore, we will be able to use a new combination of parameters from estimators that had successful predictions at least one time in previous steps. Moreover, as the number of copies of the parameter \mathbf{p} in the bag \mathbf{B}_ω^θ is equivalent to the number of successes of the same parameter in previous steps, the chance of sampling very successful parameters will increase according to its success rate.

The idea of using successful *estimators* to generate part of the new estimators is related to the Genetic Algorithm (GA) principles. Until now, the described process shares several similarities with the GA idea, such as the generation of a sample population for further evaluation and feature improvement. Furthermore, we are concerned about boosting our estimation process (based on the estimator sampling and evaluation), so we require a reasonable way to generate new estimators that can improve our estimation quality. Therefore, inspired by GAs mutation and cross-over process, we implement a GA-inspired process that supports our generation method.

Therefore, after the elimination of estimators for which the probability of making a correct prediction is lower than the threshold ξ , we will generate new estimators for our population following the mutation rate of m , where part of our population is generated randomly following a uniform distribution \mathcal{U} , and the rest following a process inspired by the cross-over, using our bags of successful parameters. With some domain knowledge, different distributions could be used. Figure 5.5 illustrates how the estimator set changes during this described process and indicates the portion of particles generated using the bags or randomly.

The generation process using the bags can be seen in Algorithm 9 Line 10-13 . There, a new estimator is created by sampling n different parameters (with repetition) from the target bag, and then choosing their i -th parameters. Hence, essentially if the parameter of new estimator (e^{new}) is $\mathbf{p}_{e^{new}} = \langle p_1, p_2, \dots, p_n \rangle$, then p_i is chosen by sampling $\mathbf{p}_{sampled} \sim \mathbf{B}_\omega^\theta$ and then taking the i -th parameter from it ($p_{i,sampled}$).

Algorithm 9 summarises this generation procedure.

Algorithm 9 Generating new estimators in OEATE

```

1: procedure GENERATION( $\omega, \Omega, \Theta, m, N, n\_removed, s^t$ )
2:    $n\_mutations \leftarrow m * n\_removed$  ▷ Calculating the number of
3:   mutations to perform.
4:   for each  $\theta \in \Theta$  do
5:     while  $n\_removed > 0$  do
6:        $e^{new} \leftarrow new\ Estimator()$  ▷ Initialising the new estimator.
7:       if  $n\_mutations > 0$  then ▷ Generating a estimator from
8:         the mutation process
9:          $\mathbf{p}_{e^{new}} \leftarrow \mathcal{U}_e(\mathbf{p}_{min}, \mathbf{p}_{max}, \theta)$ 
10:         $n\_mutations = n\_mutations - 1$ 
11:       else ▷ Generating an estimator using
12:         the bags
13:         for  $i = 0; i < n\_parameters; i = i + 1$  do
14:            $\mathbf{p}_{sampled} \sim \mathbf{B}_\omega^\theta$  ▷ Sampling a parameter
15:           from the bag
16:            $p_{i,e^{new}} \leftarrow p_{i,sampled}$  ▷ Assigning the  $i$ -th parameter
17:           of  $\mathbf{p}_{sampled}$  to  $\mathbf{p}_{e^{new}}$ 
18:          $n\_removed = n\_removed - 1$ 
19:          $\mathfrak{s}_{e^{new}} \leftarrow s^t$ 
20:          $\tau_{e^{new}} \leftarrow predict_\omega(s^t, \theta, \mathbf{p}_{e^{new}})$ 
21:          $\mathbf{E}_\omega^\theta \leftarrow \mathbf{E}_\omega^\theta \cup e^{new}$ 

```

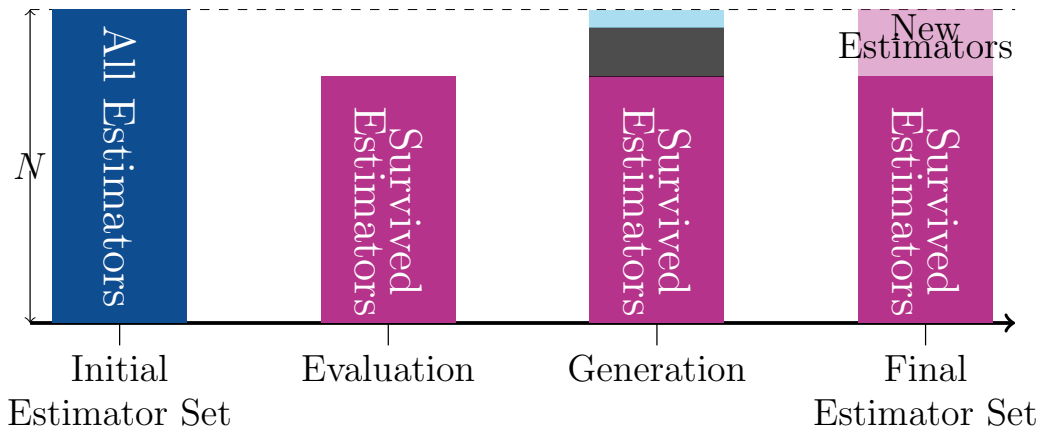
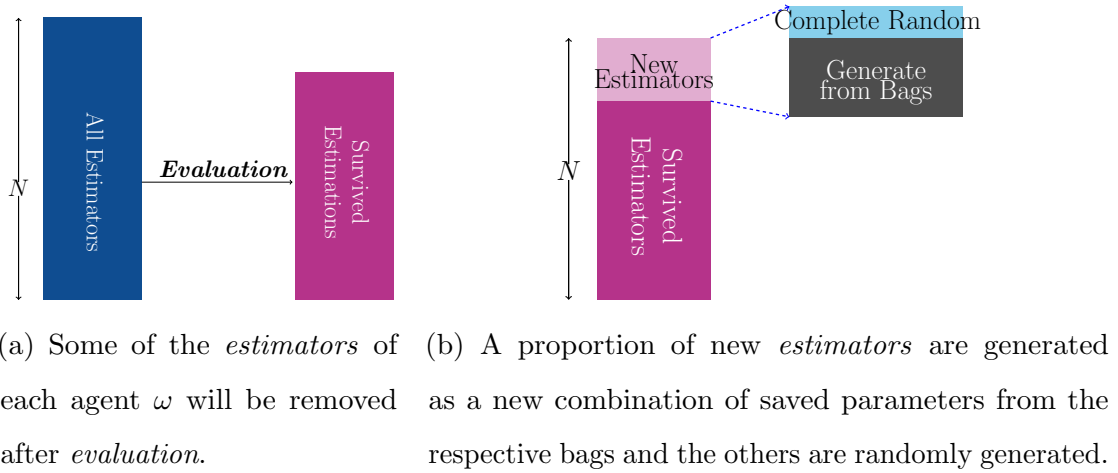


Figure 5.5: Estimation set modifications from the evaluation to the end of generation process. Figures (a) and (b) present the modifications after the evaluation and after the generation, respectively. Figure (c) presents the entire modification process.

After performing all the generations, we continue to fill the estimator set with uniform generated parameters. Once the estimator set is full (i.e., $|\mathbf{E}_\omega^\theta| = N$), the current state is assigned as *Choose Target State* ($\mathbf{s}_{e^{new}}$) of every new estimator. Afterwards, a task ($\tau_{e^{new}}$) is predicted for each new estimator and the process finishes.

- **Generation Example** – Supposing $m = \frac{1}{3}$ as mutation rate, then $(1 - \frac{1}{3}) \times (5 - 2) = 2$ new *estimators* are generated by randomly sampling from the

bags, while $\frac{1}{3} \times (5 - 2) = 1$ estimator is generated randomly from the uniform distribution. Therefore, we may create new estimators with the following parameters: $(0.4, 0.5); (0.2, 0.6); (0.8, 0.7)$, where the last vector is fully random. For $\mathbf{E}_{\omega_1}^{\theta_2}$, as all estimators were removed and the corresponding bags are empty, the whole set $\mathbf{E}_{\omega_1}^{\theta_2}$ will be generated using the uniform distribution as in the initialisation process. After this, the current state $(6,4)$, is assigned as the *Choose Target State* for each new estimator and a task is predicted.

All new estimators and updated values are shown in Table 5.4.

$\mathbf{p}_e(p_1, p_2)$	\mathfrak{s}_e	τ_e	c_e	f_e	$\frac{c_e}{c_e+f_e}$	$\mathbf{p}_e(p_1, p_2)$	\mathfrak{s}_e	τ_e	c_e	f_e	$\frac{c_e}{c_e+f_e}$
$(0.4, 0.6)$	$(6, 4)$	τ^3	4	0	1	$(0.1, 0.3)$	$(6, 4)$	τ^2	0	0	0
$(0.2, 0.5)$	$(6, 4)$	τ^3	4	0	1	$(0.8, 0.7)$	$(6, 4)$	τ^2	0	0	0
$(0.4, 0.5)$	$(6, 4)$	τ^3	0	0	0	$(0.3, 0.5)$	$(6, 4)$	τ^2	0	0	0
$(0.2, 0.6)$	$(6, 4)$	τ^3	0	0	0	$(0.6, 0.9)$	$(6, 4)$	τ^2	0	0	0
$(0.8, 0.7)$	$(6, 4)$	τ^0	0	0	0	$(0.2, 0.1)$	$(6, 4)$	τ^2	0	0	0

(a) Estimators for type θ_1

(b) Estimators for type θ_2

Table 5.4: Generation example. Estimator sets $\mathbf{E}_{\omega_1}^{\theta_1}$ and $\mathbf{E}_{\omega_1}^{\theta_2}$ sets after the Generation.

5.6.2.4 Estimation

At each iteration after doing *evaluation* and *generation*, it is required to estimate a parameter and type for each $\omega \in \Omega$ to improve the decision-making. First, based on the current sets of estimators, we calculate the probability distribution over the possible types. For calculating the probability of agent ω having type θ , $\mathbf{P}(\theta)_\omega$, we use the success score c_e of all estimators of the corresponding type θ . For each $\omega \in \Omega$, we add up the success rates c_e of all estimators in \mathbf{E}_ω^θ of each type θ , that is:

$$k_\omega^\theta = \sum_{e \in \mathbf{E}_\omega^\theta} c_e, \quad \forall \theta \in \Theta$$

It means that we want to find out which set of *estimators* is the most successful in estimating correctly the tasks that the corresponding non-learning agent completed. In the next step we normalise the calculated k_ω^θ , to convert it to a probability estimation, following:

$$P(\theta)_\omega = \begin{cases} \frac{k_\omega^\theta}{\sum_{\theta' \in \Theta} k_\omega^{\theta'}} & \sum_{\theta' \in \Theta} k_\omega^{\theta'} > 0 \\ \frac{1}{N} & \text{else} \end{cases}$$

During the simulations, OEATE will sample estimations from the current estimation sets. In detail, for each agent ω , we will sample a type θ based on $P(\theta)_\omega$ and sample an estimator from ω 's estimator set of that type (\mathbf{E}_ω^θ), using the weights given by c_e of the estimators. In this way, once a type (θ) is selected, the probability of selection of each estimator $e \in \mathbf{E}_\omega^\theta$ is equals to c_e/k_ω^θ . If $k_\omega^\theta = 0$, we sample the estimator uniformly from \mathbf{E}_ω^θ . Otherwise, we perform the weighted sampling.

Using this strategy, OEATE can improve the reasoning horizon and diversify the simulations. Differently from AGA and ABU that presents only a single estimation per iteration, we present a set of the (current) best found estimators for planning and decision-making.

• **Estimation Example** – Now, we do the *Estimation* step in our example to have a probability distribution over types, and one parameter vector per type of ω_1 . At this step, in order to find the probability of being either θ_1 or θ_2 , we apply the Equation 5.6.2.4. By considering the c_e of all *estimators*, we have that:

$$k^{\theta_1} = 8, k^{\theta_2} = 0,$$

Hence, to calculate the probability of each type, we use the Equation 5.6.2.4. Accordingly, the probabilities are:

$$P'(\theta_1) = \frac{8}{8+0} = 1, P'(\theta_2) = \frac{0}{8+0} = 0,$$

which means that the probability of being θ_1 is the higher one.

Now, for the sampling process, we sample a type using the previously calculated distribution. Let's say that we sample θ_1 . Now, from this type, we also sample an estimator, using the ratio c_e/k^{θ_1} as the probability of each estimator in $E_{\omega_1}^{\theta_1}$. Concretely, we get:

$$\mathbf{P}((0.4, 0.6)|\theta_1) = \frac{4}{8}, \mathbf{P}((0.2, 0.5)|\theta_1) = \frac{4}{8}$$

while the other estimators have probability 0. So, we use these probabilities to sample an estimator, let's say (0.4,0.6). Therefore, type θ_1 and the parameters (0.4,0.6) will be our estimated type and parameter for the current estimation step.

During the planning phase in the root of the MCTS (for the learning agent ϕ perspective), the OEATE will sample the simulating type and parameter respecting the probabilities calculated above. Moreover, to calculate the error of the estimation of our method, we use the mean square error (MSE) between the true parameter and the expected parameter of the true type (θ^*). The expected parameter of a type (θ) and agent ω is calculated as:

$$\mathbf{P}_{exp} = \sum_{e \in \mathbf{E}_{\omega}^{\theta}} \frac{c_e}{k_{\omega}^{\theta}} \mathbf{P}_e$$

5.6.2.5 Update

As mentioned earlier, there are possible issues that might arise in our estimation process, they occur:

- (i) when a certain task τ is accomplished by any of the team members (including agent ϕ), and some other non-learning agent was targeting to achieve it, or;
- (ii) when a certain non-learning agent is not able to choose a task to target (e.g., cannot see or find any available (or valid) task within its vision area considering possible parameters limitations, such as vision radius and angle).

If some non-learning agent ω faces one of these problems, it will keep trying to find a task to pursue. Hence, from the perspective of the learning agent ϕ , OEATE

must handle this problem updating its teammates' targets. Otherwise, it might incorrect evaluate the available estimators given the outdated prediction.

Algorithm 10 Updating the OEATE Estimators

```

1: procedure UPDATE( $s^t, \Omega, \Theta$ )
2:   for each  $\omega \in \Omega$  do
3:     for each  $\theta \in \Theta$  do
4:       for each  $e \in \mathbf{E}_\omega^\theta$  do
5:         if no task or valid task was assigned to  $\tau_e$  then  $\triangleright$  Agent cannot
6:           find or see a valid task
7:            $\tau_e = \text{predict}_\omega(s^t, \theta, \mathbf{p}_e)$   $\triangleright$  Predicting a task
8:            $\mathfrak{s}_e \leftarrow s^t$ 
9:         else if  $\tau_e$  was completed by other agent then  $\triangleright$  Task completed
10:          by other agent
11:           $\tau_e = \text{predict}_\omega(s^t, \theta, \mathbf{p}_e)$   $\triangleright$  Predicting a task
12:           $\mathfrak{s}_e \leftarrow s^t$   $\triangleright$  Note that  $\tau_e$  can be  $\emptyset$ .

```

Therefore, the OEATE's Update process exists to guarantee the estimator set integrity for future evaluation. At each iteration, the update step will analyse the integrity of each estimator e and its respective chosen target τ_e given the current world state. If it finds some inconsistency, it will simulate the estimator's task selection for the next states, considering ω 's perspective. The process is carried out in each successive state until it returns a new valid target for the indecisive estimator. The Algorithm 10 presents the described update routine.

• **Update Example** – In the update step, we look at our estimators from Table 5.4 and check whether the conditions for update (from Algorithm 10) are met. Evidently, for our case, we see that every estimator has a valid task assigned to it and therefore, nothing will happen in the update step.

5.7 OEATE with Partial Observability

Assuming full visibility for the learning agent is a strong presupposition and it rarely occurs in a real application (due to data or technology limitations). Thus, towards a more realistic application, we considered scenarios where agent ϕ is working with limited visibility of the environment. Therefore, we formalise our problem as a *single agent* POMDP model, which will allow us to adapt POMCP [93] with our *OEATE*, as explained in Section 2.2.3. In this section, we will outline the main changes compared to our previous MDP model (presented in the Background, Section 2.1.1, and adapted to our ad-hoc context in Section 5.3) and how we designed our POMCP-based solution for distributed task execution problems into an ad-hoc teamwork context.

5.7.1 POMDP model

Our POMDP model considers one agent ϕ acting in the same environment as a set of non-learning agents ($\omega \in \Omega$), and ϕ tries to maximise the team performance without any initial knowledge about ω agents' types and parameters. We consider the same set of states \mathcal{S} , action \mathcal{A} , transition \mathcal{T} and reward function \mathcal{R} defined previously. Additionally, ϕ 's objective is still to maximise the expected sum of discounted rewards. However, now ϕ has a set of observations \mathcal{O} that defines its current state. Every action a produces an observation $o \in \mathcal{O}$, which is the visible environment in agent ϕ 's point of view (all of the environment within the *visibility region*, in the state s' reached after taking action a). We assume ϕ can perfectly observe the environment within the *visibility region*, but it cannot observe anything outside the *visibility region*. Hence, our POMDP model works within a observation function which is deterministic instead of stochastic – so, all values denote *empty square*, *agent* or *task*. True types and parameters are not observable.

The state cannot be observed directly by ϕ , so it builds a history \mathcal{H} instead. \mathcal{H} consists of a set of collected information h_t from the initial timestamp $t = 0$ until the current time. Each h_t is an action and observation pair ao , representing the action a

taken at time t , and the corresponding observation o that was received. The current agent history will define its *belief state*, which is a probability distribution across all possible states. Therefore, ϕ must find the optimal action for each *belief state*.

This formalisation enables the extension of our planning model, from a full observable context using MCTS to a partially observable context for POMCP application. This transition to a POMCP application is a straightforward process, however, we make further modifications to guarantee the on-line estimation and planning features, which OEATE presents.

5.7.2 POMCP modification

POMCP [93] is an extension of UCT for problems with partial observability. The algorithm uses an unweighted particle filter to approximate the belief state at each node in the UCT tree and requires a *simulator*, which is able to sample a state s' , reward r and observation o , given a state and action pair. Each time we traverse the tree, a state is sampled from the particle filter of the root. Given an action a , the simulator samples the next state s' and the observation o . The pair ao defines the next node n in the search tree, and for the current iteration, the state of the node will be assumed to be s' . This sampled state s' is added to node n 's particle filter, and the process repeats recursively down the tree.

However, we do not know the true transition and reward functions, since they depend on the pdfs of the non-learning agents ($\omega \in \Omega$). Therefore, we employ the same strategy as previously: at each time we go through the search tree, we sample a type for each agent from the estimated type probabilities and use the parameters that correspond to the sampled type. These remain fixed for the whole traversal until we re-visit the root node for the next iteration. Note that these sampled types and parameters are also going to be used in the POMCP *simulator*, when we sample a next state, a reward and an observation after choosing an action in a certain node.

As mentioned previously, POMCP has been modified before to sample transition functions [45]. Here, however, we are employing a technique that is commonly used

in UCT (for MDPs) in ad-hoc teamwork [5, 14], but now in a partially observable scenario, which allows us to work on the type/parameter space instead of directly on the complex transition function space. In this way, we can then employ OEATE for the type and parameter estimation.

The same OEATE algorithm described in Section 5.6 can handle the cases where any agent $\omega \in \Omega$ is outside the agent ϕ 's visibility region. In order to do so, it samples a particle from the POMCP root, which corresponds to sampling a state from the *belief state*. That allows us to have complete (estimated) states when predicting tasks for ω agents. States that are considered more likely will be sampled with a higher probability for the OEATE algorithm following the POMCP belief state filtering probabilities. However, we assume in our implementation (and in all algorithms we compare against) that agent ϕ knows when an agent ω has completed a task τ , even if it is outside our visibility region. That is, agent ϕ would know exactly which task was completed by a certain agent. That would require in a real application some global signal of task completion (e.g., boxes with radio transmitters).

5.8 Theoretical Analysis

In this section, we analyse, theoretically, OEATE's capability to optimally estimate types and parameters while performing online planning. We show that as the number of tasks goes to infinite, under full observability, OEATE perfectly identifies the type and parameters of all agents ω , given some assumptions. Our analysis follows:

► First, since each of our updates are related to completing the tasks, this analysis assumes that the agents are able to finish the tasks in the environment. Additionally, we consider that parameters have a finite number of decimal places. This is a light assumption, as any real number x can be closely approximated by a number x' with finite precision, without much impact in a real application (e.g., any computer has a finite precision). Hence, as each element p_i in the parameter vector is in a fixed

range, there is a finite number of possible values for it. To simplify the exposition, we consider ψ possible values per element (in general they can have different sizes).

Let n be the dimension of the parameter space. Additionally, let \mathbf{p}^* be the correct parameter, and θ^* be the correct type, of a certain agent ω . We define $\theta^- \neq \theta^*$, and $\mathbf{p}^- \neq \mathbf{p}^*$, representing wrong types and parameters, respectively. We will also use tuples (\mathbf{p}, θ) to represent a pair of parameter and type.

Assumption 5.1 *Any (\mathbf{p}, θ^-) , and any (\mathbf{p}^-, θ^*) has a lower probability of making a correct task estimation than (\mathbf{p}^*, θ^*) , and the correct parameter-type pair (\mathbf{p}^*, θ^*) will also be able to have the correct Choose Target State (\mathfrak{s}_e).*

► This assumption is light under our defined domain because if a certain pair (\mathbf{p}, θ^-) or (\mathbf{p}^-, θ^*) has a higher probability of making correct task predictions, then it should indeed be the one used for planning, and could be considered as the correct type-parameter pair. We consider this assumption to be reasonable given our defined scenario of investigation, where the environment is fully observable, the templates used to simulate the teammates are parameterised, and the teammates' policies are mostly deterministic, as presented in our problem formulation and discussed throughout our methodology (Sections 5.4 and 5.6, respectively). For more stochastic or uncertain decision-making scenarios, this assumption would need to be supported by more sophisticated techniques from theoretical statistics in order to hold more generally.

Assumption 5.2 *Any (\mathbf{p}, θ^-) , and any (\mathbf{p}^-, θ^*) will not succeed infinitely often. That is, as $|\mathbb{T}| \rightarrow \infty$ there will be cases where it successfully predicts the task, but the number of cases is limited by a finite constant c .*

Assumption 5.3 *The assumption has 2 parts: (i) a correct value p_i^* in any position i may still predict the task wrongly (since other vector positions may be wrong), but it will eventually predict at least one task correctly in at most t trials, where t is a constant; (ii) a wrong value p_i^- in any position i may still predict the task correctly*

(since other vector positions may be correct), but that would happen at most \mathfrak{b} times for each bag, across all wrong values. Therefore, $\mathfrak{b} \ll \psi$.

► This assumption is needed to distinguish our method from a random search. If one of the vector positions i is correct, \mathbf{p} will not fail infinitely, even though other elements may be incorrect. That is valid in many applications (considering our defined domain), as in some cases only one element is enough to make a correct prediction. For example, if a task was nearby, for almost any vision radius it would be predicted as the next one if the vision angle were correct. On the other hand, wrong values will not always succeed. That is also true in many applications: although by the argument above, wrong values may make correct predictions, but these are a limited number of cases in the real world. Eventually, all tasks nearby will be completed, and a correct vision radius estimation becomes more important to make correct predictions. Usually, ψ would be large (e.g., they may approximate real numbers), so we would have $\mathfrak{b} \ll \psi$. Additionally, we will consider the case with lack of previous knowledge, so parameters and types will be initially sampled from the uniform distribution. As before, we denote by $\mathbf{P}(\theta)$ the estimated probability of a certain agent having type θ , but we drop the subscript ω for clarity.

Theorem 5.1 *OEATE estimates the correct parameter for all agents as $|\mathbb{T}| \rightarrow \infty$. Hence, $\mathbf{P}(\theta^*) \rightarrow 1$.*

Proof: Since wrong parameters-type pairs will not succeed infinitely often, we always will generate new *estimators* with a random \mathbf{p}_e . As we sample from the uniform distribution, \mathbf{p}^* will be sampled with probability $1/\psi^n > 0$. Hence, eventually it will be generated as $|\mathbb{T}| \rightarrow \infty$. As the generation defines a Bernoulli experiment, from the geometric distribution, we expect ψ^n trials.

Therefore, eventually, there will be an *estimator* with the correct parameter vector \mathbf{p}^* . Furthermore, since (\mathbf{p}^*, θ^*) has the highest probability of making correct predictions (Assumption 5.1), it has the lowest probability of reaching the failure threshold ξ . Hence, as $|\mathbb{T}| \rightarrow \infty$, there will be more *estimators* (\mathbf{p}^*, θ^*) , than any

other *estimator*. Further, any (\mathbf{p}^-, θ^*) will eventually reach the failure threshold, and will eventually be discarded, since it succeeds at most c times by Assumption 5.2. Therefore, by considering our method of sampling an estimator from the estimator sets, we will correctly estimate \mathbf{p}^* when assuming type θ^* . Hence, when $|\mathbb{T}| \rightarrow \infty$ the *sampled estimator* from $\mathbf{E}_\omega^{\theta^*}$ will be \mathbf{p}^* .

Further, when we consider the Assumption 5.2, then the probability of the correct type $P(\theta^*) \rightarrow 1$. That is, we have that $c_e \rightarrow \infty$ in the set $\mathbf{E}_\omega^{\theta^*}$. Hence, $k_\omega^{\theta^*} \rightarrow \infty$, while $c_e < c$ for θ^- (by assumption). Therefore:

$$P(\theta^*) = \frac{k_\omega^{\theta^*}}{\sum_{\theta' \in \Theta} k_\omega^{\theta'}} \rightarrow 1,$$

while $P(\theta^-) \rightarrow 0$, as $|\mathbb{T}| \rightarrow \infty$. ■

► This ensures that the as $|\mathbb{T}| \rightarrow \infty$, the sampled type is θ^* . Moreover, we saw in Theorem 5.1 that a random search from the mutation proportion takes ψ^n trials in expectation. OEATE, however, finds \mathbf{p}^* much quicker than that, since a proportion of *estimators* are sampled from the corresponding bags $\mathbf{B}_\omega^{\theta, i}$. In the following proposition, we will prove that OEATE will indeed find \mathbf{p}^* and under Assumption 5.1, \mathbf{p}^* would have highest probability of not being removed from the estimator set and will continue to add it's own parameters back to the bag, thereby further increasing the probability of sampling those parameters at each mutation.

Proposition 5.1 *OEATE finds \mathbf{p}^* in $O(n \times \psi \times (\mathbf{b} + 1)^n)$.*

Proof: Consider Assumption 5.3, we know that at some time, we must encounter a parameter value p_i . Sampling the correct value for element p_i would take ψ trials in expectation. Once a correct value is sampled, it will be added to $\mathbf{B}_\omega^{\theta^*}$ if it makes at least one correct task prediction. It may still make incorrect predictions because of wrong values in other elements, and it would be removed (from the estimator set) if it reaches the failure threshold ξ . However, for a constant number of trials $t \times \psi$, it would be added to $\mathbf{B}_\omega^{\theta^*}$. Similarly, sampling the correct value for all n dimensions

at least one time would take $n \times \psi$ trials in expectation, and in at most $t \times n \times \psi$ trials $\mathbf{B}_\omega^{\theta^*}$ would have at least one estimator each with the correct value in position i . The bags store repeated values, but in the worst case, there is only one correct example at each $\mathbf{B}_\omega^{\theta^*}$, leading to at least $1/(\mathbf{b} + 1)$ probability to sample the correct value from the bag. Hence, given the bag sampling operation, we would find \mathbf{p}^* with at most $t \times n \times \psi \times (\mathbf{b} + 1)^n$ trials in expectation. Hence, the complexity is close to $O(\psi)$, instead of $O(\psi^n)$ as the random search. Moreover, note that this is true because $\mathbf{b} \ll \psi$. ■

Considerations – In Assumption 1, the *choose target state* (\mathbf{s}_e) of an estimator is dependent only on the previous predicted tasks and the main agent’s observation. Therefore, in a fully observable case, the true parameters have the highest probability of having the correct *choose target state*. Furthermore, we leave the proof for partially observable cases as future work.

Time Complexity – It is worth noting that the actual time taken by the algorithm is dependent on $(\mathbf{b} + 1)^n$. So, as an example, if $\mathbf{b} = 10 \ll \psi = 100$, then if $n = 3$, $(\mathbf{b} + 1)^n = 1000 \gg \psi = 100$. However, when we write the time complexity, we are focusing on how the algorithm will scale with larger search space (i.e., higher ψ). Further, since ψ is the precision of parameters, it is likely to be a large value.

► For instance, if there are 3 elements in parameter vector (\mathbf{p}), if range of each element (p_i) is $[0,1]$ and we want our answer to be accurate up to only 3 places of decimal, then $\psi = 10^3$.

5.9 Study Benchmarks

5.9.1 Level-based Foraging Domain

The level-based foraging domain is a common problem for evaluating ad-hoc teamwork [3, 5, 111]. In this domain, a set of agents collaborate to collect items displaced in a rectangular grid-world environment in a minimum amount of time (Figure 5.6).

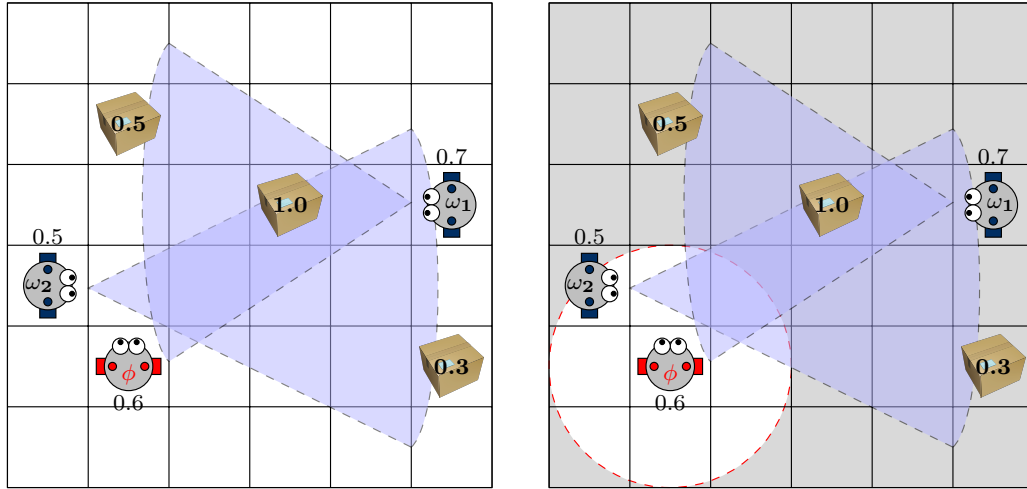
Additionally, items have a certain weight, and agents have a certain skill level, which defines how much weight they can carry. Hence, agents may need to collaborate to pick up a particularly item. Further, we assume that tasks are spawning in the environment during the execution.

Differently from Albrecht and Stone (2017) [5], and Yourdshahi et al. (2018) [111], this approach enables a continuous level of information in the scenario, which ϕ must analyse and reason about to improve the team's performance. The performance here will regard the number of completed tasks in the environment instead of the necessary time to complete all tasks. Concretely, we define the number of tasks that can be in the environment simultaneously. If some agent (or group of agents) accomplishes a task, we spawn a new one for each completion at that execution time. In this way, we manage to maintain a fixed number of tasks in the environment, hence the same problem level from the beginning to the end.

Finally, we defined this problem over full and partial observability, which Figure 5.6 illustrates possible scenarios configuration.

• **Agent's Parameters** – Each agent has a visibility region and can only choose items as a target if they are in its visibility cone. Therefore, to know which items are in the visibility area of each agent, we need to have the *View Angle* and the maximum *View Radius* of the agents. Additionally, each agent has a *Skill Level* which defines its ability to collect items. Also, each item has a certain weight, so each agent can collect items that have a weight below their *Skill Level* or equal to it. Based on what we described above, each agent can be defined by three parameters:

- l , which specifies the *Skill Level* and $l \in [0.5, 1]$;
- a , which is referring to *View Angle*. The actual angle of the visibility cone is given by the formula $a * 2\pi$. Additionally, it is assumed that $a \in [0.5, 1]$;
- r , which is referring to the *View Radius* of the agent. The actual *View Radius* is given by $r\sqrt{w^2 + h^2}$, where w and h are the width and height of the grid.



(a) Level-based foraging domain. The number next to the boxes indicate their weight, and the one next to agents indicate their skill levels. The coloured area represents the vision area of ω agents. ϕ has full visibility of the environment.

(b) Level-based foraging domain. The number next to the boxes indicate their weight, and the one next to agents indicate their skill levels. The coloured area represents the vision area of ω agents. ϕ has partial visibility of the environment.

Figure 5.6: Possible problem scenarios in the defined Level-based Foraging Domain.

Also, the range of the radius is $r \in [0.5, 1]$.

All of these parameters are applicable to all $\omega \in \Omega$. Agent ϕ has the parameter *Skill Level* when it has either full or partial observability, but the *View Angle* and *View Radius* parameters are only applicable when it has partial observability.

• **Agent’s Types** – Concerning types of non-learning agents, we took inspiration from [5], type definitions in the foraging domain. They considered four possible types for the agents in Ω : two “leader” types, which choose items in the environment to move towards, and two “follower” types, which attempt to go towards the same items as other agents, in order to help them load items. However, “follower” agents may also choose other agents as target, while in our work we handle agents that choose tasks as target. Therefore, we only consider “leader” agents in our work. Hence, based on agent ω ’s type and parameter values, a target item will be selected, and the agent’s internal state (memory) will be set to the position of that target. Afterwards, the agent will move towards the target using the A^* algorithm [47]. Here is the detail for how the different types choose their targets:

- $L1$: if there are items visible, return the furthest item; else, return \emptyset .
- $L2$: if there are items visible, return the item with highest sum of coordinates; else, \emptyset .
- $L3$: if there are items visible, return the closest item; else, return \emptyset .
- $L4$: if there are items visible, return the item with lowest sum of coordinates; else, \emptyset .
- $L5$: if there are items visible, return the first item found (considering the orientation: west to east, north to south); else, \emptyset .
- $L6$: if there are items visible, return an random item; else \emptyset .

• **Actions** – Each agent has five possible actions: *North*, *South*, *East*, *West*, *Load*. The first four actions will move the agent towards the selected direction if the destination cell is empty and it is inside the grid. The fifth action, *Load*, helps the agent to load its target item. The only time that an agent can collect an item is when the item is next to the agent, and the agent is facing it. Also, for loading the

item, the *Skill Level* of the agent should be equal to or higher than the items' weight. If the agent does not have enough *Skill Level* to collect the item, then a group of agents can do the job if the sum of the *Skill Levels* of the agents that surround the target is greater than or equal the item's weight. Therefore, the item can be "loaded" by a set of agents or just one agent. In the situation when the agent does not have enough ability to collect the target item, it will standstill in the same place when issuing the *Load* action. In the case of collecting an item, the team of agents receives a reward and it will be removed from the grid.

- **Foraging Process** –: First of all, we describe the process of foraging and choosing a target for agents ω in Algorithm 11 in order to facilitate the understanding.

In the very first step as ω has not chosen any target, the *Mem*, which holds the target item, is initialised to \emptyset . In Line 9, the *VisibleItems* routine is called, which gets the agent ω 's parameters, *View Angle* and *View Radius*, and returns a set containing the visible items. In Line 10, the *ChooseTarget* routine gets the *Skill Level* and *Type* of the ω agent, and the list of visible items, returned from *VisibleItems* routine as input. The output of this routine is the target item that agent ω should go towards. In Line 14, there might be cases where agent ω is not able to find any target task. Hence, all actions get equal probabilities and consequently, it will perform actions uniformly randomly until it is able to choose a task.

This is an algorithm's template that we assume non-learning agents are following. We use the same template in our simulations, but in practice agents, ω could follow different algorithms. Hence, in the results section, we will also evaluate the case where the agents *do not* follow the same algorithm as in our template.

5.9.2 Capture the Prey Domain

Intending to evaluate the present range of applicability of our proposal over different domains, we further perform experiments in the Capture-the-Prey domain.

This domain is presented as a discrete, rectangular grid-world as in Section 5.9.1.

Algorithm 11 Foraging

```
1: procedure MoveOmega (SkillLevel, ViewRadius, ViewAngle, Type)
2:   if item in Mem is collected then
3:     Mem  $\leftarrow$   $\emptyset$  ▷ Memory to keep target
4:     Loc  $\leftarrow$  location of  $\omega$ ;
5:     Dest  $\leftarrow$   $\emptyset$ 
6:     if Mem  $\neq$   $\emptyset$  then
7:       Dest  $\leftarrow$  Mem
8:     else ▷ Choose new target
9:       I  $\leftarrow$  VisibleItems(Loc, ViewRadius, ViewAngle)
10:      Targ  $\leftarrow$  ChooseTarget (SkillLevel, Type, I)
11:      if Targ  $\neq$   $\emptyset$  then
12:        Dest  $\leftarrow$  Targ
13:      Mem  $\leftarrow$  Dest
14:      if Dest =  $\emptyset$  then
15:        Assign probability 0.2 to each action
16:      else
17:        if Loc is next to Dest then
18:          Assign probability 0.96 to Load action
19:        else
20:          Use  $A^*$  to find path from Loc to Dest
21:          Assign probability 0.96 to first move action in path
22:          Add probability 0.01 to each move action
23:      Return pdf over actions
```

It is a variant of the Pursuit Domain described in [12, 14]. There are several “preys” in the environment, which represents the objectives that the Ad-hoc Team must pursue, similar to the “tasks” from our Level-based Foraging environment. However, the preys are also non-learning agents, which are running a reactive algorithm and trying to escape from being captured – defining decentralised tasks, which are moving in the scenario. Each prey can also be identified by a numeric index given to it. The ad-hoc team is composed of non-learning agents $\omega \in \Omega$ and a learning agent ϕ . They must surround the prey and capture it, which means to block the movement of the prey on all discrete four sides: North, South, East and West. It can be done only by agents, or with the support of walls and/or by other preys. Note that surrounding is mandatory, hence the agents must collaborate in the most efficient way in order to improve their performance. The tasks are re-spawning in this environment as well. Figure 5.7 illustrates the problem.

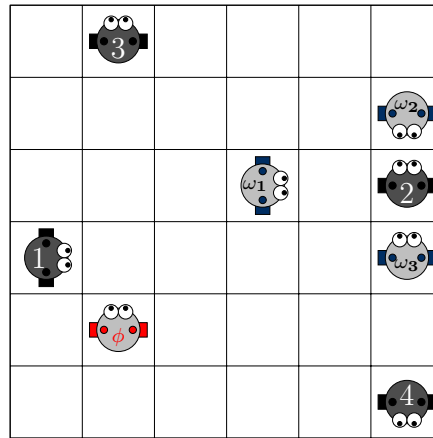


Figure 5.7: Possible problem scenario in the defined Capture the Prey Domain. The agent with red details is the learning agent ϕ . The agents with blue details are the non-learning agents $\omega \in \Omega$. The grey agents are the prey.

Agent’s Parameters – The parameter of each agent is the same as earlier, but there is no longer a *Skill Level* parameter since the completion condition is not related to task parameters. In this way, the ω agents still have a visibility region to see and choose targets, which follows:

- a , which is referring to *View Angle*. The actual angle of the visibility cone is given by the formula $a * 2\pi$. Additionally, it is assumed that $a \in [0.5, 1]$;
 - r , which is referring to the *View Radius* of the agent. The actual *View Radius* is given by $r\sqrt{w^2 + h^2}$, where w and h are the width and height of the grid. Also, the range of the radius is $r \in [0.5, 1]$.
- **Agent’s Types** – Concerning types of non-learning agents, we created 2 main types to run the experiments:
- *C1*: the spatial type of the set, which chooses the furthest visible prey to pursue, if there are visible preys in its vision area; else, return \emptyset .
 - *C2*: the index-based type, which chooses preys with an even identification, if there are visible preys in its vision area; else, return \emptyset .
- **Actions** – Each agent has 5 actions : *North, South, East, West, Block* . The first four actions will move the agent towards the selected direction if the destination cell is empty or it is inside the grid. The *Block* is the action that actually captures the prey, where the agent stands at its position blocking the passage of prey. Notice that there is no *Load* action, as the completion of the task (or “Capturing the Prey”) is done by the surrounding. So the agent must block one passage of the prey, trying to create a capture situation.
- Unlike in level-foraging, the tasks are no longer stationary. At each step, the tasks also move randomly to one of the empty squares next to them. If no such free space exists and, at least, one agent is surrounding the task, it gets captured. So, each task can be completed by at least 1 agent, depending on the location of the task and the whole state configuration (such as other agents positions, other preys positions and map borders).
- **Capturing Process** – Directly, the process of choosing actions and targets remains similar to the process defined for Foraging by Algorithm 11.

5.10 Results

5.10.1 Evaluation Settings

- **Baselines** – We will compare our novel algorithm (OEATE) against two state-of-the-art parameter estimation approaches in ad-hoc teamwork: AGA and ABU [5] (both presented in Section 2.4). As we mentioned before, we sample sets of parameters (for a gradient ascent step or a *Bayesian* estimation), which is similar to *set of estimators* in the OEATE for estimating parameters and types. Therefore, for better comparison against them, we use the same set size as *estimator* sets (N). Note that [5], also introduced an approach called Exact Global Optimisation (EGO). We do not include it in our experiments since it is significantly slower than the ABU/AGA, without outperforming them in terms of prediction.

Additionally, we compare our approach against the proposed *POMCP-based method* (also presented in Section 2.4) for type and parameter estimations. As described, in estimation with POMCP, we assume that the agent ϕ can see the whole environment, however, the teammates’ type and parameters are not observable. Hence, agent ϕ applies POMCP’s particle filter for estimation. We use $N \times |\Omega| \times |\Theta|$ particles, matching the total number of *estimators* in our approach (since we have N per agent, for each type).

- **Experiments configuration** – We executed random scenarios in Level-based Foraging and Capture the Prey domains (Section 5.9.1 and 5.9.2, respectively) for a different number of distributed tasks, agents and environment size for all aforementioned estimation methods. The experiment finishes by reaching 200 iterations. Every run was repeated 20 times, and we plot the average results and the confidence interval ($\rho = 0.05$). Therefore, when we say that a result is **significant**, we mean statistically significant considering $\rho \leq 0.05$, according to the result of a *Kruskal-Wallis test*. In detail, as a first test, we applied the Kruskal-Wallis to determine whether a statistically significant difference exists between all the

algorithms considered; afterwards, we evaluated each pair of algorithms using a Wilcoxon Rank Sum Test (with Bonferroni correction) to determine which ones were different from the others. Following these steps, we could accurately calculate the confidence interval in the results obtained by each approach, thus finding which one is significantly better than the others. We avoid presenting every p-value to improve the readability of the work. So, we maintain our focus on presenting the p-values that are **meaningful for our analysis** and avoid reporting the p-value for results where there is **clearly no significance** (i.e., $\rho \geq 0.05$). Note that error bars and coloured regions indicate the confidence interval at a 95% confidence level, not the standard deviation, supporting the confidence visualisation.

For each scenario, we assume one of the four estimation methods ABU, AGA, POMCP and OEATE to be agent ϕ 's estimation method. We kept a history of estimated parameters and types for all iterations of each run and calculated the errors by having true parameters and true types in hand. Then, we evaluate the mean absolute error (as in Equation 5.6.2.4) for the parameters, and $1 - P(\theta^*)$ for type; and what we show in the plots is the average error across all parameters. Additionally, since we are aggregating several results, we calculate and plot the average error.

In this way, we first fix the number of possible types as 2 ($L1$, $L2$ and $C1$, $C2$ for Level-based Foraging and Capture the Prey domains, respectively), and later we show the impact of increasing the number of types. Type and parameters of agents in Ω are chosen uniformly randomly. At the Level-based Foraging environment, the skill level for agent ϕ is also randomly selected. Every parameter $p_i \in \mathbf{p}$ is a value within the minimum-maximum interval $[p_i^{min}, p_i^{max}] = [0.5, 1.0]$.

Every task is created in random positions, but we exclude the scenario's borders and free the adjacent tiles. That allows agents to set up their positions to perform the load action from any direction (i.e., North, South, East, West), making it always possible for 4 to simultaneously load an item, which guarantees that all tasks are solvable. For the Capture the Prey environment, this guarantee is not secured since the tasks are moving.

• **Estimation methods configuration** – In our experiments, we used the following configuration for parameters values of OEATE:

- the number of *estimators* N equals to 100;
- the threshold for removing estimators ξ equals to 0.5, and;
- mutation rate m equals to 0.2.
- “information-level” score ($score(e)$) is taken as the number of steps between assigning the *Choose Target state* and completing the task.

We apply the same configuration for all baselines (AGA, ABU and POMCP) and through every experiment performed. For UCT-H [111], we run 100 iterations per time step, and the maximum depth is kept as 100.

5.10.2 Level-based Foraging Results

Before showing the aggregated results, we will first show an example of the parameter and type error estimation across successive iterations. Consider the experiment with $|\Omega| = 7$, a scenario with dimension equals to 30×30 and 30 tasks distributed in the environment. Figure 5.8 shows this result.

As we can see in Figure 5.8 (a), our parameter estimation error is consistently significantly lower than the other algorithms from the second iteration, and it (almost) monotonically decreases as the number of iterations increases. AGA, ABU, and POMCP, on the other hand, do not show any sign of converging to a low error as the number of iterations increases. We can also see that our type estimation quickly overcomes the other algorithms in the mean, becoming significantly better after some iterations, as more and more tasks are completed.

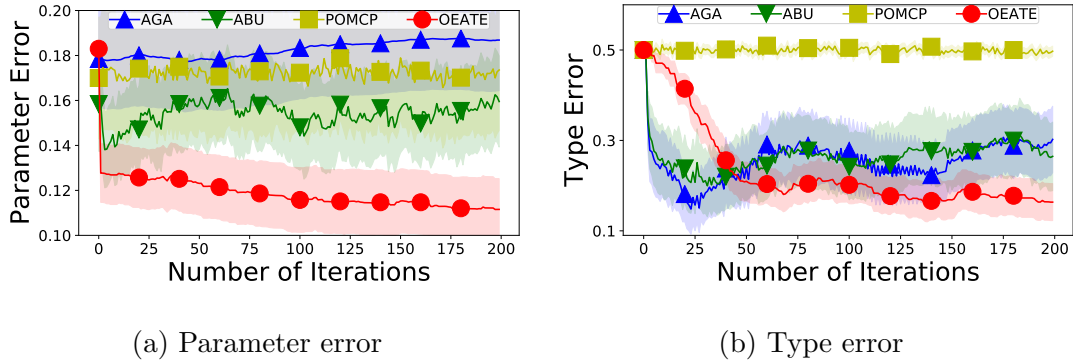


Figure 5.8: Parameter and type estimation errors for $|\Omega| = 7$, dimension 30×30 and $|\mathbf{T}| = 30$

Multiple numbers of items – We now show the results for different numbers of items. Therefore, we fixed the scenario size as 30×30 and the number of agents ω to 7 ($|\Omega| = 7$). Then, we run experiments for a varying number of items in the environment (20, 40, 60, 80). Figure 5.9 shows the result plots.

As we can see in the figure, OEATE has consistently lower error than the other algorithms in terms of parameters estimation. Considering the type estimation, OEATE presents significantly better results for 20, 40 and 80 tasks. We also see that the number of accomplished tasks is very similar, which means that there is no significant difference between the results.

It is interesting to see that our parameter error drops for a very large number of items (80), as OEATE gets a larger number of observations. We can also note that the algorithm scales well to the number of items, and our performance actually *improves* in the mean with more than 20 items. This happens because OEATE gets observations more often for a larger number of items in the environment.

Multiple numbers of agents – After comparing with multiple numbers of items, we run experiments for different numbers of agents. Here, we fix the number of items to 30 and the scenario size to 30×30 . Then, we run experiments for a different number of agents (5, 7, 10, 12, 15) and the plots are shown in Figure 5.10.

Again, for different numbers of agents, OEATE can present a lower or similar

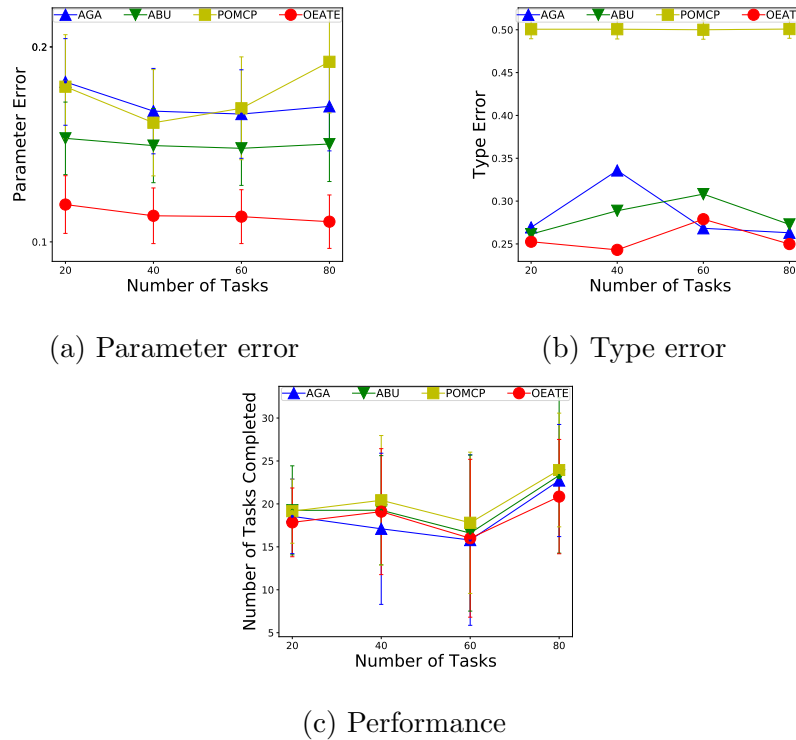


Figure 5.9: Results for a varying number of tasks with full observability.

error than the other algorithms, both in parameter and type estimation. Moreover, we can see that the performance of the team by having a learning agent ϕ (which runs OEATE) is also better than others with the increasing number of teammates. Regarding parameters and type errors, OEATE is significantly better than AGA, ABU and POMCP in almost all cases, except for type error with 15 agents where OEATE is very similar to AGA, respectively. Interestingly, we can see in this case that, even being slightly worse than AGA, OEATE can improve the coordination and complete a higher number of tasks than the baselines. Additionally, the experiment with 15 agents presents the higher difference between the estimation methods performance, where OEATE is clearly the best one.

Multiple scenario sizes – After comparing multiple numbers of items and agents, we run experiments for different scenario sizes to study our scalability to harder problems. For that, we fix the number of items to 30 and the number of ω agents to

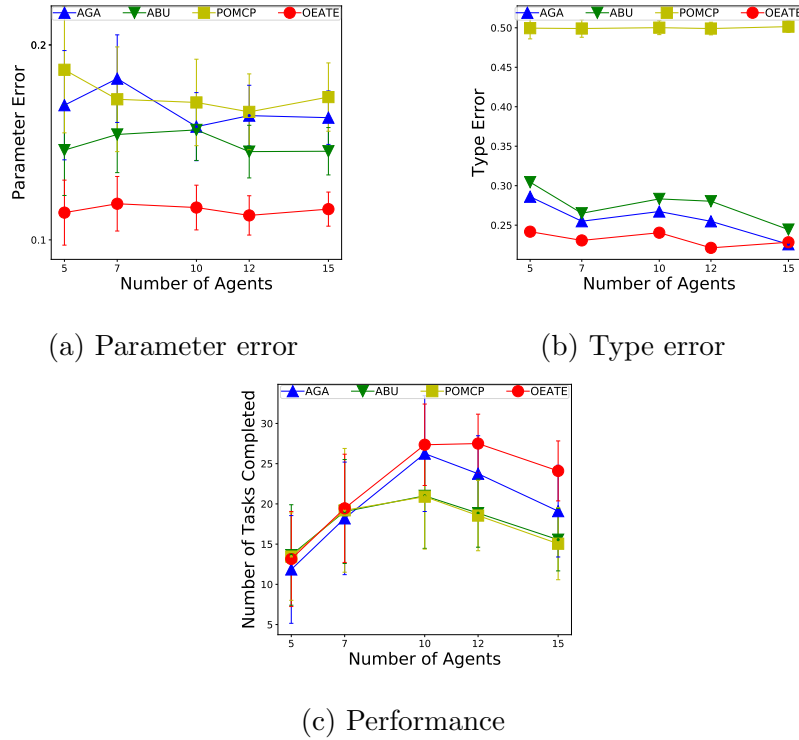


Figure 5.10: Results for a varying number of agents with full observability.

7 ($|\Omega| = 7$). Then, we run experiments for a varying scenario size (20×20 , 25×25 , 30×30 , 35×35 , 45×45) and the plots are shown in Figure 5.11.

As we can see, OEATE has consistently lower error than the other algorithms, both in terms of parameters and type estimation. In fact, OEATE is significantly better than AGA, ABU and POMCP in terms of type and parameters error for all scenario sizes, with $\rho < 0.001$. Additionally, in Figure 5.11 (c) we see that there is no significant difference between task completion of the methods. Overall, OEATE seems to maintain good estimation even with the increasing of scenario dimension.

Partial observability experiment – Here, agent ϕ has partial observability of the environment and employs the POMCP modification for handling that, as described in Section 5.7.2. In these experiments, the number of ω agents is 7 and the environment size is 30×30 , but the variation of items is 20, 40, 60, 80. The radius of ϕ 's view is 15 and the angle is 180° .

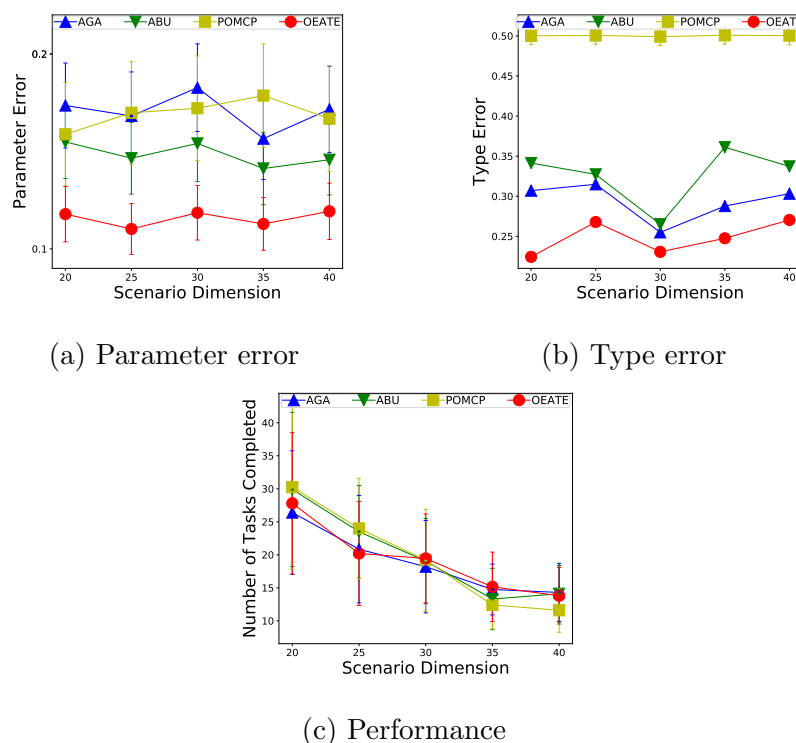


Figure 5.11: Results for various environment sizes in full observability.

Note that AGA/ABU results for partial observability are not shown in Albrecht and Stone (2017) [5], and thus are presented by us for the first time. Hence, in the cases presented here, by OEATE, AGA and ABU, we mean the modified POMCP version, following the approach described in Section 5.7.2; and by POMCP we mean the *POMCP-based estimation*, as before, which does not embed the ad-hoc teamwork algorithms for type and parameter estimation.

We show our results for partially observable scenarios in Figure 5.12. Again, we obtain significantly lower parameter error than previous approaches (Figure 5.12 (a)). In the case of type error (Figure 5.12 (b)), OEATE presents worst type estimation than the competitors, except POMCP. However, they are not significantly better than OEATE. For 20 items, AGA and ABU present $\rho > 0.2$. For 40 and 60, AGA and ABU present $\rho > 0.09$. Finally, for 80 items AGA and ABU present $\rho > 0.35$. In Figure 5.12 (c), we see that we obtain similar performance to the previous approaches in 40 and 60 items.

OEATE represents a task-based solution that depends on the prediction of tasks for unknown teammates for any possible state of the problem. The difficulty in estimating types over partial observability is a result of the lack of precision on reasoning about the part of the map that is not visible. Our proposed modification for POMCP could enable the estimation of parameters and types over partial observability. However, as the problem presents a high level of uncertainty, the belief states need not approximate the actual states of the world, hence OEATE could not perform a good evaluation of its estimators and improve the prediction. Therefore, finding a manner of refining the POMCP belief state approximation can adapt OEATE to handle this new layer of uncertainty that can improve the results as we found for full observability.

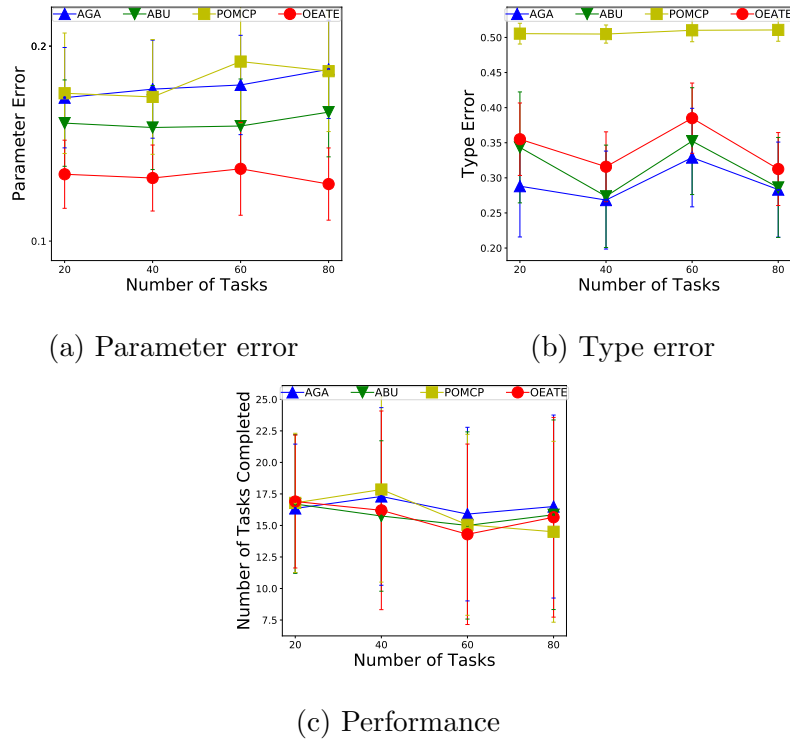


Figure 5.12: Results for a varying number of items in partially observable problems.

Experiments with larger numbers of types – Besides trying to estimate two types ($L1$ and $L2$), we also want to push the uncertainty level of the problem running experiments for a larger number of potential types ($|\Theta|$). In this way, we

run experiments with four types: $L1$, $L2$, $L3$ and $L4$. Figure 5.13 shows the results.

Results displayed in Figure 5.13 (a) demonstrates parameters error, where we are significantly better than all other methods for all number of items with $\rho \leq 0.011$. From Figure 5.13 (b), OEATE outperforms AGA and ABU only with 20 and 60 items in the environment. AGA and ABU are better than OEATE for 40 and 80 items respectively. In the performance, as we can see in Figure 5.13 (c), there is no significant difference between the methods. After studying the four different types case for the ω agents, we experiment with six potential types ($L1$, $L2$, $L3$, $L4$, $L5$, $L6$). The results are shown in Figure 5.14.

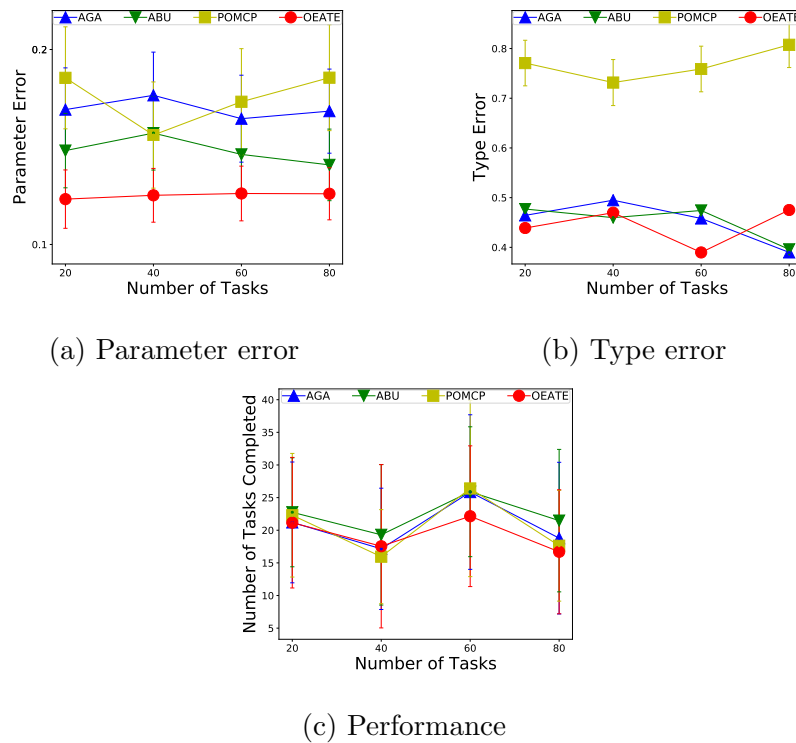


Figure 5.13: Results for a varying number of items, with randomly selected types among 4 types.

Considering parameters error, OEATE is significantly better than the competitors $\rho \leq 0.0005$. Taking type error into account, we are better in all number of items with $\rho \leq 0.06$, except for 40 items, where we are significantly better than POMCP, but against AGA and ABU, we are worst with $\rho \leq 0.92$ and $\rho \leq 0.34$, respectively.

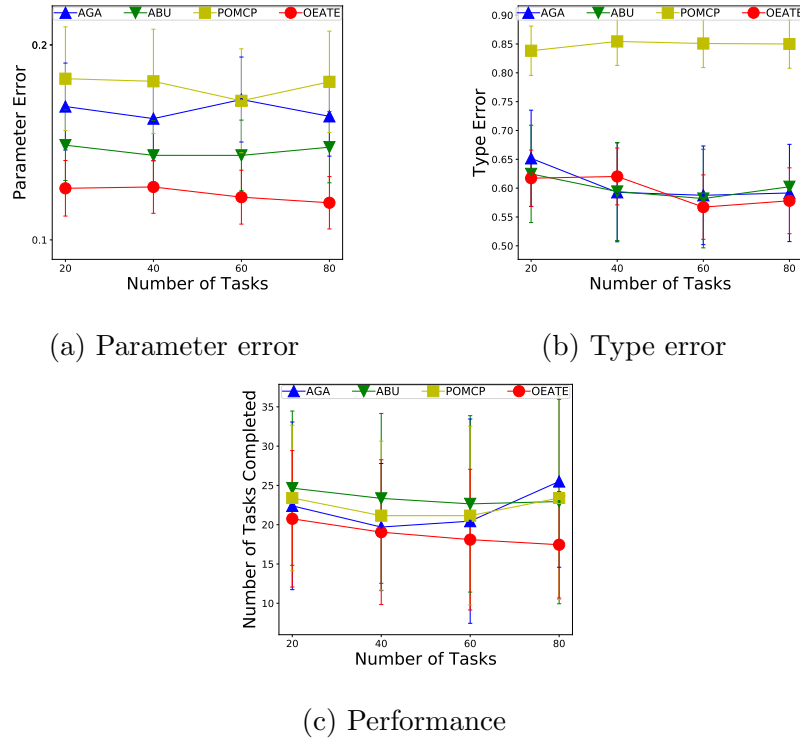


Figure 5.14: Results for a varying number of items, with randomly selected types among 6 types.

For performance, OEATE decreases monotonically as the number of tasks increases.

Overall, OEATE presents a better result performing estimation with fewer types. Its parameter estimation is significantly better for all studied cases. However, when it is facing a higher number of possible templates for types, its type estimation quality decreases and its performance is still similar to the competitors.

Wrong types – We also study our method’s behaviour when the agent ϕ *does not* have full knowledge of the possible types of its teammates. That is, we run experiments where all agents in Ω have a type which *is not* in Θ . In these experiments, we assume that agent ϕ is only aware of type $L1$ and $L2$, but we assign $L3$ and $L4$ to the ω agents as their type (sampled uniformly randomly). We ran experiments with 7 agents and fixed the size of the scenario to 30×30 , with various numbers of items (20, 40, 60, 80). We can see our results for performance in Figure 5.15.

As the figure illustrates, without knowing the potential types for teammates, OEATE only outperform the competitors with 80 items, except POMCP. Surprisingly, POMCP shows the better performance in the group. We believe that, without the knowledge of the possible types and considering the difficulty associated with the problem, acting greedily can show better results in such cases.

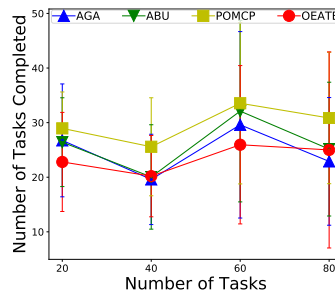


Figure 5.15: Performance of the ad-hoc team for a varying number of items without having information of correct potential teammates types.

5.10.3 Capture the Prey Results

As mentioned before, we run experiments into the Capture the Prey domain. Considering the same settings defined for Level-based Foraging, we define the experiment with $|\Omega| = 7$, a scenario with dimension equals to 30×30 and the set of tasks distributed in the environment (20,40,60,80) as the main result from the set of experiments. Figure 5.16 shows these data plot.

As we can see, OEATE still presenting a significantly lower parameter error in comparison with the competitors. Even though showing worse results compared to AGA and ABU in type estimation, OEATE seems to be able to decrease its error with the increasing number of tasks, while AGA and ABU seem to converge after considering 60 tasks (preys) in the scenario. Additionally, the performance of all methods is very similar in the capture environment.

The defined Capture the Prey domain defines a hard problem to tackle. Improving the team's performance relates to choosing actions that will facilitate the preys

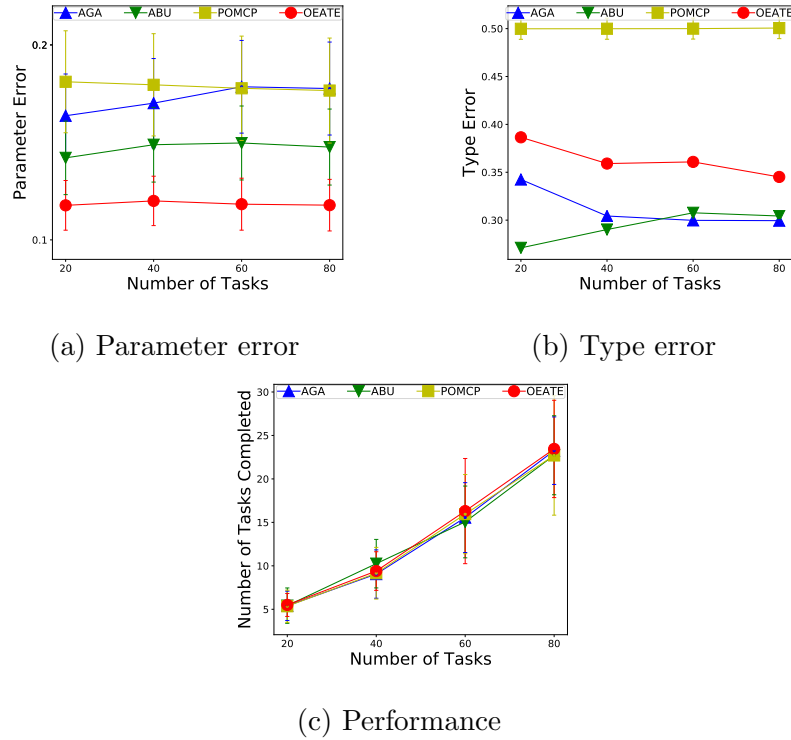


Figure 5.16: Parameter errors, type estimation errors, and performance for a varying number of items in the Capture the Prey domain.

capture. We believe that OEATE can present better results against AGA and ABU over an adaptation of the POMCP for adversarial contexts, where OEATE will be able to reason about the preys, and hence increase the number of tasks accomplished and the type estimation (based on this characteristic).

Overall result – Intending to directly present the conclusions found after performing the complete set of defined experiments and also provide support for further analysis of this research, we present our compiled results of this section regarding the experiments for the Level-based Foraging and Capture the Prey Environments (Table 5.5).

Figure	Experimental Setup	Analysis
Level-Based Foraging Environment		
8	Expected behaviour	OEATE shows an almost monotonic decreasing trend in both type and parameter error. Overall, OEATE could significantly outperform the baselines ($\rho \leq 0.025$ for parameter estimation and $\rho \leq 0.048$) in some of the tested scenarios and this result corroborates to the developed theoretical analysis (Section 5.8).
9	Results for an increasing number of tasks in the environment	OEATE presents a slight improvement on parameter estimation with the increasing number of tasks. On the other hand, there is no observable effect of the increasing number of tasks for type estimation. Overall, considering the defined task-based perspective, we show that OEATE can significantly outperform the baselines parameter and type estimation (both with $\rho \leq 0.002$) for scenarios where <i>key observations</i> (distributed tasks completion) are more often available.
10	Results for an increasing number of agents (teammates) in the environment	The increasing number of agents present no relevant impact for OEATE's parameter estimation. However, OEATE shows better mean performance with the increasing number of teammates . Overall, OEATE can outperform the baselines with the increasing number of teammates

		(with significance of $\rho \leq 0.039$ for some cases) in the environment since the number of tasks completed, in a distributed manner, increases.
11	Results for an increasing environment size	The increasing environment size presents no clear trend for parameter or type estimation . Since the number of distributed tasks in the environment is fixed, the number of tasks completed within the defined time-frame for the experiments (200 iterations) gets lower with the increasing size. Overall, we show that OEATE is still outperforming the baselines ($\rho \leq 0.041$ in parameter and type estimation), regarding type and parameter error, even susceptible to the lower frequency of task completion.
12	Results for partial observability	Over partial observability and considering an increasing number of tasks, OEATE presents significantly better parameter estimation ($\rho \leq 10^{-5}$) than the previous approaches. On the other hand, OEATE shows higher type estimation error (but, with $\rho \geq 0.35$). Overall, even facing the impacts of the partial observable environment, OEATE presents similar performance than the previous approaches.
13, 14	Results for an increasing number of potential types for teammates	The increasing number of potential types for the agents in the environment presents no clear impact in OEATE's parameter and type estimation. On the other hand, OEATE is still outperforming the baselines for most of the cases (considering an $\rho \leq 0.11$). Overall, OEATE

		presents a precise estimation even facing the more complex problem such as this experiment suggests.
15	Results for wrong potential types	OEATE presents a comparable performance to AGA, ABU and POMCP ($\rho > 0.98$). With the lack of a better set of potential, the parameter and type estimation shows no impact on the performance. Overall, we can see that, considering the defined types, the set of potential types impacts directly the final outcome and can must be representative to enable a problem performance improvement.
Capture the Prey Environment		
16	Results of increasing number of preys (tasks)	At the capture the prey environment, there is no clear trend for parameter estimation . On the other hand, the type estimation decreases with the increasing number of preys in the map. Overall, the complexity of this problems leads to a lower number of <i>key observations</i> for OEATE to perform the estimation, which justifies the higher type estimation error , but OEATE is still presenting similar performance and lower parameter estimation ($\rho < 0.0007$).

Table 5.5: Summary of our experiments and results.

Ablation Study – As an interesting piece for the readers, we carried out an ablation study. The intention of this experiment is to show how our internal method choices impact the method outcome. We defined 4 different configurations for the OEATE considering their impact on the quality of the estimation:

- *OEATE*: representing the full version of our proposal;
- *OEATE (No Score)*: representing the version that does not apply the score approach of our final proposal, removing the weighting of decisions made in different choose target states and facing different levels of uncertainty;
- *OEATE (Uniform Scored)*: representing the version that does not perform the process of generating new estimators from the bag. Hence, we removed the bag from our proposal and adapt it to work only with the uniform replacement of estimators, and;
- *OEATE (Uniform)*: representing the version that does not apply the score approach of our final proposal and does not perform the bag generation process, categorising the simplest version of our proposal.

Additionally, considers the experiment with $|\Omega| = 7$, a scenario with dimension equals to 30×30 and 30 tasks distributed in a Level-based Foraging environment (2 types were used in this experiment). Figure 5.17 shows this result.

Regarding the parameter estimation, as the figure shows, we can see that OEATE performs the estimation similarly for all configurations, but the main impact is regarding the starting point of the estimation method. Using each defined strategy leads OEATE, after few iterations performing the estimation process, to correct the parameter values. Differently, from the process carried out by simpler versions of our proposal, OEATE showed to be capable of fixing its estimation in this ablation study. We attribute this improvement to the weighting of estimators during the sampling due to the scoring and bag approach.

On the other hand, the improvement in the results related to the type estimation is even higher. The full version of OEATE presents a significantly better result in comparison with the simpler versions. Interestingly, the second better result found in this ablation study comes from the simplest OEATE configuration. Both, the scored and the uniform scored versions presents higher type error than the uniform one. At this point, we attribute the improvement to the fact that scores of the estimators

help in improving the sampling and maintenance of good estimators in the estimation set. Without recovering estimators from the bag, the scoring can only lead to the trivial game of guessing the correct parameter (hence the type) randomly. Therefore, OEATE represents a fine solution, which combines two unsuccessful tools to obtain a powerful estimation capability.

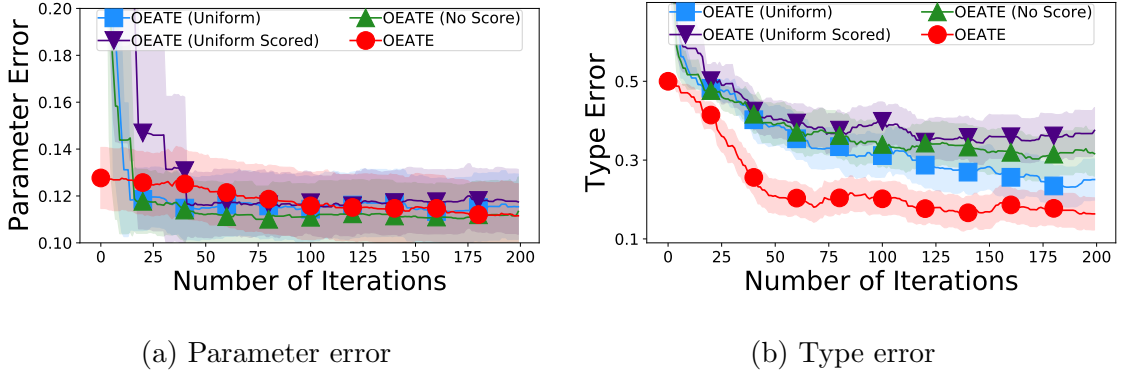


Figure 5.17: Ablation Study for parameter and type estimation errors considering $|\Omega| = 7$, dimension 30×30 and $|\mathbb{T}| = 30$ in the Level-based Foraging domain.

5.11 Chapter Conclusion

In this chapter, we presented *Online Estimators for Ad-hoc Task Execution* (OEATE), a new algorithm for estimating types and parameters of teammates, specifically designed for problems where there is a set of tasks to be completed in a scenario. By focusing on decentralised task execution, we are able to obtain lower error in parameter and type estimation than previous works, which leads to better overall performance. We also studied the convergence of our algorithm theoretically and found empirical results that support our findings. This work opens the path to diverse studies regarding the improvement of ad-hoc teams through a task-based perspective and using an information-oriented approach. We also kindly refer the reader to Chapter 7.2.2 for a more comprehensive discussion about OEATE’s contributions for this thesis’ purposes.

Chapter 6

Bayesian Adversary Estimation

In this chapter, we present the technical details of BAE, our proposed estimation method capable of identifying an adversarial agent (impostor) disguised as a teammate in a cooperative environment. This work was published as a conference paper at AAMAS 2024, titled as “It Is Among Us: Identifying Adversaries in Ad-hoc Domains Using Q-valued Bayesian Estimations” [29].

6.1 Introduction

Ad-hoc Teamwork models represent a relevant tool in MAS for addressing problems with distributed tasks in environments with unknown teammates [13, 100]. For example, it is common to encounter situations in which multiple agents are shipped together to collaborate and quickly resolve a common objective without receiving proper training to perform coordination beforehand [11]. Hence, enabling agents to estimate their teammates’ capabilities might be necessary to guarantee performance.

The advancement of coordination is intrinsically linked to agents’ ability to comprehend their teammates’ strategies and predict their behaviour in the common scenario [83, 2, 48, 26]. Consequently, these studies assume that agents might not adhere to pre-established rules, leading to diverse behaviours inside the team, but all agents are looking to accomplish the same objective [15, 5]. Now, the relevant

question that arises is: *What happens if there is an impostor among us?*

An impostor, in our context, is an intelligent agent acting as an “explicit adversary”. While other agents try to enhance task completion by improving coordination, the impostor endeavours to hinder the team’s performance, disrupting objective achievement by blocking paths or faking its real intention to the team.

Several multi-player games have explored this problem denominated as “*Deduction*” or “*Hidden Roles*” Games [60]. Examples are Among Us and Deceit, which present hypothetical situations where “agents” need to collaborate in order to win the game but there is an “impostor” (adversarial agent) trying to sabotage the team. We can also see the occurrence of similar situations in the real world: (i) in the *robotic manufacturing context*, an adversary robot can slow down or stop an entire production line by making the wrong moves or “being lazy on purpose”; (ii) in *robot rescuing missions*, an impostor may attempt to infiltrate and sabotage the operation by damaging rescue units, and; (iii) in *autonomous vehicles applications*, if a single car gets hijacked, it can throw the traffic system into chaos, besides putting other vehicles and people at risk.

Inspired by this context, some research lines are focusing on developing algorithms capable of performing planning and the deduction of adversarial agents in an online manner [90, 60]. Note that, performing online planning with deduction is crucial in these collaborative MAS contexts because it enables agents to adapt their actions in real time, improving coordination with teammates and reducing the adversarial harm to the overall performance. However, most of them usually fall on hard assumptions to run their algorithms or rely on estimation methods that are expensive to train, besides requiring previous knowledge about the target world and adversary model to guarantee good performance.

Therefore, we present *Bayesian Adversary Estimation* (BAE), a novel algorithm capable of identifying an impostor agent among the team in an online manner by performing estimations solely using the observations collected while acting in the environment and simulating potential outcomes. We propose the application of

what we denominate as the *Q-valued Bayesian Estimation* (QvBE) approach, which considers approximating latent information about the environment using Q-value estimations. In other words, our approximation strategy considers evaluating action values by using information that is not directly available in the observable world. QvBE does it by approximating agents' probability distribution function (pdf) using Q-value estimations. We focus on developing our approach using a Monte-Carlo Tree Search (MCTS) based method; however, we emphasise that it can be extended to any online planning algorithm that estimates Q-values. The main idea behind our proposal is to embed the QvBE approach inside an Adversarial-MCTS (A-MCTS), which performs planning from scratch and estimates its own actions considering the existence of an adversary. The Q-table found at the end of the search process is used to estimate the impostor among the team.

Our approach can run alone and together with different state-of-the-art estimation algorithms to improve coordination while deducting the impostor. Our experiments were performed in the “*Level-based Foraging Environment*”, which is a popular ad-hoc domain to test planning and estimation methods [2, 26]. We demonstrate that our method is capable of improving the accuracy in detecting the impostor agent without significantly penalising the reasoning time or computational resource usage. Across all tested scenarios, we show that BAE can improve impostor identification in comparison with state-of-the-art baselines without relying on prior knowledge about teammates or pre-training data.

6.2 Problem Formalisation and Background

- **Problem Description** We consider the problem where an intelligent (strategic) agent ϕ is in a teamwork context, collaborating with a set of non-strategic agents $\omega \in \Omega$ but one of these non-strategic agents is an impostor ψ , i.e., a strategic agent trying to disturb the accomplishment of the objectives by the team. Note that a

strategic agent is an agent that runs an algorithm for planning and is capable of modelling other agents in its reasoning process, while non-strategic agents do not model other agents in their reasoning process [107]. The main goal of the team is to accomplish a common and shared objective in the environment. Hence, ϕ is the ad-hoc agent that is trying to maximise the performance of the team and it must figure out which agent is ψ to improve coordination.

The model used by the ad-hoc agent ϕ considers that there are $|\Omega|$ agents, apart from itself, which assumes $|\Omega \setminus \psi|$ agents as non-strategic agents and ψ as an intelligent (strategic) agent. On the other hand, the impostor ψ 's model also considers that there are $|\phi \cup \Omega \setminus \psi|$ agents (apart from itself), but all of them are non-strategic agents, including ϕ . In summary, ϕ runs an adversarial reasoning method, considering the optimisation of its own actions and the estimation of the impostor actions to maximise the overall performance, while ψ runs a direct minimisation algorithm, optimising its own action to reduce the overall performance.

- **Formal Model** In this study, we focus on the application and implementation of Markovian-based models together with RL algorithms to handle the stated problem. Hence, we describe it as a Multi-agent Markov Decision Process (MMDP) [19], with $M = |\phi \cup \Omega|$ agents sharing the same environment and comprising the team Λ . The MMDP model contains a finite set of states $s \in \mathbf{S}$ with transition probability \mathcal{T} and expected reward equal to $\mathcal{R}(s, \mathbf{J})$ depending on the joint-actions of all agents $\mathbf{J} = \{a_1, \dots, a_M\}$, where each action is defined in the action space $a_i \in \mathbf{A}$. Therefore, given an MMDP, we want to estimate the actions that maximise the expected reward that all agents will receive as the system progresses through time.

- **Planning Algorithm** We use the defined MMDP (Chapter 2.1.4) to implement our proposed planning algorithm from the family of the MCTS algorithms. We choose this state-of-the-art method considering its capabilities of performing online planning and estimation, besides running it from scratch at every execution as presented in

other works in the literature [2, 91]. However, we highlight here that our solution can be extended to any online planning algorithm capable of estimating Q-values.

The MCTS algorithm aims to find the optimal action a^* for a given state s and agent by simulating the world steps within a tree structure. In order to perform MCTS planning, the literature suggests the application of UCT or UCT-H algorithms [58, 91]. In our model, each node in the Monte-Carlo tree \mathbf{T} is represented by $(s, \mathcal{V}, \mathcal{N}, \phi)$, i.e., a tuple with a *state* s , a *value* $\mathcal{V}(s, a)$, a *visitation count* $\mathcal{N}(s, a)$ for each action $a \in \mathbf{A}$ and the agent ϕ which will define the perspective of simulation within the tree. We define this agent perspective ϕ because it enables us to run a single-agent MDP model while performing the tree search procedure, hence, the MCTS will estimate values and simulate the actions and world transitions considering ϕ as the ad-hoc agent and the other agents $\omega \in \Omega$ as part of the environment. The *value* of the node represents the expected cumulative reward for the simulated states. The number of visits to the state s is represented by $\mathcal{N}(s) = \sum_{a \in \mathbf{A}} \mathcal{N}(s, a)$.

In the adversarial reasoning case, the node in the MCTS will be represented by the tuple $(s, \mathcal{V}, \mathcal{N}, \phi, \psi)$ where the ψ agent will define the impostor perspective in the min-max MCTS process – with ϕ being the max and ψ the min part of the tree simulation. The tree represented by this tuple will be an adversarial tree \mathbf{T}_ψ and we denominate this MCTS process as an Adversarial-MCTS (A-MCTS). The actual difference between the MCTS and A-MCTS approaches is highlighted in the simulation process presented below.

• **Simulations** While performing simulations within an MCTS, each state in the search tree is viewed as a multi-armed bandit taking actions chosen by the Upper Confidence Bound (UCB1) algorithm. In a traditional maximisation problem, *UCB1* tries to increase the value of less-explored actions by attaching a bonus inversely proportional to the number of times each action is tried, following $UCB1(s, a) := \mathcal{V}(s, a) + c\sqrt{\frac{\log(\mathcal{N}(s))}{\mathcal{N}(s, a)}}$. The constant scalar c is the exploration constant, which is responsible for weighting the exploration value $\sqrt{\frac{\log(\mathcal{N}(s))}{\mathcal{N}(s, a)}}$. We can fit this constant

to the target problem by considering the desired balance, exploiting close and future rewards. Analogously, while solving a minimisation problem, the UCB1 function considers the subtraction of the exploration value $UCB1(s, a) := \mathcal{V}(s, a) - c\sqrt{\frac{\log(\mathcal{N}(s))}{\mathcal{N}(s, a)}}$, in order to correctly balance the exploration and exploitation levels within the tree.

Now, algorithm-wise, the simulation process is categorised by the expansion of the tree considering possible paths or succession of nodes that a sequence of actions can lead to in the problem's world. In this case, the simulation only considers the perspective of the ad-hoc agent ϕ , simulating its actions, and the other agents Ω as part of the environment. Hence, a path in the tree is the succession of actions a_t taken by ϕ that aims to find the best sequence $\{a_0, a_1, \dots, a_D\}$ that maximises or minimises its reward collection.

On the other hand, the A-MCTS simulation process is also categorised by the expansion of the tree considering the possible paths in the world. However, it considers the sequence of actions taken by ϕ and ψ to find the best path that maximises or minimises reward collection. Therefore, the path in the tree is the succession of action pairs $\{a_0^\phi a_0^\psi, a_1^\phi a_1^\psi, \dots, a_D^\phi a_D^\psi\}$ taken by ϕ and ψ . Note that this characteristic leads the tree to present an architecture that always alternates between an ϕ node to an ψ node and vice-versa.

- **Estimation Strategy** Bayesian inference is a powerful tool for updating the probability of a hypothesis when more evidence or information becomes available. Mathematically, it is represented by the equation $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, where A is the hypothesis and B is the observation or evidence from the model. Hence, $P(A)$ is the *prior probability*, $P(B)$ the *marginal likelihood*, $P(B|A)$ the *likelihood* of observing B given A , and $P(A|B)$ our *posterior probability*. In this thesis, we will develop this Bayesian application to estimate the probability of an agent being an adversary (hypothesis), given the action performed by it in the real world (evidence). We precisely adapt the tree search process together with the idea of roles in a MARL system to estimate teammates and adversaries.

6.3 Online planning using Q-valued Bayesian Estimations

In this section, we present **B**ayesian **A**dversary **E**stimation (BAE), a novel lightweight estimation method capable of identifying adversarial agents in an online manner. BAE performs planning in the context we denote as *Ad-hoc Reasoning context*, which is described by ad-hoc teamwork problems with adversaries. In this section, we present our methodology behind BAE, explaining its implementation step by step as we discuss the main contributions.

6.3.1 Initialisation

From the perspective of our intelligent agent ϕ , we initialise the probability of being an adversary for every agent $\omega \in \Omega$ sharing the environment with ϕ . By default, we set a uniform distribution \mathcal{U}_Ω as the initial probability distribution across agents Ω . Therefore, BAE assumes an equal probability for all agents to be the impostor among the team equal to $P(\omega = \psi) = \frac{1}{|\Omega|}, \forall \omega \in \Omega$. If there is any additional information about the target problem, it is possible to adjust this initial distribution to the knowledge available accordingly. Note that the described step defines the probability distribution function (pdf) that will be used for sampling potential adversaries at each traversal in the tree.

• **Initialisation Example** Let's assume that ϕ is playing together with three other agents, i.e. $|\Omega| = 3$. ϕ knows there is an impostor in the team, but has no hint about who it is. Therefore, the ad-hoc agent ϕ initialises its tree search structure and the probability of each agent $\omega \in \Omega$ to be the impostor following the uniform distribution \mathcal{U}_Ω . Figure 6.1 illustrates this initialisation process.

After initialisation, we begin the search process with the A-MCTS, where ϕ reasons about its best action considering different potential impostor agents ψ in the environment.

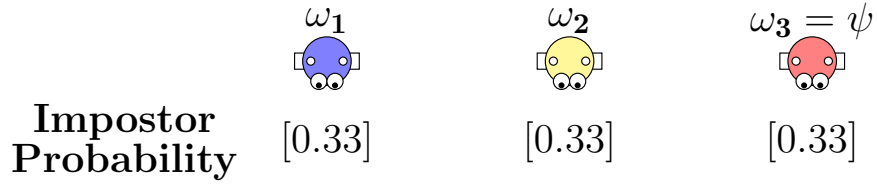


Figure 6.1: BAE’s initialisation process in an environment with 3 other agents where one is an impostor and two are non-strategic teammates.

6.3.2 Search and Q-table Extraction

The search process proposed here follows the same high-level procedure defined in the literature to run the A-MCTS approach and estimate Q-values for each action of the ad-hoc and adversarial agents. In general, we can describe the search process as the expansion of a tree structure that ends with the back-propagation of the found values through traversal in the tree. Figure 6.2 depicts the effect of this process.

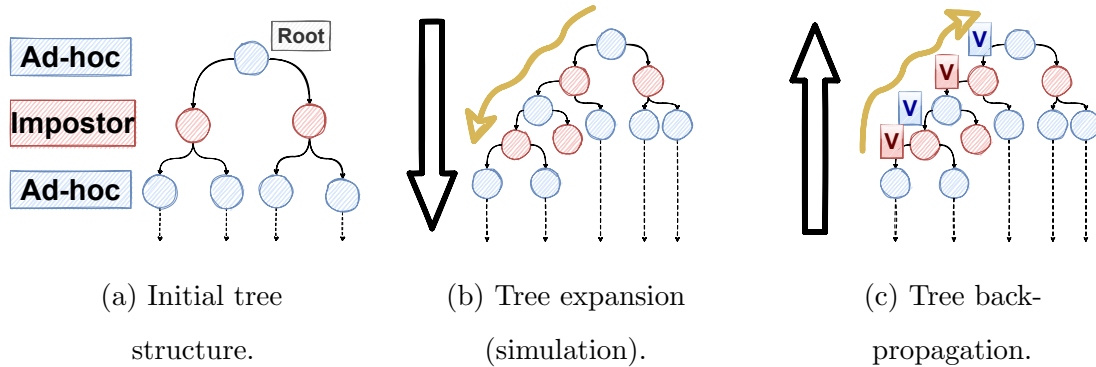


Figure 6.2: A high-level view of the tree search process. The arrows with dotted lines represent the existence of further paths in the tree not included in the illustration.

Whenever we initiate a traversal within the tree or a simulation process (Figure 6.2a), our objective is to expand the initial tree structure, thereby identifying new promising paths within the tree that solve our problem (Figure 6.2b). Each path or branch within the tree corresponds to a specific sequence of actions that our agent simulates to predict a possible outcome. Every time we finish a simulation

on the tree, we update the value of all nodes encountered along the path through the process denominated as back-propagation (Figure 6.2c). These estimated and back-propagated values determine the best action for the ad-hoc agent considering the existence of the potential adversary in the environment.

When running a *single-adversary* A-MCTS search process with a *previously known* adversary, it considers the perspective of an ad-hoc ϕ and an adversarial agent ψ to perform the simulations within the tree. In our case, when running a *single-adversary* A-MCTS search process with *different potential adversaries* in the environment, we propose calculating the expected Q-values via sampling within the tree. We refer to this approach as “Expected Q-value MCTS” (EQ-MCTS) and consider that an agent ω is sampled to be simulated as an impostor every time we start a simulation procedure. We consider the current estimated adversarial pdf across agents to weigh impostor sampling.

Since the simulation procedure is performed multiple times, the result of successive simulations, considering each sampled adversary at each traversal in the tree, will estimate the Q-values for each action in the adversary nodes that represent the *expected value for all potential impostors* across all actions. Because our MMDP considers the perspective of an impostor agent ψ , we also consider spatial features while simulating actions and transitioning between states, i.e., all agents’ positions and actions in the environment. At the end of the search process, we can decide the best action for the ad-hoc agent to take in the real world and hence extract the Q-table for the adversary after stepping into the adversarial node that succeeds the best action. The below example intends to clarify and illustrate the adversary Q-table extraction process.

• **Q-table Extraction Example** Let us consider the tree shown in Figure 6.2a. Each time we perform the expansion (Figure 6.2b), we consider the perspective of ϕ to take an action in the blue nodes, and the perspective of a sampled agent ψ as our adversary taking actions in the red nodes. Therefore, note that ϕ is fixed through

the search, and ψ is resampled every time for each simulation.

Once we find a stop condition in the simulation (e.g., reach the maximum depth of the tree, or find a terminal state), we back-propagate all rewards found through the traversal in the tree to the root node (Figure 6.2c). These rewards update each node’s Q-table values. From the root, we can select the best action for our ad-hoc agent ϕ , and from the next node stepping on the tree, considering ϕ ’s best action, we can extract the adversary Q-table.

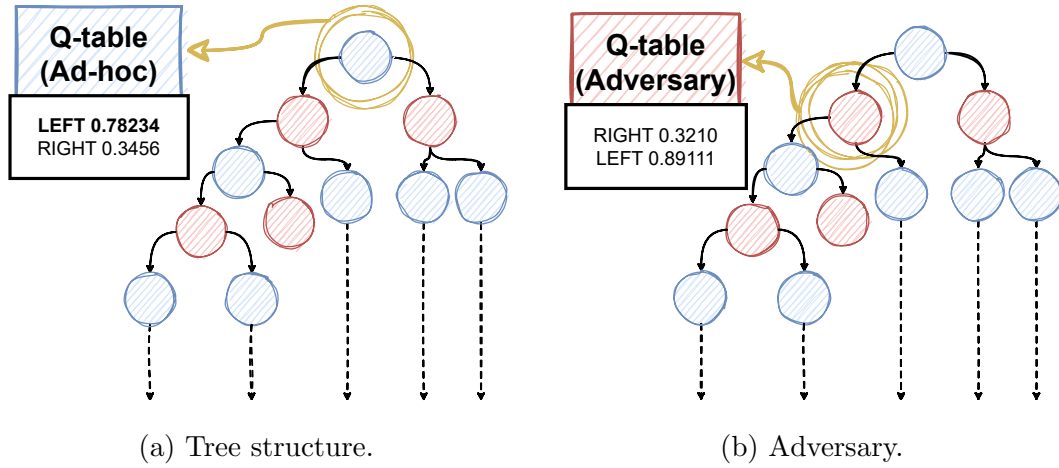


Figure 6.3: The selection of the best action and the extraction of the Q-table for the ad-hoc ϕ and the adversary ψ agents.

Figure 6.3 demonstrates this situation, highlighting the node from which the Q-table is extracted. From the Q-table of the ad-hoc agent (Figure 6.3a), we can pick the “LEFT” action as the best one for our ϕ agent, since it maximises the value of the game. Stepping to the left node, we find the adversary node from which we extract the impostor’s Q-table and the best action to minimise rewards (i.e., action “RIGHT”). Note that we will step into the right node to update the tree because we cannot confirm the real action of the adversary (since we do not know the true adversary) but we estimated that the “RIGHT” action is the best option to minimise the overall value.

6.3.3 Q-table Translation and our Bayesian Update

After performing the Q-table extraction, we translate the final Q-table values from the adversary’s perspective (i.e., the adversarial “root” in the tree search process) into probabilities to be used in a Bayesian update process. As previously mentioned, we consider the probability of an agent being an adversary as our hypothesis, and the action performed by it in the real world a_ω as our evidence in the inference process. Therefore, we can rewrite the Bayesian equation as such:

$$P(\omega = \psi|a_\omega) = \frac{P(a_\omega|\omega = \psi)P(\omega = \psi)}{P(a_\omega)} \quad (6.1)$$

The prior probability $P(\omega = \psi)$ in the equation is the initialised value at the very first iteration, which will be replaced by the updated value for the next iteration. On the other hand, our likelihood $P(a_\omega|\omega = \psi)$ is the normalised value of the estimated Q-value for the action a_ω extracted from the final Q-table from the search process, following the equation:

$$P(a_\omega|\omega = \psi) = \frac{r_{max} - Q(a_\omega)}{\sum_{a' \in \mathbf{A}} (r_{max} - Q(a'))} \quad (6.2)$$

Note that we consider the difference between r_{max} and $Q(a_\omega)$, where r_{max} is the maximum reward value threshold possible for the problem. We use it because we want lower-valued actions to increase the probability of agents being adversaries since the impostor’s behaviour wants to minimise the team’s performance. For simplicity, we normalise all the results *a posteriori* across the probabilities which will sum up to 1. Therefore, we will only consider the upper part of Equation 6.1, i.e., considering $P(a_\omega) \propto 1$.

• **Q-table Translation and Bayesian Update Example** With the adversary Q-tables in hand, we now transform its values into probabilities, i.e., the final estimated value will be translated into the probability of an adversary taking the action using the $P(a_\omega|\omega = \psi)$ equation. Let us consider $r_{max} = 1$ and the Q-tables presented in Figure 6.3. We can update the impostor probabilities following:

$$P(a_\omega | \omega = \psi) = \frac{1 - Q(a_\omega)}{\sum_{a' \in \mathbf{A}} 1 - Q(a')}$$

Therefore, we have that $\sum_{a' \in \mathbf{A}} 1 - Q(a') = 0.7879$, therefore $P(a = R | \omega = \psi) = \frac{1 - 0.3210}{0.7879} \approx 0.86$ and $P(a = L | \omega = \psi) = \frac{1 - 0.89111}{(0.3210 + 0.89111)} \approx 0.14$. Following the rationale and performing this procedure, we can already apply the result in Equation 6.1 and update the probability of each agent being the impostor in our problem. Let us assume the following actions were observed for each agent in the environment: $a_{\omega_1} = \text{“L”}$, $a_{\omega_2} = \text{“L”}$, and $a_{\omega_3} = \text{“R”}$. The result of our Bayesian Update follows the steps shown in Figure 6.4. After successive iterations using this update approach, BAE correctly estimates the impostor among the team without penalising the execution time or relying on extra resources.




	ω_1 	ω_2 	$\omega_3 = \psi$ 
Probability Prior Impostor	[0.33]	[0.33]	[0.33]
Bayesian Update	[0.14 * 0.33]	[0.14 * 0.33]	[0.86 * 0.33]
Updated Result	[0.05]	[0.05]	[0.28]
Normalised Result	[0.13]	[0.13]	[0.74]

Figure 6.4: An illustration of BAE’s update.

6.3.4 Algorithm Outline

After presenting our complete methodology and each step of BAE, Algorithm 12 provides the pseudo-code for the implementation of our proposal.

Algorithm 12 Bayesian Adversary Estimation. \mathbf{P}_ψ is the vector of probabilities $P(\omega_i = \psi) = p_\psi^{\omega_i}$; Q_ϕ is the Q-table for the ad-hoc agent; Q_ψ is the Q-table for the adversary agent, and; $\mathbf{P}_{\omega,\psi}$ is the vector of probabilities $P(a_i|\omega = \psi) = p_{\omega,\psi}^{a_i}$.

<pre> 1: procedure BAE(\mathbf{P}_ω^ψ) 2: <i># 1. Initialising adversary's probabilities</i> 3: if $\mathbf{P}_\omega^\psi = \emptyset$ then 4: $\mathbf{P}_\omega^\psi \leftarrow$ Initialisation(Ω) 5: <i># 2. Performing the search process</i> 6: <i># and extracting the Q-tables</i> 7: $a_{best}, Q_\phi, Q_\psi \leftarrow$ Search(ϕ) \triangleright <i>A-MCTS</i> 8: <i># 3. Translating the ψ Q-values</i> 9: <i># into probabilities</i> 10: $\mathbf{P}_a^\psi \leftarrow$ QtableTranslation(Q_ψ) 11: <i># 4. Updating the adversary's</i> 12: <i># probabilities</i> 13: $\mathbf{P}_\omega^\psi \leftarrow$ Update($\mathbf{P}_\omega^\psi, \mathbf{P}_a^\psi$) 14: <i># 5. Returning planning and estimation</i> 15: <i># results</i> 16: return $a_{best}, \mathbf{P}_\omega^\psi$ </pre>	<pre> 1: procedure QTABLETRANSLATION(Q_ψ) 2: <i># 1. Transforming Q-values into</i> 3: <i># probabilities</i> 4: $M \leftarrow \mathbf{A} , \mathbf{P}_a^\psi = [p_{a_0}^\psi, p_{a_1}^\psi, \dots, p_{a_{M-1}}^\psi]$ 5: for $a_i \in \mathbf{A}$ do 6: $p_{a_i}^\psi \leftarrow \frac{r_{max} - Q(a_i)}{[\sum_{a' \in \mathbf{A}} r_{max} - Q(a')]} \triangleright$ <i>Eq. 6.2</i> 7: 8: return \mathbf{P}_a^ψ </pre>
<pre> 1: procedure INITIALISATION(Ω) 2: <i># 1. Uniform initialisation</i> 3: $N \leftarrow \Omega , \mathbf{P}_{init} \leftarrow [p_{\omega_0}^\psi, p_{\omega_1}^\psi, \dots, p_{\omega_{N-1}}^\psi]$ 4: for $\omega_i \in \Omega$ do 5: $p_{\omega_i}^\psi \leftarrow \frac{1}{N}$ 6: return \mathbf{P}_{init} </pre>	<pre> 1: procedure UPDATE($\mathbf{P}_\omega^\psi, \mathbf{P}_a^\psi$) 2: <i># 1. Updating adversary probabilities</i> 3: for $\omega_i \in \Omega$ do 4: $p_{prior} = p_{\omega_i}^\psi$ 5: $p_{likelihood} = p_{a_{\omega_i}}^\psi$ 6: 7: $p_{\omega_i}^\psi \leftarrow p_{likelihood} * p_{prior} \triangleright$ <i>Eq. 6.1</i> 8: 9: <i># 2. Normalising the result</i> 10: $\beta \leftarrow \sum_{i=0}^M p_{\omega_i}^\psi$ 11: for $\omega_i \in \Omega$ do 12: $p_{\omega_i}^\psi \leftarrow p_{\omega_i}^\psi / \beta$ 13: 14: return \mathbf{P}_ω^ψ </pre>

6.4 Evaluation Settings

• **Simulation Environment** – We performed experiments and collected data in the *Level-Based Foraging* (LBF) environment [2, 26]. The environment and all benchmark scenarios were implemented in *AdLeap-MAS* [25] (Appendix A), an open-source multi-agent simulator for ad-hoc reasoning. Our code can be found publically available on GitHub¹.

• **Benchmarks** – The LBF environment is commonly used in the literature to test online planning algorithms and estimation methods [2, 26]. In this scenario, agents are placed in a 2D grid world, where they can navigate and attempt to collect boxes distributed in the environment. However, the collection of a box is successful only if the sum of the levels of the agents involved in loading it is equal to or higher than the box’s weight (or level). In addition to the level, each agent has a certain vision radius and angle as parameters. Considering this ad-hoc environment, agents must estimate their teammates’ parameters to improve performance. In this study, we disguised an adversarial agent among the team which attempts to minimise performance (by disturbing the box collection). We defined 4 different scenarios as benchmarks:

(LBF.a) The first one is a “small scenario” (Figure 6.5). It is a 5×5 environment where the ad-hoc agent ϕ must estimate between two agents which one is the impostor (a total of 3 agents), and there is only one *individualistic task* available for accomplishment in the environment. An individualistic task is a task that any agent can accomplish without cooperating with others.

Each agent has a type and parameters associated with it, which follows the information presented in Table 6.1. Each task is also associated with a weight, which follows the information presented in Table 6.2.

¹BAE’s GitHub repository: <https://github.com/lsmcolab/bae-adversary-detection>

(LBF.b) Our second scenario is a “medium scenario” (Figure 6.6) that extends our first scenario by *increasing* the *dimensions* of the environment from 5×5 to 9×9 , and the number of *tasks* from 1 to 5. We kept the total number of agents at 3.

Each agent has a type and parameters associated with it, which follows the information presented in Table 6.3. Each task is also associated with a weight, which follows the information presented in Table 6.4.

(LBF.c) Our third scenario, in Figure 6.7, develops our second scenario by *increasing* the number of *agents* from 3 to 5, adding 2 new non-strategic agents as potential impostors. We denominate this scenario setting as the “big scenario”.

Each agent has a type and parameters associated with it, which follows the information presented in Table 6.5. Each task is also associated with a weight, which follows the information presented in Table 6.6.

(LBF.d) Our last scenario uses the big scenario configuration (Figure 6.7); however, we extend this benchmark by *requiring agents to cooperate* between themselves to accomplish some tasks. Hence, tasks are mostly cooperative, rather than individualistic, in contrast with the LBF.c scenario.

Each agent has a type and parameters associated with it, which follows the information presented in Table 6.7. Each task is also associated with a weight, which follows the information presented in Table 6.8.

Table 6.1: Agents’ details in the small scenario (LBF.a).

Agent	True	Parameters
Index (color)	Type	[Radius, Angle,Level]
A (red)	ϕ	[1.0, 1.0, 1.0]
1 (grey)	$l1$	[1.0, 1.0, 1.0]
X (blue)	ψ	[1.0, 1.0, 1.0]

Table 6.2: Task' details in the small scenario (LBF.a).

Task Position	(5, 5)
Weight	1.0

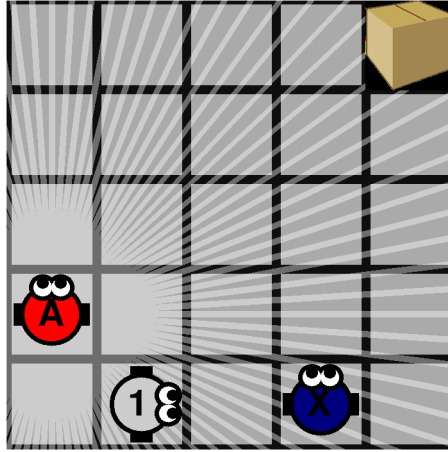


Figure 6.5: Small scenario spatial configuration (LBF.a).

Table 6.3: Agents' details in the medium scenario (LBF.b).

Agent	True	Parameters
Index (color)	Type	[Radius, Angle, Level]
A (red)	ϕ	[1.0, 1.0, 1.0]
1 (grey)	$l1$	[1.0, 1.0, 1.0]
X (blue)	ψ	[1.0, 1.0, 1.0]

Table 6.4: Tasks' details in the medium scenario (LBF.b).

Task Position	(3, 3)	(7, 7)	(7, 3)	(3, 7)	(5, 5)
Weight	0.5	0.5	0.5	0.5	0.5

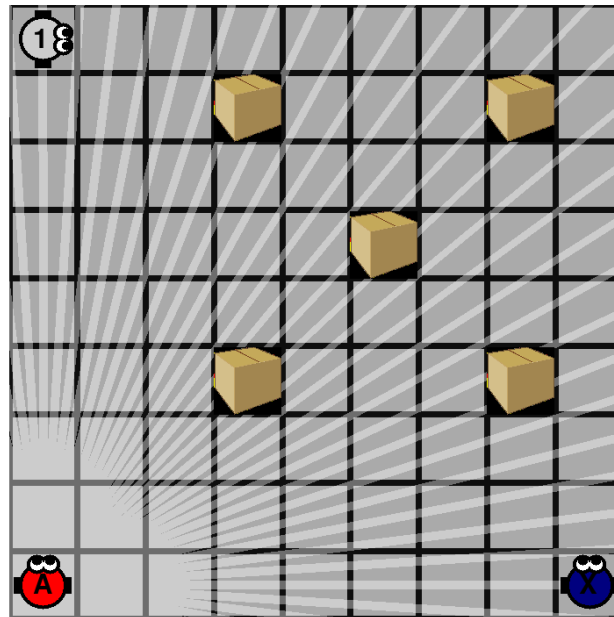


Figure 6.6: Medium scenario spatial configuration (LBF.b).

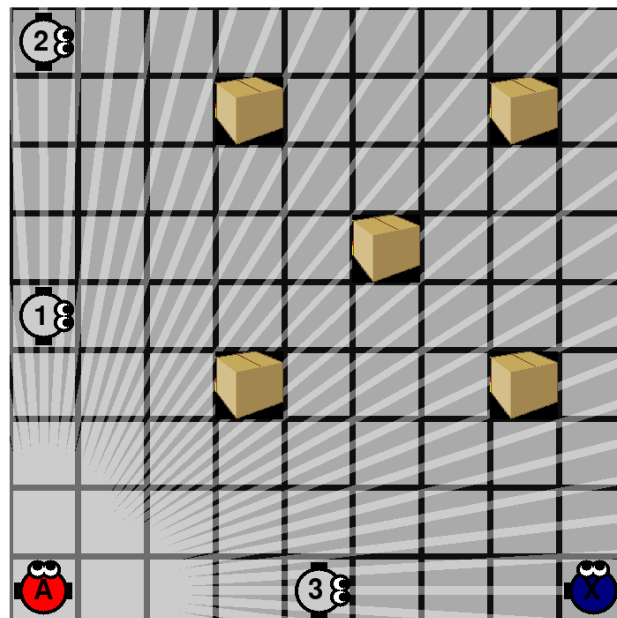


Figure 6.7: Spatial configuration for the big scenario (LBF.c) and the cooperative scenario (LBF.d).

Table 6.5: Agents' details in the big scenario (LBF.c).

Agent	True	Parameters
Index (color)	Type	[Radius, Angle,Level]
A (red)	ϕ	[1.0, 1.0, 1.0]
1 (grey)	$l1$	[1.0, 1.0, 1.0]
2 (grey)	$l2$	[1.0, 1.0, 1.0]
3 (grey)	$l3$	[1.0, 1.0, 1.0]
X (blue)	ψ	[1.0, 1.0, 1.0]

Table 6.6: Tasks' details in the big scenario (LBF.c).

Task Position	(3, 3)	(7, 7)	(7, 3)	(3, 7)	(5, 5)
Weight	0.5	0.5	0.5	0.5	0.5

Table 6.7: Agents' details in the cooperative scenario (LBF.d).

Agent	True	Parameters
Index (color)	Type	[Radius, Angle,Level]
A (red)	ϕ	[1.0, 1.0, 0.1]
1 (grey)	$l1$	[1.0, 1.0, 0.3]
2 (grey)	$l2$	[1.0, 1.0, 0.4]
3 (grey)	$l3$	[1.0, 1.0, 0.5]
X (blue)	ψ	[1.0, 1.0, 0.6]

Table 6.8: Tasks' details in the cooperative scenario (LBF.d).

Task Position	(3, 3)	(7, 7)	(7, 3)	(3, 7)	(5, 5)
Weight	0.5	0.5	0.5	0.5	0.5

- **Experimental Setup** We propose two different setups to test the capabilities of BAE and our defined baselines.

- **Adversarial Detection:** In this setup, we focus on evaluating the capability for detecting the true adversary, i.e., identifying the impostor agent among the team. To do so, we consider that the ad-hoc agent has full knowledge of their teammates' types and parameter values.

- **Ad-hoc Adversarial Detection:** In this setup, we focus on evaluating the impact of running ad-hoc teamwork algorithms at the same time we run our adversarial detection approach, BAE. Therefore, we consider that the ad-hoc agent has no knowledge of its teammates' types or their parameter values.

- **Baselines** We propose the following three methods as our baselines for Adversarial Detection, divided into two groups:

- **Type-based methods** – which are **AGA** and **ABU** [2], estimation methods based on gradient ascent and bayesian updates, respectively.

- **Adversarial detection method** – **OEATA-A** [90] an ad-hoc teamwork algorithm capable of running the OEATA estimation algorithm together with the detection of adversaries.

We do not perform the analysis and experiments using purely OEATA or OEATE because they represent task-oriented solutions and, since the impostor agent in our experimental settings does not accomplish tasks (aiming to decrease the performance of the team), cannot perform the estimation of the adversarial agent. We refer the reader to the Related Work (Chapter 3) for further details regarding each baseline presented here. Our baseline implementations can be found on GitHub.

As aforementioned, we want to test the impact of running an ad-hoc teamwork algorithm at the same time we run BAE's adversarial detection in the Ad-hoc Adversarial Detection setting. Therefore, we propose two new estimation methods for this analysis:

- **AGA-BAE** and **ABU-BAE** are adaptations that consider the embedding of our proposed method into AGA and ABU algorithms, respectively. They use ad-hoc teamwork methods to perform type and parameter estimations while applying BAE’s detection strategy to identify adversaries in the environment.
- **Adversarial Detection using Type-based baselines** To enable a fair comparison between BAE and the type-based approaches, we generated a simple adversarial behavioural template, denoted $\tilde{\psi}$, and added it to the knowledge of these methods. Under this experiment configuration, the type-based agent will approximate the probability of an agent being the adversary across N templates (which matches the number of teammates in the environment), where $N - 1$ templates are non-strategic templates and one is the created impostor template. After performing the estimation across templates, we normalise the probability of being the adversary across agents and then use it as the adversarial detection metric.

Table 6.9 presents the real and template types used in each scenario to run the type-based methods, AGA and ABU.

Table 6.9: Real and template types per scenario for the Adversarial Detection experimental setup. Note that each type in the “Real Types” column refers to the true type of each agent in the scenario, and all types in the “Template Types” column will be used to approximate each agent’s behaviour.

	N agents	Real Types	Template Types
Small scenario	2	$[l1, \psi]$	$[l6, \tilde{\psi}]$
Normal scenario	2	$[l3, \psi]$	$[l6, \tilde{\psi}]$
Individualistic	4	$[l1, l2, l3, \psi]$	$[l4, l5, l6, \tilde{\psi}]$
Cooperative	4	$[l1, l2, l3, \psi]$	$[l4, l5, l6, \tilde{\psi}]$

We refer the reader to Chapter 5.9.1 for further details about the types used as templates, which are proposed together with OEATE.

- **Impostor Template** We created the impostor template $\tilde{\psi}$ as a simple Q-learning adversarial model, where, for each benchmark scenario, we ran 200 episodes and saved the estimated Q-values for each state. We ran the same minimisation MCTS implemented to run the true impostor ψ . For each search process, we performed 1000 iterations of simulations, each with a maximum depth of 25.

Whilst simulating an agent running $\tilde{\psi}$, we sample the action, given a state s , by first transforming the estimated Q-values into probabilities and sampling a random action from this distribution.

- **Metrics and Analysis** We use the following metrics:

- *Average impostor probability* $P(\omega_\psi = \psi)$, which is the average estimated probability of the true adversary being the adversary.
- *Average planning time (t)*, which is the average time spent by the ad-hoc agent to plan its actions and perform estimations.

Mean results are calculated across *50 executions*. Every experiment runs independently, so no knowledge is carried from one execution to another. The calculated errors ($\pm Err$) represent the confidence interval of a two-sample t-test with 99% of confidence; we label a result as “significant” if it is statistically significant considering $\rho \leq 0.01$ unless otherwise stated.

- **Hyper-parameters** Discount factor for future rewards $\gamma = 0.95$; maximum depth for the tree is 25; Maximum of simulations equals to 250 per search.

- **Hardware and System information** Each experiment run was performed in a single node of a high-performance cluster, containing 16 Intel Ivy Bridge cores with 64GB of RAM.

6.5 Empirical Results and Discussion

In this section, we present the main results found in our experimental dataset. We begin our empirical analysis with the results for the Adversarial Detection Setting, in Section 6.5.1. Afterwards, we move to the the Ad-hoc Adversarial Detection Setting evaluation, in Section 6.5.2 In the end, we present further plots that support the visualisation of the numerical results presented through this section, in Section 6.5.3.

6.5.1 Adversarial Detection

Figure 6.8 shows the performance of BAE in detecting the impostor among the team members, while Table 6.10 presents a summary of our results including the performance of our baselines.

Table 6.10: Summarised results for each adversary detection approach. The highlighted values indicate when a method presents statistical significance in its results among all competitors.

	LBF.a		LBF.b		LBF.c		LBF.d	
	$P(\omega_\psi = \psi)$	\mathbf{t} (sec)	$P(\omega_\psi = \psi)$	\mathbf{t} (sec)	$P(\omega_\psi = \psi)$	\mathbf{t} (sec)	$P(\omega_\psi = \psi)$	\mathbf{t} (sec)
BAE	0.76 ± 0.06	2.15 ± 0.40	0.61 ± 0.06	6.08 ± 0.60	0.35 ± 0.02	10.45 ± 1.45	0.31 ± 0.04	13.73 ± 1.10
AGA	0.53 ± 0.10	2.79 ± 0.63	0.51 ± 0.09	5.50 ± 0.56	0.25 ± 0.07	9.44 ± 1.24	0.27 ± 0.07	10.63 ± 1.17
ABU	0.52 ± 0.11	2.83 ± 0.63	0.51 ± 0.11	5.54 ± 0.55	0.27 ± 0.08	9.48 ± 1.21	0.27 ± 0.09	10.11 ± 1.13
OEATA-A	0.50 ± 0.00	9.98 ± 1.23	0.50 ± 0.00	19.11 ± 2.07	0.25 ± 0.00	74.96 ± 29.96	0.25 ± 0.00	75.52 ± 26.81

From Figure 6.8, it is evident that BAE successfully detects the impostor agent across *all* scenarios with statistical significance. However, although BAE did not attain a maximum confidence level of $P(\omega_\psi = \psi) = 1$, it was able to effectively distinguish, in probability, the agent most likely to be the impostor.

We now turn our attention to the summary of the results given in Table 6.10. Numerically, we observe that BAE consistently outperforms its baselines in detecting adversaries across *all* scenarios, with the sole exception being the Medium scenario (LBF.b), against which the p-value stands at $p \geq 0.23$. In addition, it is important to state that in the Big scenario (LBF.c), a significant difference between BAE and

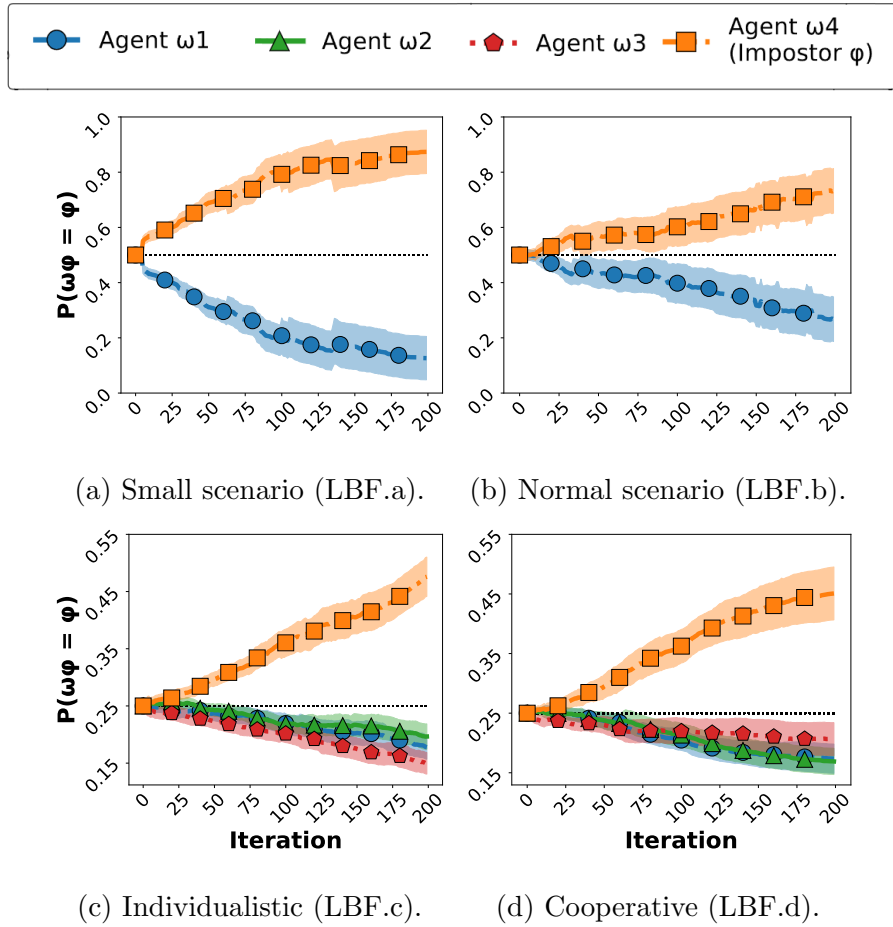


Figure 6.8: BAE’s impostor probability across iteration per agent for each defined scenario in the LBF environment.

AGA is only observed when considering a p-value $p < 0.06$.

This enhancement of performance can be attributed to BAE’s improved capability to discern the agents’ objectives by abstracting their actions while considering the current knowledge available in real time. In contrast to the type-based baselines, BAE exhibits the ability to swiftly identify variations in agent behaviour by relying on its rationale, defined by its planning strategy, together with the conducted simulation processes.

While we acknowledge that enhancing the complexity of the template for approximating adversarial behaviour could potentially boost the performance of type-based methods, it remains a fair comparison because BAE performs adversarial

detection from scratch and does not rely on previously known templates.

In OEATA-A’s case, we attribute our better performance to the fact that we do not rely on hard assumptions to update our knowledge. Two main assumptions lead OEATA-A to fail in estimating the impostor: (i) all templates for non-strategic agents *must fail* (after an update) in estimating the correct task to approximate the “suspicious agent” (i.e., the success counter of all estimators *must* be equal to 0 to increase the suspicious probability), and (ii) the update of an agent’s estimators *only* occurs after the competition of a task. However, the impostor agent in our experimental settings *does not* complete tasks since it wants to minimise the team’s reward collection. Therefore, OEATA-A fails in these assumptions and, hence, never updates its probability for suspicious agents.

In terms of time efficiency, BAE demonstrates an enhanced ability to detect adversaries without significantly increasing its necessary time to perform the decision-making process. In the small scenario (LBF.a), we can see that BAE significantly outperforms AGA, ABU, and OEATA-A in terms of reasoning time. In the LBF.b and LBF.c scenarios, BAE exhibits a slightly higher average reasoning time, but this difference lacks statistical significance, as indicated by p-values $p > 0.6$ when compared to all tested baselines. However, in the LBF.d scenario, BAE does show a higher reasoning time compared to its baselines. We attribute this increase in reasoning time to BAE’s simulation of cooperation between agents at deeper levels of the search tree. Its ability to better approximate adversarial agents results in a preference for exploring branches of the tree where our ad-hoc agent handles potential outcomes for a specific adversary agent. Instead of performing more simulation steps during the rollout phase, we conjecture that BAE often performs more simulation steps inside the existent tree because of its confidence in identifying the true adversary, i.e., as the confidence about who is the true impostor increases, the probability of exploring new paths in the tree decreases. Note that, OEATA-A requires significantly more time to run than other baselines due to its estimation approach, which requires the generation and evaluation of a large amount of estimators

one by one to successfully perform estimations.

Overall, our method demonstrated success in detecting impostor agents across three out of four scenarios with statistical significance, all while maintaining efficiency in terms of time. We highlight that BAE achieved these results without relying on pre-defined templates and by conducting adversary detection from scratch.

6.5.2 Ad-hoc Adversary Detection

For this experimental configuration, Table 6.11 provides a summary of our results.

Table 6.11: Summarised results for the Ad-hoc Adversary Detection experimental setup. The highlighted values indicate instances when AGA-BAE improves AGA’s results and when ABU-BAE improves ABU’s results with statistical significance.

	LBF.a		LBF.b		LBF.c		LBF.d	
	$P(\omega_\psi = \psi)$	t (sec)	$P(\omega_\psi = \psi)$	t (sec)	$P(\omega_\psi = \psi)$	t (sec)	$P(\omega_\psi = \psi)$	t (sec)
AGA-BAE	0.58 ± 0.09	2.79 ± 0.62	0.56 ± 0.08	5.20 ± 0.55	0.29 ± 0.04	8.92 ± 1.20	0.28 ± 0.03	8.94 ± 1.00
ABU-BAE	0.57 ± 0.10	2.82 ± 0.62	0.58 ± 0.08	5.29 ± 0.54	0.27 ± 0.03	9.28 ± 1.21	0.30 ± 0.03	8.61 ± 1.06

Upon comparing the results in Table 6.10 to those in Table 6.11, we see that BAE demonstrates a significant enhancement in the performance of AGA and ABU methods with regard to adversary detection performance. This improvement was observed across all scenarios for both presented solutions, AGA-BAE and ABU-BAE, with the exception of ABU and ABU-BAE in the Big scenario (LBF.c), where the p-value was notably high at $p = 0.99$, and AGA and AGA-BAE in the Cooperative scenario (LBF.d), where $p = 0.15$.

As for reasoning time, AGA-BAE consistently achieves estimations significantly faster than AGA across three out of the four scenarios, with all p-values below 0.01. The only exception was the Small scenario (LBF.a), where $p = 0.98$. On the other hand, ABU-BAE exhibits notably faster estimation times than ABU in two out of four scenarios, with p-values under the 0.01 threshold. The Small (LBF.a) and Big (LBF.c) scenarios stand out as exceptions, with p-values of $p = 0.81$ and $p = 0.19$, respectively.

Overall, we summarise that the proposed approach significantly enhances the performance of type-based estimation methods in detecting adversarial agents within an environment while avoiding the need to create a reliable template type for impostors or train the method using historical data. Additionally, BAE proved capable of reducing the time required for estimation for both baselines, an attribute that can be beneficial in certain scenarios and applications.

6.5.3 Additional Plots for Results Visualisation

In this section, we provide additional materials to support the reader with the visualisation of our results presented in Table 6.10 and 6.11.

Therefore, we first present AGA’s estimations of impostor type probabilities through a line plot in Figure 6.9 followed by ABU’s impostor type probabilities, in a similar representation, in Figure 6.10. These plots present the algorithms’ performance in estimation across various templates and particularly for the true impostor. Subsequently, we turn our attention to the line plots to analyse the performance of AGA and ABU when running together with BAE. Specifically, Figures 6.11 and 6.12 delineate the outcomes for AGA-BAE and ABU-BAE, respectively. These results considers the algorithms’ estimations across agents (teammates), including the true adversary.

Analysing Figures 6.9 and 6.10, we can see that both algorithms encounter challenges in accurately differentiating the true impostor as the adversarial agent in the environment. Notably, there is a lack of statistical significance in the probabilities associated with different potential impostor types during these estimations.

On the other hand, when we analyse the AGA and ABU versions running with BAE, in Figures 6.11 and 6.12, a noticeable improvement emerges compared to their standalone applications. In different parts of the execution, the algorithms demonstrate statistical significance in differentiating the true impostor as the adversarial agent in the environment. Remarkably, ABU-BAE exhibits superior deductive capabilities in identifying the adversary, concluding the execution of the

problem by designating agent ω_4 as the true impostor among teammates in 3 out of 4 scenarios ($\rho < 0.05$), whereas AGA-BAE achieves statistical significance in only 1 out of 4 scenarios.

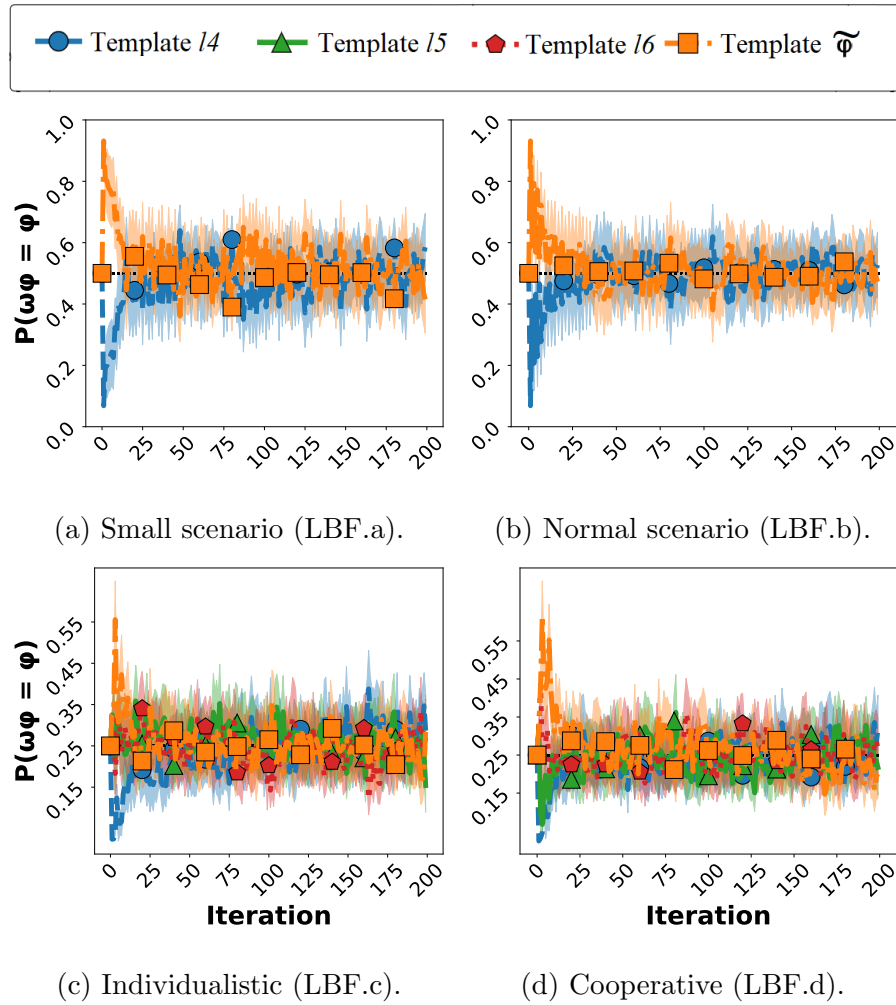


Figure 6.9: AGA's impostor probability across iteration per template type for each defined scenario in the LBF environment.

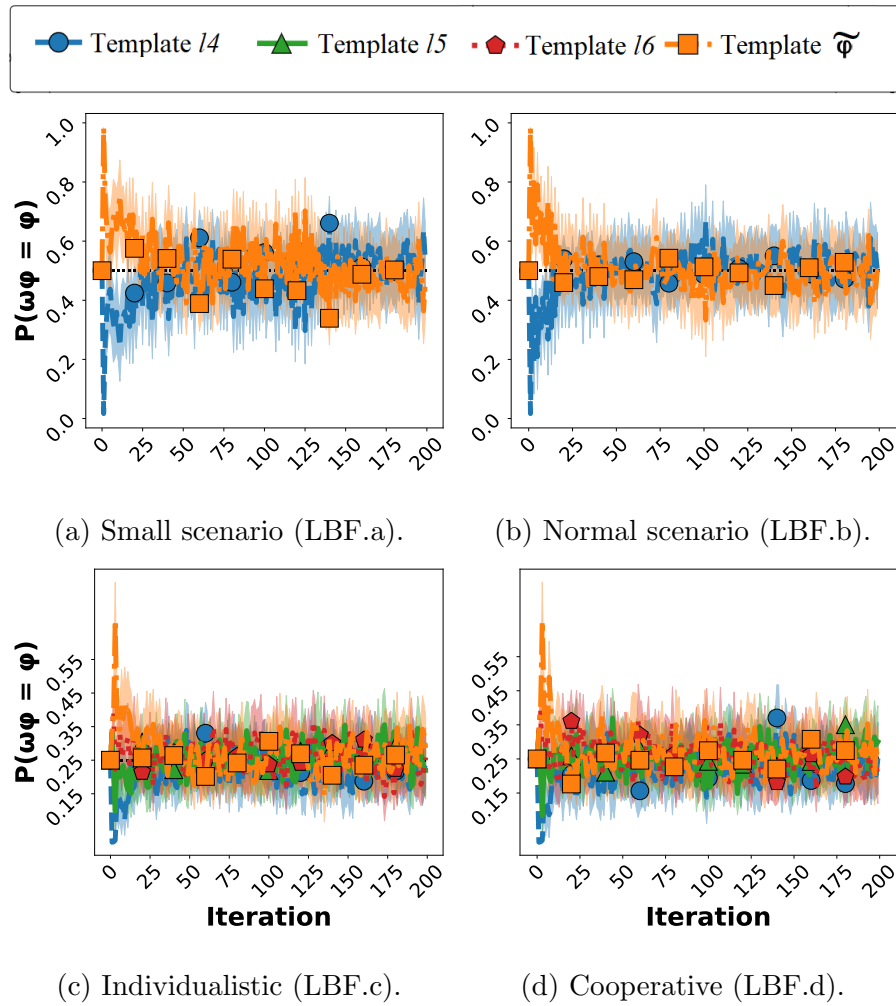


Figure 6.10: ABU's impostor probability across iteration per template type for each defined scenario in the LBF environment.

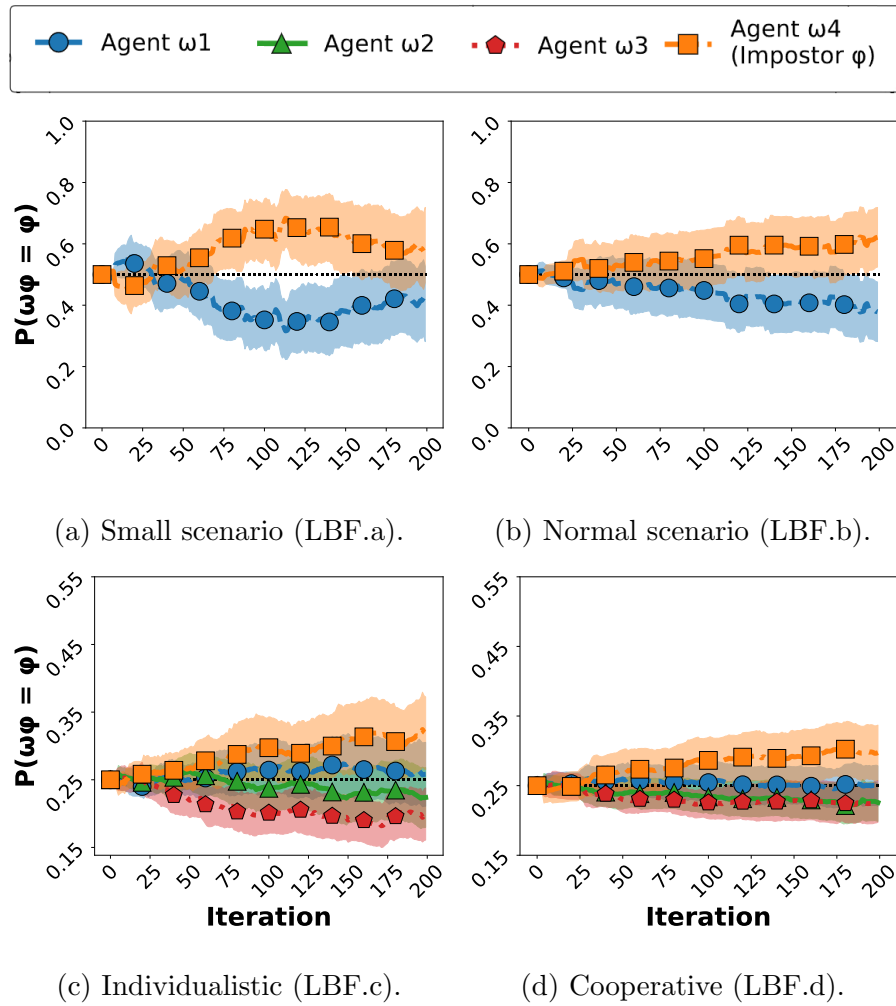


Figure 6.11: AGA-BAE's impostor probability across iteration per agent for each defined scenario in the LBF environment.

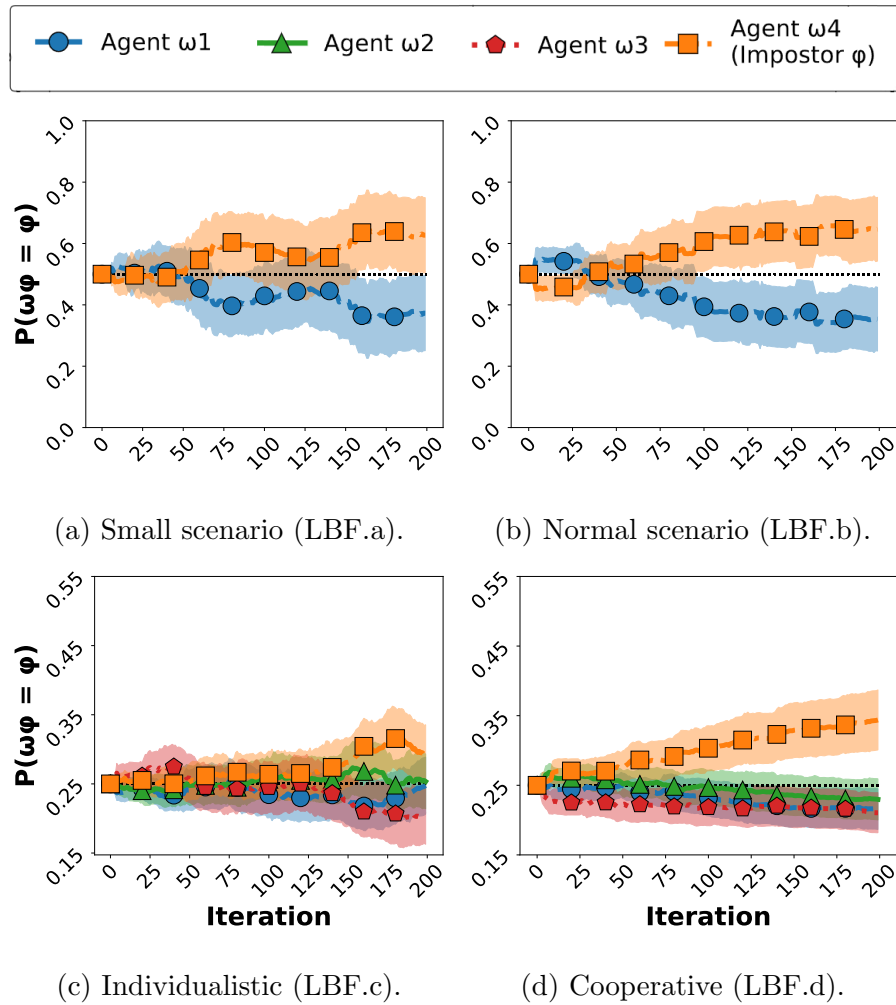


Figure 6.12: ABU-BAE’s impostor probability across iteration per agent for each defined scenario in the LBF environment.

6.6 Chapter Conclusion

BAE is a novel algorithm for online planning and estimation in ad-hoc reasoning domains, where agents share the same environment but have no information about their teammates’ type, parameters and true intentions. We show that our method is capable of efficiently identifying an impostor agent across four different scenarios without relying on pre-trained models or previously available data.

Chapter 7

Conclusions

7.1 PhD Outline

In this PhD thesis, we have systematically explored different approaches and strategies to handle uncertainty in partially observable scenarios. By conducting a comprehensive analysis of both contemporary state-of-the-art methodologies and classical literature, we have introduced a spectrum of solutions for online planning and online estimation across a diverse application context.

Our contributions lie in the formulation of innovative algorithms that combine statistical and reinforcement learning techniques in a precise manner. The integration of these methodologies has shown great benefit for the optimization of autonomous processes as a powerful tool, enhancing the algorithms' adaptability and efficiency in the face of dynamic and uncertain environments. Notably, our approaches could also alleviate the prevalent dependency on extensive computational resources and massive amounts of data to achieve good performance. We have ventured beyond the boundaries of existing literature, pushing our proposals to more complex and challenging problems. Since the beginning, we have focused on providing smart, practical, and accessible solutions for the implementation of efficient autonomous agents in terms of learning, planning, and estimation capabilities.

“Autonomous agents are prevalent and recurrent in our society, playing an important role in optimizing and improving people’s quality of life. Hence, it must be accessible and efficient for everyone.”

The central theme highlighted in this thesis can be described through the following rationale and sequence of events:

We started by examining the challenges faced by a single agent navigating an environment full of uncertainties. Under this perspective, we have explored various approaches to enhance decision-making processes featuring the constraints stated by the partial observability. Throughout our investigation, we identified a notable limitation in existing literature hindering performance improvement: the prevalent reward-guided paradigm embedded within reinforcement learning solutions. Facing the uncertainty and reasoning about how to surpass the barriers imposed by this paradigm, we proposed an innovative approach capable of extracting valuable insights from environmental observations and informed problem reasoning. Considering the recurrent integration of Markovian models with reinforcement learning solutions, our approach could enhance the performance of a single agent acting in a partially observable scenario without relying on additional resources, such as computational power or time. Besides that, our solution shows versatility and generality in its application, showcasing its potential in handling various contexts and modifying different frameworks. This adaptability opens avenues for the creation of methods adhering to alternative planning paradigms. As a consequence of this advancement, we introduced the concept of the “information-guided planning paradigm,” wherein planning capabilities evolve not solely from rewards but also from the systematic collection of information from the environment. A concrete manifestation of this contribution’s significance is the proposal of IB-POMCP, whose novelty and impact will be summarized in Section 7.2.1.

“Measuring information instead of rewards to enable and develop efficient planning has demonstrated promising progress in overcoming the barriers related to the scarcity of data or computational resources.”

After presenting a solution to the aforementioned problem from a single-agent perspective, a natural question arises: how can we address similar challenges within a multi-agent scenario? Upon delving into this context, we observed that the primary barriers in multi-agent systems were not exclusively tied to uncertainties in the environment. Instead, these challenges predominantly stemmed from uncertainties related to fostering effective collaboration and coordination among agents. As demonstrated throughout this thesis, various scenarios present challenges when it is required to understand how each teammate reasons about the world and plans its actions toward cooperation. Revisiting the literature, we identified that current teamwork models and planning solutions in this context could benefit from the information gained through observations retrieved from the world. This strategy resembles the single-agent solution but now is applied to reduce uncertainty related to potential teammates instead of the pure environment. Consequently, we proposed a novel approach that integrates reinforcement learning, type and parameter estimation, and genetic algorithms as a potent tool to enhance planning, coordination, and cooperation in an environment with unknown teammates. Through this strategy, we effectively addressed these challenges without requiring the estimation of true models for our teammates, thus conserving computational resources without compromising performance. The concrete contribution related to this outcome is the proposed OEATE, whose novelty and impact will be summarized in Section 7.2.2.

“When cooperating, we do not need to delve into our teammates’ minds; we need to find effective ways to understand them and adapt ourselves.”

At the culmination of this journey, after developing an agent capable of navigating scenarios with uncertainty from both the environment and its teammates' behavior, a final question emerged: What if someone assumes the role of an adversary, attempting to deceive our planning and operate beyond the bounds of our accumulated knowledge? While it may be hard to acknowledge, the presence of malicious applications capable of undermining an agent's planning capabilities cannot be ignored. This thesis illustrates scenarios where autonomous agents mandatorily rely on their mutual trust to successfully complete tasks and achieve common objectives. However, under our assumptions, being aware of potential impostors within the system can prevent damage to both the system itself and those surrounding it. As a recent topic in the literature, the multi-agent systems community is starting to further explore these scenarios with disguised adversaries, lacking relevant solutions in the context. From our exploration of the literature, we noticed that, akin to our previous application of information knowledge, it might be plausible to identify adversarial agents operating within a teamwork context using a similar strategy. Through the blended perspective of computing and statistics, we discovered that extracting information from observations and comparing it with observable actions of other agents could help identify potential impostors. The consequence of this investigation was the proposal of a novel approach capable of aggregating knowledge by observing the environment and the actions of other agents to discern their intentions and roles in the system without relying on true models or previous knowledge. By systematically evaluating the impact of each action for potential adversarial agents in the environment, we could effectively identify impostors within the team. The concrete contribution of this project is the proposal of BAE, whose novelty and impact will be summarized in Section 7.2.3.

“Intelligence is ineffective if we cannot comprehend the diversity of the world around us; the same for the robots.”

In light of the comprehensive discussion above, this conclusion section aims to revisit some details of our contributions, underscoring their significance for the community in different aspects.

First, in Section 7.2, we present a brief but refreshing summary of our contributions to support this concluding discussion. In Section 7.3, we discuss in detail all the actual and potential limitations for each of our proposed methods. This detailed exploration provides transparency regarding the constraints and challenges associated with our approaches. Finally, in Section 7.4, we extrapolate potential paths for future research, building upon the insights gained from both the achievements and shortcomings of this thesis. The thesis ends with the appendix and references sections.

7.2 Contributions

This thesis analyzed and developed solutions in online planning under partial observability, considering unknown features of the environment and potential other agents (teammates) in the environment. Our primary focus centered on Monte Carlo-based methods, applied to both single and multi-agent scenarios, and we extended their application to diverse and complex settings for empirical and theoretical evaluations.

In essence, this thesis aimed to assess how an autonomous agent can effectively manage various levels of uncertainty by aggregating information during the planning and decision-making processes in an online manner. Our findings demonstrate that employing different strategies enables the aggregation of information and extraction of knowledge about the problem not accessible prior to its execution. Furthermore, we illustrate that our solutions can execute and complete tasks without significantly increasing resource usage, challenging the prevailing belief in the literature that suggests an extensive amount of data is necessary to train and solve complex problems.

In summary, the primary contributions of this thesis that are both relevant and novel within the community and our research area include:

- A novel perspective on applying information to enhance online planning methods facing uncertainty;
- Innovative models for the application of statistical approaches and learning methods that do not require significantly increased resources to solve problems; and
- A novel framework capable of unifying and implementing all the aforementioned methods and scenarios discussed for research and evaluation purposes.

To provide a more in-depth analysis of each contribution, we save space in the following sections to discuss each method individually, offering a comprehensive examination of the advancements developed during this thesis.

7.2.1 IB-POMCP

As the first contribution presented in this thesis, we proposed *Information-based POMCP (IB-POMCP)*, a novel algorithm for planning under uncertainty that is capable of aggregating information entropy into a decision-making algorithm using our modified version of the UCB function, *I-UCB*. Our information-guided planning approach leads the agent to surpass recurrent limitations imposed by the traditional reward-guided planning paradigm presented in the state-of-the-art literature.

IB-POMCP is a direct answer to two research questions introduced in this thesis:

- *Considering the aspects of planning under uncertainty, how can we handle the lack of information without penalizing time or spending significantly more resources?*, and;
- *Considering the current reward-guided paradigm in RL solutions, how can we solve problems with sparse rewards?*

We handled the lack of information by proposing a new paradigm for planning, the “*information-guided planning paradigm*”, which can evaluate and plan actions based

not solely on the reward collection but also on an entropy analysis procedure over the set of collected observations. The collected observations by each action enable the calculation of entropy, hence, the gain of information for each action and possible path ahead. We use this value to improve the reasoning quality, even when no reward is available. In addition, IB-POMCP presents theoretical properties for convergence under certain assumptions, which are supported by empirical results collected in five different domains and several different scenarios. Overall, our proposal could increase the reward collection by up to 10 times in comparison with the state-of-the-art method, TB ρ -POMCP (U-shaped, F1), in addition to reducing the reasoning time by up to 93% compared to ρ -POMCP (MazeHoles).

7.2.2 OEATE

As the second contribution proposed in this thesis, we presented *Online Estimators for Ad-hoc Task Execution* (OEATE), a new algorithm for estimating types and parameters of teammates, specifically designed for problems where there is a set of tasks to be completed in a scenario. By focusing on decentralized task execution, we are able to enhance the capabilities of state-of-the-art methods presented in the literature, obtaining lower error in parameter and type estimation than previous works, which leads to better overall performance.

Besides being lightweight (addressing the first research question of this thesis), OEATE focuses on delivering a solution that replies specifically to the question:

- “*Considering multi-agent systems working under uncertainty, how can we surpass the limitations posed by performing teamwork with unknown teammates?*”

Our answer considers a blend of applications between online planning, online learning, feature estimation, and genetic algorithms. We developed this study theoretically and empirically, showing that OEATE converges to zero error as the number of tasks increases (under some assumptions) in a wide range of situations (e.g., scaling number of items, number of agents, scenario sizes, and number of types

in our experiments). OEATE could outperform the previous works with statistical significance in some of these cases. This work opens the path to diverse studies regarding the improvement of ad-hoc teams through a task-based perspective and using an information-oriented approach.

7.2.3 BAE

As the last concrete contribution of this thesis, we present *Bayesian Adversary Estimation* (BAE), a novel algorithm for online planning and estimation while handling an ad-hoc reasoning domain, where agents share the same environment but have no information about their teammates' type, parameters, and true intentions. We show that our method is capable of efficiently identifying an impostor agent across four different scenarios without relying on pre-trained models or previously available data.

This proposal aims to answer the last research question of this thesis:

- “*Considering the emerging discussion about adversaries in decision-making systems, how can we handle it without penalizing the algorithm planning capabilities?*”

Straightforwardly, we propose the application of what we denominate as the *Q-valued Bayesian Estimation* (QvBE) approach, which considers evaluating action values using information that is not directly available by observing the world – such as the probability distribution function of teammates' actions using the Q-values estimated during the reasoning process. Although we focused on proposing a solution based on Monte-Carlo Tree Search methods, we emphasize that it can be extended to any online planning algorithm that estimates Q-values. The main idea behind our proposal is to embed the QvBE approach inside an Adversarial-MCTS, which performs planning from scratch and estimates its own actions considering the existence of an adversary. The Q-table found at the end of the search process is used to estimate the impostor among the team.

7.3 Limitations

7.3.1 IB-POMCP

While our proposed approach for IB-POMCP, which involves the integration of an information-guided planning strategy through the application of our innovative I-UCB function, exhibits certain limitations in specific contexts where a reduction in uncertainty may adversely affect performance.

Consider a hypothetical scenario where agents, aiming to enhance performance, must undertake risky actions to achieve their objectives. In such a case, IB-POMCP may fail to improve performance depending on the underlying model design of the planning algorithm. For example, if the model suggests that accumulating knowledge is advantageous, yet the value for information gain consistently surpasses the value obtained through collecting rewards with high uncertainty, IB-POMCP may tend to prioritize knowledge aggregation over taking risky actions. Consequently, it will avoid accomplishing tasks in favor of reducing uncertainty.

Another constraint emerges from the characteristics of our spaces inside the decision-making model. The first limitation can occur when the observation space is too limited to differentiate actions based on entropy values. From a similar perspective but in the opposite direction, in cases where the action space is excessively large, IB-POMCP would necessitate a higher number of simulations to distinguish actions by analyzing potential observations in the environment. However, it is crucial to note that, for both cases, the performance of IB-POMCP remains bounded by the efficacy of traditional reward-guided planning algorithms, where it may exhibit comparable or superior performance.

7.3.2 OEATE

One of the key strengths of OEATE lies in its flexibility, allowing teammates to operate with different problem representations and employ diverse algorithms without explicit agreement in task selection. OEATE successfully models and views each agent from a task-based perspective, enabling effective estimation even when teammates operate with distinct paradigms or task representations. However, from this perspective, a notable limitation arises when dealing with a “lazy agent”, which presents challenges in approximating template models to its behavior. The task-based ad-hoc teamwork model assumes that all agents within a team will actively contribute to the accomplishment of tasks for the team’s benefit at some point in time. Consequently, if the agent is “lazy” or even an impostor in the team, OEATE might fail to estimate its types and parameters correctly due to the lack of proactivity from the agent. Note that this situation is different from our case of study with wrong templates. There, an agent is still completing tasks, but the correct template for estimation is not available.

In scenarios involving partial observability, our algorithm still relies on knowledge regarding which agents completed specific tasks, even beyond our controlled agent visibility region. If the algorithm fails to obtain this information, it might inaccurately estimate the agent’s potential type and parameters. Again, due to the assumptions imposed by the task-based ad-hoc model, OEATE is constrained to these situations.

Finally, an important implication, which highlights another limitation of our study, is: improving the knowledge of the ad-hoc agent about non-learning teammate types did not always lead to an improvement in performance. This observation suggests that traditional benchmark problems may not be ideal for evaluating methods emphasizing accurate modeling of neighbor types, particularly in scenarios involving individualistic agents brought together for planning without direct cooperation towards a common objective. It is crucial to note that our approach’s performance would be constrained to the efficacy of the planning method it employs in such scenarios.

7.3.3 BAE

Although BAE is lightweight and efficient in estimating potential impostors in a teamwork environment, it still shows some limitations in its developed strategy.

Our empirical findings highlight certain limitations in BAE’s estimations related to the action space size. If there are not enough actions to differentiate non-adversarial and adversarial agents, our method may take significantly more time to accurately update the impostor probability among teammates. The underlying rationale is that if all agents within the environment are often performing the same actions (even considering the spatial difference between them, i.e., their position), the absence of distinctive actions will prevent the identification of adversaries. The same can happen when there is a large number of agents in the scenario if agents rarely take actions that differentiate their objective or intention from other agents. We acknowledge that this setting somewhat benefits the adversarial agent, which makes the problem challenging for any estimation method including BAE. Hence, it is possible that under different and more challenging settings for the adversary, BAE may yield different results regarding this limitation, which also raises the question: how quickly can an impostor be detected by BAE after engaging in suspicious behavior?

Moreover, similar to OEATE and contrary to expectations, augmenting knowledge about teammates’ roles and their true objectives in the scenario does not consistently result in improvements for team performance. Particularly in an adversarial context, there may be cases where attempting to identify and mitigate the impact of an adversary in the environment can negatively impact task completion. Sometimes, if this information was overlooked during the planning phase, the benefits would be greater in terms of performance. In other words, the adaptability in planning, stemming from extensive knowledge, sometimes disrupts agent reasoning by introducing multiple layers of complexity in the planning process. Finding a balance between these aspects is imperative to consistently achieve optimal performance when implementing BAE and other estimation methods; but sometimes, information to find this balance is faulty or nonexistent.

7.4 Future Works

7.4.1 IB-POMCP

Exploring potential avenues for future research inspired by IB-POMCP’s proposal involves considering its extension to multi-agent systems with uncertainty. While this thesis has delved into challenges related to estimating types and parameters for teammates sharing an environment, a crucial aspect remains unexplored – how the performance of information-guided planning influences coordination among agents in cooperative scenarios. Managing multiple layers of uncertainty simultaneously poses challenges but offers advantages across various applications.

For instance, extending IB-POMCP to calculate uncertainty while deliberating a chosen path and considering uncertainties related to the decision-making of other agents could empower the ad-hoc agent to tailor strategies based on confidence levels in its estimations about non-learning agents. This, in turn, holds the potential for performance improvements.

Another direction for extension involves adapting the information-guided planning paradigm to address scenarios where increasing uncertainty is desirable, rather than decreasing it. Our current approach predominantly focuses on problems where decreasing uncertainty benefits the decision-making capabilities of our ad-hoc agent. However, as pointed out in the previous section, there are problems where this perspective in planning can disrupt the process. Expanding IB-POMCP’s framework to handle increasing uncertainty would be essential for defining a comprehensive and versatile reasoning process, enabling the application of our approach to a broader range of problems.

Finally, evaluating IB-POMCP’s capacity to handle large action spaces and small observation spaces would contribute to refining our proposal. Our current suspicion is that the algorithm might be constrained by the characteristics of the POMDP model. Exploring alternative statistical methods, beyond our proposed Shannon Entropy application, may also empower IB-POMCP to transcend these constraints

and establish a more robust planning process. However, the feasibility of applying more complex entropy calculation methods should be considered in terms of resource requirements, keeping in mind the trade-off between performance and the necessary computational power to achieve this. Under this perspective, we suggest exploring the adaptation of traditional statistical calculations to an online planning and learning perspective (as we proposed), introducing a lightweight yet effective approach.

7.4.2 OEATE

OEATE's future works encompass the generalization of its applications beyond the current capabilities of the ad-hoc teamwork model employed in its estimation process, expanding its capabilities to different real-world contexts.

For instance, addressing the partial observability limitation discussed in the preceding section necessitates further exploration to refine the estimation process when information is lacking for agents beyond the ad-hoc agent vision region. In real-world scenarios, potential solutions may involve incorporating external hardware to detect agents and tasks outside the line of sight. Alternatively, calibrating the method to synergize with communication channels, relying on trusted agents to exchange messages and continuously estimate types and parameters, could enhance performance across various problems. However, eliminating these assumptions from "task-based ad-hoc teamwork" under partial observability emerges as an intriguing and challenging avenue for future research.

Another pertinent direction for future work on OEATE involves developing planning capabilities that account for potential variations in the roles of teammates, as proposed in BAE, which considers adversarial agents in the team. The initial findings presented by Shafipour and Fallah (2021) [90] with the OEATA-A proposal, utilizing a prior version of OEATE (OEATA [91]), demonstrate the potential of this avenue for advancement. However, numerous aspects need refinement to present a superior strategy compared to OEATA-A (whose limitations were discussed in our related works) and BAE.

7.4.3 BAE

Discussing future developments for BAE involves refining its sensitivity to spatial features during the simulation of potential and multiple adversaries at the same time.

As preliminary results, we have conducted tests and assessments on an alternative version of BAE. One of them was labeled the Multi-Tree MCTS BAE. This variant involves simulating each potential impostor within distinct adversarial trees, all at once. In contrast to EQ-MCTS, the version presented in this thesis, this approach eliminates the necessity to aggregate multiple estimated Q-values from different impostors during the tree search process for adversary identification.

Our rationale behind this approach was to isolate each potential adversary within its own tree, concentrating exclusively on their spatial and temporal characteristics, with the expectation of potentially enhancing results, albeit at the expense of increased computational resources. However, despite running more simulations across different trees, no significant improvement in outcomes was observed. We posit that refining the model and devising a more effective strategy for aggregating Q-values, considering the spatial configuration of the target agent's actions, could lead to improvements in adversarial detection results. Our limitations discussion gives insights into the cause and how to solve these problems, incentivizing the development of future works in this line of research.

Appendix A

AdLeap-MAS: An Open-Source Multi-Agent Simulator for Ad-hoc Reasoning

A.1 Introduction

Autonomous systems play notable roles in contemporary society. They perform vital daily tasks, provide several critical services, and collectively constitute a significant proportion of digital systems. The increasing number of devices sharing the same environment present a set of more complex problems that require the proposal of new intelligence methods capable of solving tasks, learning about each other and handling uncertainties in an online-fashion.

A typical approach presented by the state-of-art is to assign multiple intelligent agents to solve a common objective, defining a *Multi-Agent System* (MAS) context. Within this domain, ad-hoc teamwork modelling has been a successful approach to coordinate task accomplishment of autonomous systems [95, 102, 111]. This allows agents to cooperate with previously unknown teammates over full or partial observability of an environment, leading the agents to reason and learn about each other and the environment in a decentralised-fashion [2, 46, 81, 91].

In such scenarios, some features are critical for agents to act in the environment. Besides being able to interact with the scenario, it is commonly necessary that they present the capability to make decisions in an online manner while handling uncertainties to efficiently solve problems in the environment. It considers, for example, the application of estimation algorithms to approximate knowledge about the world and about the role potential teammates are playing. Consequently, these domains usually involve a tight combination of on-line learning, on-line planning and on-line estimation algorithms in order to solve a problem. We refer this context as an *Ad-hoc reasoning* context.

When proposing solutions in this complex domain, it is extremely beneficial to thoroughly evaluate how a system reacts to its environment through simulations, which helps avoid significant investment in equipment and deployment costs. Hence, simulators have been crucial for the progress of Ad-hoc reasoning research by facilitating reliable and reproducible experiments to test new concepts, strategies, and algorithms [59].

Regarding the community interested in ad-hoc reasoning solutions (e.g., ad-hoc teamwork, opponent modelling, online planning, latent features estimation, etc), every research team seems to be implementing scenarios using their own self-built simulators [6, 20, 35, 78, 79, 85, 98, 105, 91], missing the opportunity to propose a common and trustworthy platform that addresses the community necessities to develop research and propose new solutions.

Therefore, we propose the open-source *Adaptive Learning and Planning Multi-agent Simulator* (AdLeap-MAS), a novel framework focused on simulating ad-hoc reasoning domains and aiding the evaluation of on-line learning and planning methods for individualistic, cooperative and adversarial Ad-hoc contexts. Through a component-based architecture, this proposal aims to minimise the implementation cost that precedes domain evaluation; including designing the environment, component settings, and benchmarks definition. AdLeap-MAS presents a set of ready-to-use tools and inputs that enables low-effort “plug-in” functionalities, such as

testing new learning and estimation methods. Our framework uses the *Gymnasium* package [101], the recent and updated version of *Open-AI Gym* [21], to build its solution and is publicly available to encourage code sharing and platform usage, offering a comprehensive documentation and an active support to the project.

Up to date, our framework’s repository provides twelve (12) distinct environment and more than twenty (20) different scenarios in total to perform experiments. Beyond that, AdLeap-MAS supports the execution of reactive algorithms, neural networks, estimation methods for modelling previously unknown agents and reinforcement learning application over full and partial observability, requiring the user only to plug-in their algorithms into the Ad-hoc reasoning base model. In this matter, we offer several reactive algorithms to run non-strategic agents, besides eight (8) planning algorithms and six (6) estimation methods from the state-of-the-art as baselines. AdLeap-MAS also offer the implementation of supportive codes for the collection, analysis and plot of the data generated by the framework.

In summary, the contributions of the AdLeap-MAS’s project can be listed as:

1. It is a novel and *open-source framework* for the simulation of ad-hoc reasoning domain(Section A.3–A.4);
2. It is build over a *component-based architecture*, which design enables the creation of input points that facilitate the adaptation or modification of the simulation characteristics in a quickly and efficiently manner (Section A.4), and;
3. It presents a *generic model for implementing Ad-hoc reasoning problems* focused on providing higher code portability and a broader problem representation (Section A.4.2). That will allow the community to easily benchmark new algorithms for on-line learning, planning and estimation in ad-hoc contexts.

Double-check our Git’Hub for further details and tips about how to use it at <https://github.com/lsmcolab/adleap-mas/>.

A.2 Related Work

In multi-agents research, simulators are recurrently used as a supportive tool aiming to test new concepts, strategies, and algorithms quickly and efficiently. As a consequence, several works evaluate their novel algorithms by doing simulations over relevant domains for the community, such as level-based foraging [91], capture-the-flag games [81] and autonomous driving environments [79]. However, it is costly to implement or find a good simulator that fits these applications.

For instance, Gazebo [59] tackles the representation and simulation of 3D scenarios in robotics. Gazebo became popular due to its ability to present high-fidelity simulations of multi-robot problems. However, Gazebo does not provide support for learning/planning algorithms, maintaining its focus on simulating restricted robotics scenarios without plenty of agents in the same execution. In contrast, AdLeap-MAS offers a lightweight simulation alternative that supports the execution of learning/planning algorithms within the experiments. Using the Gymnasium and our available benchmarks, we allow the user to quickly run the intelligence algorithms within the multi-agent scenario and also support the implementation and evaluation of new methods to tackle the context of interest.

Intending to solve the same problem as Gazebo, the Stage simulator [103] proposes a scalable version of the previous Player/Stage simulator [41]. Although presenting a good performance and scalability, Stage does not provide a world-model for on-line planning, making it hard to be used as a simulator in ad-hoc contexts. Similar limitations emerge in the Swarmanoid project [82] or the SUMO [16] simulators, specialised in the representation and assessment of swarms and urban mobility problems, respectively.

Therefore, we address these problems by presenting an implementation that easily allows a world model, and agents algorithms to be used as inputs to define an ad-hoc problem setting for simulation. The AdLeap-MAS framework can represent the same problems described by these state-of-art proposals but through our generic design proposal to better match with Ad-hoc reasoning domains. Furthermore, in

order to make it portable, we implemented it using Python 3 language, supported by Gymnasium, and created ready-to-use input points to change the execution settings.

Also, from the state-of-art, the GAMA simulator [37] proposes the modelling and simulation of spatially explicit multi-agent domains. However, it is not focused on ad-hoc contexts, making it hard to implement the required combination of on-line learning and planning in this framework. Despite the issue of acquiring a world model, the framework also requires an understanding of its dedicated programming language before utilisation. In contrast, AdLeap-MAS uses the Python programming language for the entire implementation without requiring the usage of supportive languages. We built our framework over a component-based architecture that provides independence for each module (fully implemented in Python). This loose coupling improves robustness and facilitates the local management of components and an architecture-independent testing procedure. In other words, the environment simulation occurs through a unilateral communication channel within the architecture, which establishes a standard and precise execution flow to simulate every problem implemented in the framework, allowing the user to debug problems locally without touching the architecture.

OpenSpiel [64] presents a collection of environments and algorithms for research in general reinforcement learning and search/planning in games. Supporting implementations in C++ and Python, this proposal presents relevant contributions related to decision-making evaluation in several contexts. However, the OpenSpiel is not focused on the simulation of Ad-hoc reasoning domains, demanding from the user the design and modelling of these contexts before utilisation. The AdLeap-MAS tackles this deficiency by directly implementing the necessary Ad-hoc simulation tools and a Markovian base model within the framework. We allow the users to easily switch from partial to full observability scenarios and vice-versa, apart from enabling the quick plug-in of estimation algorithms into the reasoning methods. The AdLeap-MAS also offers a set of relevant benchmarks to use or modify before running these problems.

From the old Open-AI Gym [21], Gymnasium [101] is a platform that aims to provide a collection of benchmarks in a software package, offering also convenience and accessibility. Gymnasium represents all the elements in the world as part of the environment. The package abstracts these components to facilitate applying reinforcement learning algorithms, which would receive all this information aggregated into a single state/environment. Additionally, the software package suggests the versioning of its environments to ensure that results remain meaningful and reproducible through the updates. We extend the benefits of these platforms and also improve its range of application. By directly modelling and offering support of Ad-hoc reasoning applications, we provide the opportunity to use the Gymnasium’s benchmarks into a focused environment for evaluation and simulation in this context. We also allow the components to easily swap without requiring the complete environment implementation understanding. Our framework also handles visibility restrictions modifications without requiring the implementation of a new class or method. The AdLeap-MAS describes an embedded framework that seeks to improve the user’s experience with Gymnasium for Ad-hoc domains via its extension related to implemented features for ad-hoc reasoning, and friendly usage.

We propose the AdLeap-MAS as an open-source project that can support the community as a centralised code-source repository providing the algorithms via updates made by the authors or other collaborators, providing valuable baselines for benchmark. We use GitHub as the official website to make our simulator public, a decision that considers GitHub’s reliability and wide use – encouraging code sharing within the community.

Finally, proposing the AdLeap-MAS framework has no intention of replacing or surpassing the capabilities of other simulators. Instead, we aim to provide a robust and efficient simulator for projects that require the simulation of a variable multi-agent context, changing the learning or reasoning paradigms in the simulation or even testing several ad-hoc agents in the same environment. Our objective is to increase collaboration by offering a safe, useful, and centralised shared platform.

A.3 The AdLeap-MAS Framework

The essential principle of our design philosophy is to propose a collaborative work and shared space to develop Ad-hoc reasoning applications, simulations and evaluation within the Multi-Agent systems community. Our framework presents a robust architecture as the main design feature, intending to provide adequate tools to encourage its usage and code spreading. Among them, we can point out **(i)** the facilities provided by the Python language and Gymnasium software package, and **(ii)** the construction of the framework with a component-based architecture. Using these tools, we can provide guarantees related to:

- **Code Portability:** offering the correct mechanisms to freely plug-in learning and reasoning algorithms and environment components without requiring extensive workloads for code adaptation.
- **Integrity:** ensuring that results remain meaningful and reproducible as the software is updated and as the user manage the framework components.
- **Adaptability:** easing the software constraints via a module and model generalisation design, reducing the implementation and adaptation workload.
- **Friendly Usability:** providing sufficient tools to easily perform modifications in the environment without demanding additional knowledge from the user about the framework procedures, presenting and solving problems locally only using the Python standard feedback.

Providing these tools within our framework guarantees, besides the aforementioned benefits, an easy start to develop a novel idea. The framework is implemented over a friendly and easy to learn software development workspace, as the Python language and the Gymnasium package are widely used by the community, representing a safer choice to corroborate our philosophy objectives in this proposal. Additionally, the Gymnasium's philosophies already support the principle linked to our framework

design since users are encouraged to submit their results and provide links to source code with detailed documentation.

Regarding the design features, we can point out the construction of a:

- **Unilateral and Cyclical simulation flow:** minimising the carried out errors, facilitating the debugging process and guaranteeing the correct execution for a diverse set of problems in a linear component execution.
- **Consistent modelling for problems in the Ad-hoc reasoning domain:** assuring the benefits of using the markovian models to present a reliable scenario representation and simulation.
- **Reliable input points to define the environment components:** presenting ready-to-use input points for components that build and modify the environment and/or the agents according to the user's interest.

Note that we are pursuing the idea of offering a robust and friendly alternative to develop, share and advance researches into the Multi-Agents community addressing Ad-hoc reasoning domains, especially.

Finally, we will discuss the details about how these features and tools impacted the decisions made during the framework implementation in Section A.3, starting from a high-level view of the architecture going through some specific implementation characteristics. AdLeap-MAS is available on the GitHub¹ with further information and instructions to usage.

A.4 Framework Architecture

A.4.1 High level view

The AdLeap-MAS's architecture is based on unilateral and cyclical module communication, where the information within the framework must be delivered

¹AdLeap-MAS (Anonymous) GitHub: <https://github.com/lsmcolab/adleap-mas/>

or received directly and exclusively by one module from another in the architecture. Such design enables the problem simulation as a step-by-step process, processing each fragment of the simulation (i.e., functionalities) independently. As in a cascade workflow definition, this specific approach guarantees the correct information analysis and transformation in each step. Furthermore, it is important to note that each module acts independently from the other components. As such, learning and reasoning are based solely on the delivered information.

Perhaps, the arising question now is: how did we separate the environment from its components and the components from their learning and reasoning modules? The answer is direct: we do not. However, considering that each module works strictly over the current information, it is reasonable to assume that this data can provide sufficient knowledge to simulate the environment without building a bilateral communication channel. Likewise, we can describe this situation as the interactions in an office environment, where people of different departments deliver reports to other departments but do not mutually participate in the deliberation processes. In this analogy, the environment module is analogous to the role of the office manager who knows its departments (\approx components) and changes the environment without necessarily deciding how specific employees (\approx agents) should act. The employees represent the agents' component modules (capable of learning, reasoning and taking actions), the structure (components that define the environment organisation) or the tasks (defined by the manager to be accomplished by the employers). Finally, the entire environment (the office in the analogy) passes through the decision-making process, where each character will modify the context by selecting the best action from their perspective.

Following the analogy and explanation, the components are really connected, but performing independent functionalities. Hence, changing the name of a component or even its characteristic will not pose a problem if the modified or new component delivers the same final product. In other words, we can change each component in execution for another component that returns the same result.

In this sense, perhaps now presenting the user as the “player of the game” instead of the “game developer” does not sound so strange. The idea is that the user should spend more time adjusting the game components than developing them. AdLeap-MAS suggests this idea in every design decision made, trying to improve the user experience over this point of view. Directly, the user can change every component in AdLeap-MAS environment, where the only requirement is to keep the same type of results at the end of each component. Figure A.1 presents the user’s flow to simulate a desired problem.

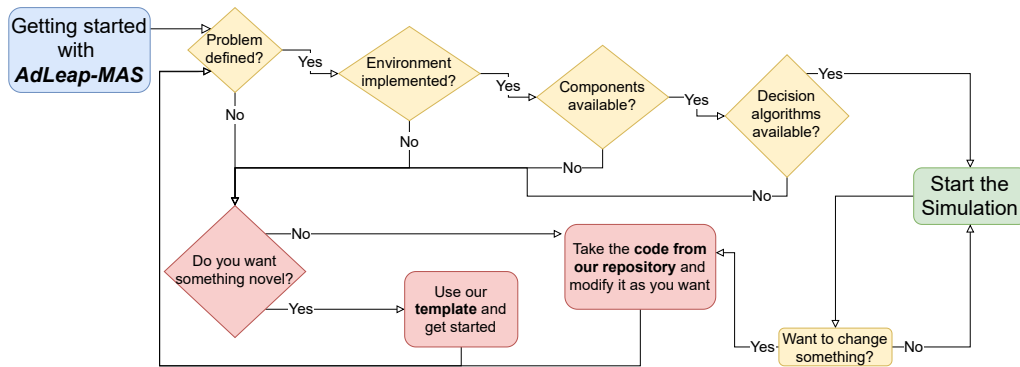


Figure A.1: User’s flow to build, modify and simulate environments in AdLeap-MAS.

Note that we can break the diagram into two groups: the *novel* and the *experiment* group. The users belonging to the novel group will invest some time adapting our base code before starting the simulation. On the other hand, the user in the experiment group can directly access our repository, choose the desired features and plug-in into the framework to start the simulations. However, both groups will spend less time than starting the process from scratch, mainly supported by the base codes. Additionally, this diagram describes the precise cascade scheme tied to our framework, leading the user to avoid repeating previous steps due to misimplementations or wrong modifications.

After defining the context of interest and starting the simulation, the AdLeap-MAS’s workflow will conduct the whole execution requesting the user participation only to fix self-implemented or adapted codes.

A.4.2 Implementation

A.4.2.1 Background

The modelling for Ad-hoc domains is diverse and presents different outcomes in the end. Following the crescent application of reinforcement learning (RL) and planning algorithms to solve problems in this domain, AdLeap-MAS proposes a generic design, applying decision-making Markovian Models and its extensions, denominated the Markov Decision Processes (MDPs). This alternative for modelling represents a relevant approach in the state-of-art [2, 81, 91, 99], fitting different application contexts and well-describing the world for learning and planning algorithms regarding the mathematical representation of the world's information. The MDPs describes the world defining:

- **State set** $s \in \mathbf{S}$: that represents the possible environment configurations.
- **Action set** $a \in \mathbf{A}$: that present the possible agents' actions in the domain.
- **Transition function**: that describes the probability of achieving the state s' from s taking the action a .
- **Reward function**: that define the expected reward value $r \in \mathcal{R}$ given a state s and the performed action a .

Therefore, an MDP model is represented by the tuple $(\mathbf{S}, \mathbf{A}, \mathcal{T}, \mathcal{R})$, which describes how the agents see the world, act and reason about it.

We also allow the user to use extensions of the MDP models. Foremost, we permit the user to design a partially observable environment via the application of a Partially Observable Markov Decision Process (POMDP). The essential difference between the MDP and the POMDP model is the addition of an element that compounds the decision model, which can be described as:

- **Observation set** $z \in \mathbf{Z}$: that delivery partial information from the states.

Observations represents states with concealed information to the agents. Hence, the agents must consider this lack of information to improve the reasoning and the decision-making process. The idea is to build a belief state $b \in \mathbf{B}$ that approximates the real state s and allows the policy development.

From our point of view, the policy represents the decisions taken by the reasoning module of each agent in the simulation, describing indirectly the probability distribution function (pdf) of the actions across the states.

Considering this background, we present the components of AdLeap-MAS.

A.4.2.2 Components Design

Over the mathematical definition of an MDP and its extensions, AdLeap-MAS applies these concepts to design the components from the environment to the decision-making modules. The main idea aims to guarantee the integrity of the code and also robustness for further adaptations, applying strong mathematical properties established by the modelling process. Unfortunately, this imposition decreases the user's freedom to implement different mathematical models. However, it definitely benefits our defined principles and supports the design philosophy.

From this perspective, we designed each component to achieve the final purpose. Consequently, we need to first understand the desired final products for each component. Figure A.2 presents the high-level workflow highlighting the desired output to be carried out at each step of simulation.

In detail, assuming that the environment module is properly implemented over the AdLeap-MAS modelling, this component should pass forward the current observation $z \in \mathbf{Z}$ of the environment. Note that the observation can be total or partial, depending on the user's choice. When the components receive the observation, those who can develop their policy and execute a decision-making process should send forward the current state $s \in \mathbf{S}$ and a valid environment simulator \mathcal{G} . The simulator mentioned here represents the transition function \mathcal{T} of the Markovian model. Since the agents do not have full information on the environment, it is impossible to deliver

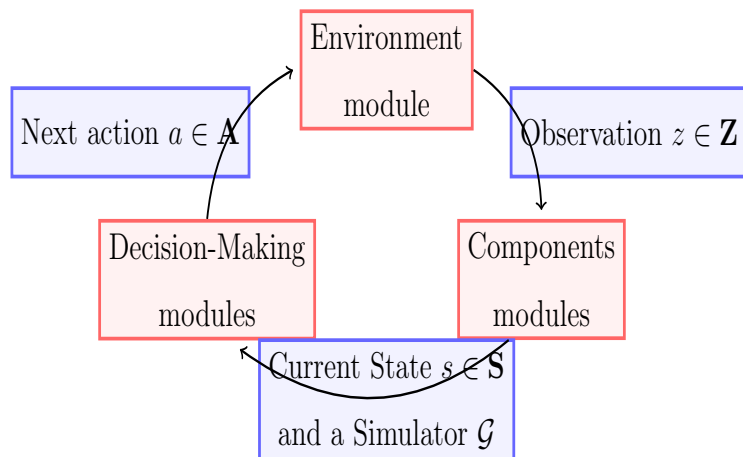


Figure A.2: The AdLeap-MAS high-level workflow, highlighting the information delivered at each step of the simulation. The red squares represent the workflow steps. The blue squares indicate the delivered information.

a correct transition function to them to evaluate the current state. For instance, we may be learning about previously unknown agents in the environment, and hence our transition function would be estimated, given the uncertain models on how these other agents may act. Furthermore, solving real-world problems require surpassing complex contexts and constraints, which turns the explicit modelling of the transition function impossible, facing the infinite uncertainty space. The simulator relaxes this problem by sampling a possible state given the current state and specific action, without need to explicitly define the transition probabilities for every possible state. Therefore, the role of the decision-making module is to evaluate the current state s and choose the best action $a \in \mathbf{A}$ for an agent at the time. This action will return to the environment module, which will commit each arrived action and update the current simulation.

In terms of implementation, Figure A.3 depicts the class diagram in the AdLeap-MAS project. The “AdhocReasoningEnv” class represents our main module for the simulation. It is responsible for managing the components and executing each simulation step using only the required modules. Following the idea presented by Gymnasium, the execution of our basic routine follows description in Algorithm A.1.

Algorithm A.1: AdLeap-MAS basic routine for simulation.

```

env = AdhocReasoningEnv( args )
state = env.reset()
while not done and env.episode < max_episode:
    env.render()
    next_action, _ = type_planning( state, agent )
    state, reward, done, info = env.step( next_action )
env.close()

```

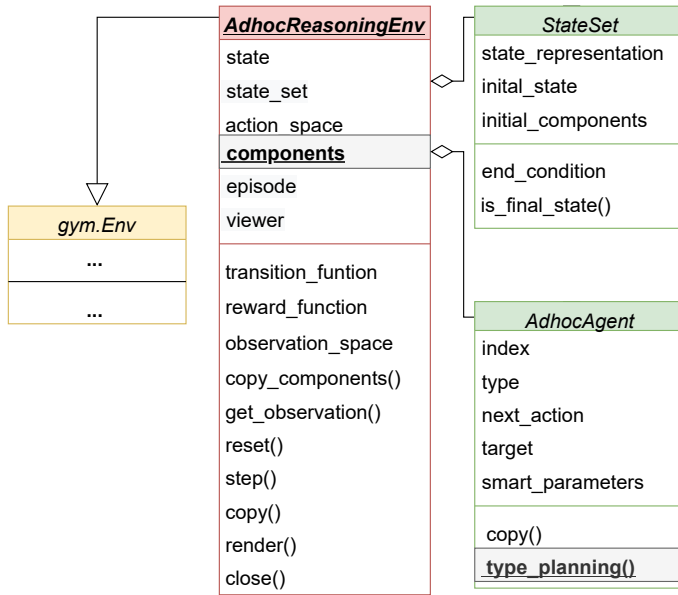


Figure A.3: Class diagram of the AdLeap-MAS project presenting base classes for simulation. The highlighted texts indicate the framework’s main components.

Basically, the routine describes (in pseudo-code) the same high-level workflow presented by Figure A.2, which can be detailed as follows: Given a defined problem and an initial state, the environment module defines, over the Ad-hoc agent point of view (line 2), the current simulation’s state, forwarding this information from the Environment module to the Component module’s level. At this point, the Ad-hoc agent will run its reasoning method, providing the state and the Ad-hoc agent reference to it (line 9).

Note that the reasoning method must evaluate the current state over the ad-hoc agent perspective, hence handles the uncertainties about the environment and its

teammates/adversaries. To support it, the AdLeap-MAS provides the necessary tools to tackle the uncertainties. For instance, our framework enables the embedding of estimation algorithms to the reasoning methods, which allows the sample of generic models used at each planning iteration to approximate the behaviour of the other, previously unknown, agents. So, the reasoning method (at the Decision-Making module's level) will return the next action (line 8) to be performed by the Ad-hoc agent and, finally, the environment will advance for the next step in the simulation (line 12); returning the observation and restarting the process by transferring this new information to the Ad-hoc agent again.

This basic routine can be easily adapted to suit different applications. For instance, considering several ad-hoc agents acting over the same environment, our framework handles this application by changing the perspective every time one component starts to reason (a feature which can be found in the Level-based Foraging environment implementation). On the other hand, if you want to apply a turn-based approach, our framework can change the environment perspective at each step in the environment (a feature that can be found at the Truco environment's code). Note that no modification in the architecture was required to change the problem approach.

Referring to the class diagram, the "AdhocAgent" class represents the agent itself. The foremost important parameter related to this class is its type because it is responsible to define the reasoning method which decides its behaviour (policy) and develop the decision-making process. As a generalisation within our framework, the type of the agent will be similar to the module that implements the reasoning/planning method. For example, if the module which implements the POMCP algorithm has the name "*pomcp.py*", the agent that follows the behaviour defined by the POMCP should have the type "*pomcp*". Therefore, the framework will import the referent module to the current Python execution (simulation) and then start the reasoning/planning.

Following the same generalisation idea to import the module, every module should implement the function "*type_planning*", highlighted in the class diagram inside the "AdhocAgent" class. Considering the example about the POMCP implementation,

this function should have the name “*pomcp_planning*” and receive as argument the state and the agent that called the function. Note that the state here represents the simulator and the state combined, enabling the performance of deterministic and Ad-hoc reasoning methods. It is also important to note that, for partial observable scenarios, the state must conceal the information before advancing to the Component’s level. Our framework implements this concealment process within the function related to the “*observation_space*” that modifies the real state and returns the partially observable one.

Overall, Figure A.4 exhibits our complete workflow, from the user selecting the components to the framework running the simulation procedure.

In the figure, we want to highlight that: **(i)** the user can modify every component in the problem and plug it in the framework without facing hard constraints (limited only by the framework’s model), **(ii)** the environment has its components, which includes the agents, hence includes the reasoning methods (as presented in Figure A.3) but the framework performs these links automatically (with no user interference), and **(iii)** after the problem definition and the establishment of each link, the whole simulation is performed exclusively by the AdLeap-MAS basic routine (as presented

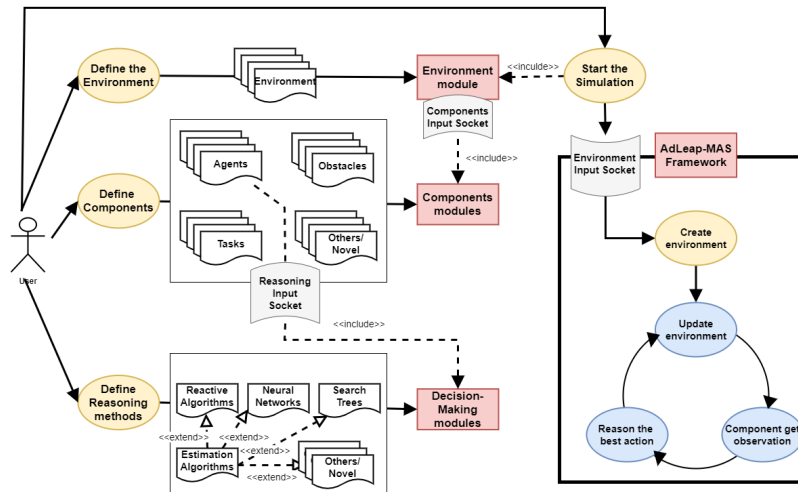


Figure A.4: Complete workflow of AdLeap-MAS framework, from the user definition to the internal simulation procedure.

Algorithm A.2: AdLeap-MAS generic code for components definition.

```
from your_agent_implementation_module import Agent
from your_task_implementation_module import Task
from your_environment_module import Environment

components = {
    'agents': [
        Agent(index='A', atype='reasoning_1'), \
        Agent(index='B', atype='reasoning_2'), \
        Agent(index='C', atype='reasoning_3')
    ],
    'tasks': [Task(index='1'), Task(index='2'), \
              Task(index='3'), Task(index='4')]
}

env = Environment(components)
```

in Figure A.2) based on the user definitions. An example of these concepts on coding is presented in Algorithm A.2.

A.5 Performance and Results

The computer's settings used in the experiments are: (i) Ubuntu 16.04 LTS 64-bit, (ii) 7.7 GiB of RAM, (iii) Intel Core i5-7200U and (iv) Intel HD Graphics 620.

We conduct the performance experiments at the Level-based Foraging environment as it allows us to scale the environment size, the number of agents and other components (such as the tasks) [5]. The purpose of running these experiments is not to surpass other simulators. Instead, we want to illustrate the trends associated with resource usage scalability.

In this way, we propose to run the experiments scaling the number of agents and tasks from 100 to 900, considering steps of 100, carried out in a 100x100 grid.

Additionally, we also scale the environment size as the population grows, it being a squared grid with width and height (considering the 2D space) equals the number of agents. We evaluated the CPU and Memory usage besides the time spent to run 1 step within the simulation. The mean and the standard deviation were calculated over the results of 20 simulations for each configuration.

Firstly, we collected the CPU and Memory usage in simulations scaling the number of agents and tasks from 100 to 900. Figures A.5 and A.6 show the results for CPU and the Memory performance, respectively.

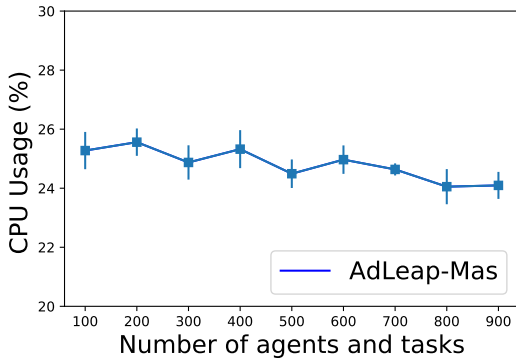


Figure A.5: CPU usage (%) scaling the components in the environment

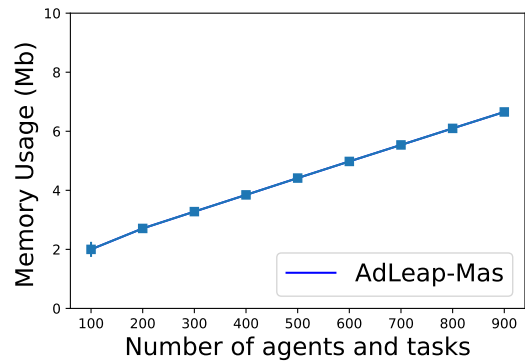


Figure A.6: Memory usage (Mb) scaling the components in the environment.

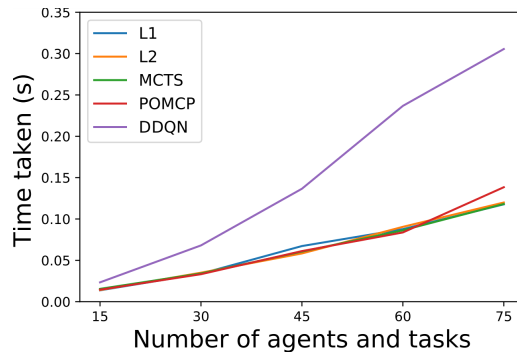


Figure A.7: Time spent in one simulation step for different reasoning methods.

As the image illustrates, the AdLeap-MAS can simulate different environment

settings managing similar CPU usage. On the other hand, the memory used in the application scales almost linearly when scaling the number of agents and tasks. Note that we are only considering the memory used by the environment component, ignoring the additional memory required by additional methods.

For the next experiment, we investigated the time performance of our framework. To collect this data, we evaluate the necessary time to run one step in the simulator while increasing the number of components from 15 to 75, using steps of 15. Additionally, we evaluate the time for the reasoning, considering one ad-hoc agent whose policy is defined by the $L1$, $L2$, $L3$, $L4$, $MCTS$ (UCT), $POMCP$ or $DDQN$ algorithms. Figure A.7 shows the result. The experiments also includes the application of estimation methods within the execution.

We can notice that the reactive algorithms, $L1$, $L2$, $L3$ and $L4$, and both $POMCP$ and $MCTS$ show similar trends for the time scalability. As expected, the reactive algorithms describe a linear trend with the increasing number of agents and tasks since they evaluate the agent’s position to act. The trend related to the $MCTS$ and $POMCP$ algorithms is a consequence of the maximum depth and iteration to perform the search procedure within the tree. Finally, the $DDQN$ algorithm demands much more time to execute one step due to the expensive calculations of convolutions and fully connected layers.

Furthermore, we also want to highlight another significant result related to the high adaptability of OpenAI gym’s codes to run in the AdLeap-MAS. To further illustrate our point, we present in our GitHub, the $DDQN$ implementation for both of our environments. We adjust it from the $DDQN$ implementation² implementation in the “cart pole” environment project for the OpenAI gym. The only modifications we carried out are about enabling the loading of the environment and the network structure. For the Level-based Foraging environment, we also reshape the 2D feature space to be compatible with the 1D observation space of the “cart pole” environment. We successfully managed to re-purpose the implementation from the “cart pole” to

² $DDQN$ implementation: <https://github.com/VXU1230/Medium-Tutorials>

ours using less than 10 (short) code lines, which saves the users' times to maintain the focus on their researches.

A.6 Conclusion

The application of multi-agent systems is recurrently used to tackle daily social problems. Similarly, *Ad-hoc reasoning* models can improve these systems' efficiency, adapting the system while handling the context uncertainties. However, before starting a real-world application, simulators play a crucial role in testing and evaluation. The lack of a centralised platform for code sharing and methods evaluation/comparisons defines an expressive bottleneck to the research advancement, where the workload required for these applications affects the results directly, demanding more time to implement a reliable testing framework than to start the experiments.

Therefore, we presented the AdLeap-MAS, a collaborative workspace for Ad-hoc reasoning applications. Built over a component-based architecture, our framework offers friendly usability, ensuring the modification of the context, reasoning method and environment quickly and efficiently. We employed AdLeap-MAS to simulate two contrasting applications of ad-hoc multi-agent systems. We investigated AdLeap-MAS's overhead and concluded that is possible to perform distinct experiments over a low scalability rate and with different reasoning models. Additionally, the proposed architecture and model delivered a significant minimisation in the implementation workload. The AdLeap-MAS presents high portability for code re-utilisation and the capability to represent diverse problems, addressing a significant problem defined in this document.

AdLeap-MAS is released as open-source for the benefit of others in the multi-agents research community.

References

- [1] Diego Agudelo-España et al. “Bayesian online prediction of change points”. In: *Conference on Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 320–329.
- [2] Stefano V Albrecht and Peter Stone. “Autonomous agents modelling other agents: A comprehensive survey and open problems”. In: *Artificial Intelligence* 258 (2018), pp. 66–95.
- [3] Stefano V. Albrecht, Jacob W. Crandall, and Subramanian Ramamoorthy. “An Empirical Study on the Practical Impact of Prior Beliefs over Policy Types”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI’15. Austin, Texas: AAAI Press, 2015, pp. 1988–1994. ISBN: 0262511290.
- [4] Stefano V. Albrecht and Subramanian Ramamoorthy. “Exploiting Causality for Selective Belief Filtering in Dynamic Bayesian Networks”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. Melbourne, Australia: AAAI Press, 2017, pp. 5085–5089. ISBN: 9780999241103.
- [5] Stefano V. Albrecht and Peter Stone. “Reasoning about Hypothetical Agent Behaviours and Their Parameters”. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’17. São Paulo, Brazil: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 547–555.

-
- [6] Mona Alshehri, Napoleon N Reyes, and Andre LC Barczak. “Evolving Meta-Level Reasoning with Reinforcement Learning and A* for Coordinated Multi-Agent Path-planning.” In: *AAMAS*. 2020, pp. 1744–1746.
- [7] Sara Amini, Maziar Palhang, and Nasser Mozayani. “POMCP-Based Decentralized Spatial Task Allocation Algorithms for Partially Observable Environments”. In: *Applied Intelligence* 53.10 (Sept. 2022), pp. 12613–12631. ISSN: 0924-669X. DOI: 10.1007/s10489-022-04142-7. URL: <https://doi.org/10.1007/s10489-022-04142-7>.
- [8] Mauricio Araya-López et al. “A POMDP Extension with Belief-Dependent Rewards”. In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1*. NIPS’10. Vancouver, British Columbia, Canada: Curran Associates Inc., 2010, pp. 64–72.
- [9] Aijun Bai et al. “Thompson Sampling Based Monte-Carlo Planning in POMDPs”. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. Portsmouth, New Hampshire, USA: AAAI Press, 2014, pp. 29–37. ISBN: 9781577356608.
- [10] Andrea Baisero and Christopher Amato. “Unbiased Asymmetric Reinforcement Learning under Partial Observability”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2022, pp. 44–52.
- [11] Samuel Barrett and Peter Stone. “Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI’15. Austin, Texas: AAAI Press, 2015, pp. 2010–2016. ISBN: 0262511290.
- [12] Samuel Barrett, Peter Stone, and Sarit Kraus. “Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain”. In: *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS ’11. Taipei, Taiwan: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 567–574. ISBN: 0982657161.

- [13] Samuel Barrett et al. “Learning teammate models for ad hoc teamwork”. In: *AAMAS Adaptive Learning Agents (ALA) Workshop*. 2012, pp. 57–63.
- [14] Samuel Barrett et al. “Teamwork with Limited Knowledge of Teammates”. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI’13. Bellevue, Washington: AAAI Press, 2013, pp. 102–108.
- [15] Samuel Barrett et al. “Making friends on the fly: Cooperating with new teammates”. In: *Artificial Intelligence* 242 (2017), pp. 132–171.
- [16] Michael Behrisch et al. “SUMO—simulation of urban mobility: an overview”. In: *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind. 2011.
- [17] Spring Berman et al. “Optimized Stochastic Policies for Task Allocation in Swarms of Robots”. In: *Trans. Rob.* 25.4 (Aug. 2009), pp. 927–937. ISSN: 1552-3098. DOI: 10.1109/TR0.2009.2024997. URL: <https://doi.org/10.1109/TR0.2009.2024997>.
- [18] T. Bösser. “Autonomous Agents”. In: *International Encyclopedia of the Social & Behavioral Sciences*. Ed. by Neil J. Smelser and Paul B. Baltes. Oxford: Pergamon, 2001, pp. 1002–1006. ISBN: 978-0-08-043076-8. DOI: <https://doi.org/10.1016/B0-08-043076-7/00534-9>. URL: <https://www.sciencedirect.com/science/article/pii/B0080430767005349>.
- [19] Craig Boutilier. “Planning, Learning and Coordination in Multiagent Decision Processes”. In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. TARK ’96. The Netherlands: Morgan Kaufmann Publishers Inc., 1996, pp. 195–210.
- [20] Felix Brandt and Martin Bullinger. “Finding and Recognizing Popular Coalition Structures.” In: *AAMAS*. 2020, pp. 195–203.
- [21] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

-
- [22] Rodney A. Brooks. “Intelligence without representation”. In: *Artificial Intelligence* 47.1 (1991), pp. 139–159. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(91\)90053-M](https://doi.org/10.1016/0004-3702(91)90053-M). URL: <https://www.sciencedirect.com/science/article/pii/000437029190053M>.
- [23] Cameron B. Browne et al. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.
- [24] Luca Carminati et al. “Hidden-Role Games: Equilibrium Concepts and Computation”. In: *arXiv preprint arXiv:2308.16017* (2023).
- [25] Matheus Aparecido do Carmo Alves et al. “AdLeap-MAS: An Open-Source Multi-Agent Simulator for Ad-Hoc Reasoning”. In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems. AAMAS '22. Virtual Event, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2022*, pp. 1893–1895. ISBN: 9781450392136. URL: <https://dl.acm.org/doi/10.5555/3535850.3536143>.
- [26] Matheus Aparecido do Carmo Alves et al. “On-line estimators for ad-hoc task execution: learning types and parameters of teammates for effective teamwork”. In: *Autonomous Agents and Multi-Agent Systems* 36.2 (2022), p. 45. ISSN: 1573-7454. DOI: <https://doi.org/10.1007/s10458-022-09571-9>. URL: <https://link.springer.com/article/10.1007/s10458-022-09571-9>.
- [27] Matheus Aparecido do Carmo Alves et al. “Information-guided Planning: An Online Approach for Partially Observable Problems”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. New Orleans Ernest N. Morial Convention Center, Nova Orleans, Louisiana, EUA, Dec. 2023.
- [28] Matheus Aparecido do Carmo Alves et al. “On-Line Estimators for Ad-Hoc Task Execution: Learning Types and Parameters of Teammates for Effective Teamwork”. In: *Proceedings of the 2023 International Conference on*

- Autonomous Agents and Multiagent Systems*. AAMAS '23. London, United Kingdom: International Foundation for Autonomous Agents and Multiagent Systems, 2023, pp. 140–142. ISBN: 9781450394321. URL: <https://dl.acm.org/doi/10.5555/3545946.3598629>.
- [29] Matheus Aparecido do Carmo Alves et al. “It Is Among Us: Identifying Adversaries in Ad-hoc Domains Using Q-valued Bayesian Estimations”. In: *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '24. Accepted for publication. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2024.
- [30] Muthukumaran Chandrasekaran et al. “Team Behavior in Interactive Dynamic Influence Diagrams with Applications to Ad Hoc Teams”. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS '14. Paris, France: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1559–1560. ISBN: 9781450327381.
- [31] Shuo Chen et al. “ATSIS: Achieving the Ad hoc Teamwork by Sub-task Inference and Selection”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 172–179. DOI: 10.24963/ijcai.2019/25. URL: <https://doi.org/10.24963/ijcai.2019/25>.
- [32] Sayak Ray Chowdhury, Rafael Oliveira, and Fabio Ramos. “Active learning of conditional mean embeddings via bayesian optimisation”. In: *Conference on Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 1119–1128.
- [33] Daniel Claes et al. “Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15.

-
- Istanbul, Turkey: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 881–890. ISBN: 9781450334136.
- [34] Aleksander Czechowski and Frans A. Oliehoek. “Decentralized MCTS via Learned Teammate Models”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI’20. Yokohama, Yokohama, Japan, 2021. ISBN: 9780999241165.
- [35] Giuseppe De Giacomo and Yves Lespérance. “Goal Formation through Interaction in the Situation Calculus: A Formal Account Grounded in Behavioral Science.” In: *AAMAS*. 2020, pp. 294–302.
- [36] Prashant Doshi, Yifeng Zeng, and Qiongyu Chen. “Graphical models for interactive POMDPs: representations and solutions”. In: *Autonomous Agents and Multi-Agent Systems* 18 (2009), pp. 376–416.
- [37] Alexis Drogoul et al. “Gama: multi-level and complex environment for agent-based models and simulations”. In: *12th International Conference on Autonomous agents and multi-agent systems*. IFAAMAS. 2013, 2–p.
- [38] Adam Eck et al. “Scalable decision-theoretic planning in open and typed multiagent systems”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05. 2020, pp. 7127–7134.
- [39] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. “Faster game solving via predictive blackwell approachability: Connecting regret matching and mirror descent”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 6. 2021, pp. 5363–5371.
- [40] Dieter Fox. “Adapting the sample size in particle filters through KLD-sampling”. In: *The international journal of robotics research* 22.12 (2003), pp. 985–1003.
- [41] Brian Gerkey, Richard T Vaughan, and Andrew Howard. “The player/stage project: Tools for multi-robot and distributed sensor systems”. In: *Proceedings*

- of the 11th international conference on advanced robotics. Vol. 1. Citeseer. 2003, pp. 317–323.
- [42] Mohammad Ghavamzadeh et al. “Bayesian Reinforcement Learning: A Survey”. In: *Found. Trends Mach. Learn.* 8.5–6 (Nov. 2015), pp. 359–483. ISSN: 1935-8237. DOI: 10.1561/22000000049. URL: <https://doi.org/10.1561/22000000049>.
- [43] Maria Gini. “Multi-robot allocation of tasks with temporal and ordering constraints”. In: *AAAI Conference on Artificial Intelligence*. 2017.
- [44] Piotr J. Gmytrasiewicz and Prashant Doshi. “A Framework for Sequential Planning in Multi-Agent Settings”. In: *J. Artif. Int. Res.* 24.1 (July 2005), pp. 49–79. ISSN: 1076-9757.
- [45] Arthur Guez, David Silver, and Peter Dayan. “Efficient Bayes-Adaptive Reinforcement Learning Using Sample-Based Search”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1025–1033.
- [46] Arthur Guez, David Silver, and Peter Dayan. “Scalable and Efficient Bayes-Adaptive Reinforcement Learning Based on Monte-Carlo Tree Search”. In: *J. Artif. Int. Res.* 48.1 (Oct. 2013), pp. 841–883. ISSN: 1076-9757.
- [47] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.
- [48] Akinobu Hayashi et al. “Reasoning about Uncertain Parameters and Agent Behaviors through Encoded Experiences and Belief Planning”. In: *Artif. Intell.* 280.C (Mar. 2020). ISSN: 0004-3702. DOI: 10.1016/j.artint.2019.103228. URL: <https://doi.org/10.1016/j.artint.2019.103228>.

-
- [49] José Miguel Henrández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. “Predictive entropy search for efficient global optimization of black-box functions”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’14. Montreal, Canada: MIT Press, 2014, pp. 918–926.
- [50] Trong Nghia Hoang and Kian Hsiang Low. “Interactive POMDP Lite: Towards Practical Planning to Predict and Exploit Intentions for Interacting with Self-Interested Agents”. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI ’13. Beijing, China: AAAI Press, 2013, pp. 2298–2305. ISBN: 9781577356332.
- [51] Marcus Hoerger and Hanna Kurniawati. “An on-line POMDP solver for continuous observation spaces”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7643–7649.
- [52] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0262082136.
- [53] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.
- [54] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. “Learning in POMDPs with Monte Carlo Tree Search”. In: *International Conference on Machine Learning (ICML)*. 2017, pp. 1819–1827.
- [55] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. “Bayesian Reinforcement Learning in Factored POMDPs”. In: *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2019.
- [56] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. “Bayesian Reinforcement Learning in Factored POMDPs”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*.

- AAMAS '19. Montreal QC, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 7–15. ISBN: 9781450363099.
- [57] Sunder Ali Khowaja and Parus Khuwaja. “Q-learning and LSTM based deep active learning strategy for malware defense in industrial IoT applications”. In: *Multimedia Tools and Applications* 80.10 (2021), pp. 14637–14663.
- [58] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Proceedings of the 17th European Conference on Machine Learning. ECML'06*. Berlin, Germany: Springer-Verlag, 2006, pp. 282–293. ISBN: 354045375X. DOI: 10.1007/11871842_29. URL: https://doi.org/10.1007/11871842_29.
- [59] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. IEEE. 2004, pp. 2149–2154.
- [60] Kavya Kopparapu et al. “Hidden agenda: a social deduction game with diverse learned equilibria”. In: *arXiv preprint arXiv:2201.01816* (2022).
- [61] Piotr Kozakowski, Mikolaj Pacek, and Piotr Miloś. “Planning and Learning using Adaptive Entropy Tree Search”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022.
- [62] Hanna Kurniawati, David Hsu, and Wee Sun Lee. “SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces”. In: *Robotics: Science and systems*. 2008.
- [63] T.L Lai and Herbert Robbins. “Asymptotically Efficient Adaptive Allocation Rules”. In: *Adv. Appl. Math.* 6.1 (Mar. 1985), pp. 4–22. ISSN: 0196-8858. DOI: 10.1016/0196-8858(85)90002-8. URL: [https://doi.org/10.1016/0196-8858\(85\)90002-8](https://doi.org/10.1016/0196-8858(85)90002-8).
- [64] Marc Lanctot et al. “OpenSpiel: A framework for reinforcement learning in games”. In: *arXiv preprint arXiv:1908.09453* (2019).

-
- [65] Mikko Lauri and Risto Ritala. “Planning for Robotic Exploration Based on Forward Simulation”. In: *Robot. Auton. Syst.* 83.C (Sept. 2016), pp. 15–31. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.06.008. URL: <https://doi.org/10.1016/j.robot.2016.06.008>.
- [66] Kristina Lerman et al. “Analysis of Dynamic Task Allocation in Multi-Robot Systems”. In: *Int. J. Rob. Res.* 25.3 (Mar. 2006), pp. 225–241. ISSN: 0278-3649. DOI: 10.1177/0278364906063426. URL: <https://doi.org/10.1177/0278364906063426>.
- [67] Yen-Chen Lin et al. “Detecting adversarial attacks on neural network policies with visual foresight”. In: *arXiv preprint arXiv:1710.00814* (2017).
- [68] Dennis V Lindley. “On a measure of the information provided by an experiment”. In: *The Annals of Mathematical Statistics* 27.4 (1956), pp. 986–1005.
- [69] Leandro Soriano Marcolino and Luiz Chaimowicz. “No robot left behind: Coordination to overcome local minima in swarm navigation”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2008, pp. 1904–1909.
- [70] Maja J. Matarić, Gaurav S. Sukhatme, and Esben H. Østergaard. “Multi-Robot Task Allocation in Uncertain Environments”. In: *Autonomous Robots* 14.2 – 3 (2003), pp. 255–263.
- [71] Giulio Mazzi et al. “Learning Logic Specifications for Soft Policy Guidance in POMCP”. In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’23. London, United Kingdom: International Foundation for Autonomous Agents and Multiagent Systems, 2023, pp. 373–381. ISBN: 9781450394321.
- [72] Francisco S. Melo and Alberto Sardinha. “Ad Hoc Teamwork by Learning Teammates’ Task”. In: *Autonomous Agents and Multi-Agent Systems* 30.2 (Mar. 2016), pp. 175–219. ISSN: 1387-2532. DOI: 10.1007/s10458-015-9280-x. URL: <https://doi.org/10.1007/s10458-015-9280-x>.

- [73] John Mern et al. “Bayesian optimized Monte Carlo planning”. In: *AAAI Conference on Artificial Intelligence*. Vol. 35. No. 13. 2021, pp. 11880–11887.
- [74] Rubens O. Moraes and Levi H. S. Lelis. “Asymmetric Action Abstractions for Multi-Unit Control in Adversarial Real-Time Games”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI’18/IAAI’18/EAAI’18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.
- [75] Ranjit Nair and Milind Tambe. “Hybrid BDI-POMDP Framework for Multiagent Teaming”. In: *J. Artif. Int. Res.* 23.1 (Apr. 2005), pp. 367–420. ISSN: 1076-9757.
- [76] Ranjit Nair et al. “Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs”. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*. AAAI’05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 133–139. ISBN: 157735236x.
- [77] Laura Niss and Ambuj Tewari. “What You See May Not Be What You Get: UCB Bandit Algorithms Robust to ϵ -Contamination”. In: *Conference on Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 450–459.
- [78] Błażej Osiński et al. “Simulation-based reinforcement learning for real-world autonomous driving”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6411–6418.
- [79] Praveen Palanisamy. “Multi-agent connected autonomous driving using deep reinforcement learning”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–7.
- [80] Georgios Papoudakis, Filippos Christianos, and Stefano Albrecht. “Agent Modelling under Partial Observability for Deep Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* 34 (2021).

- [81] Lukasz Pelcner et al. “Real-Time Learning and Planning in Environments with Swarms: A Hierarchical and a Parameter-Based Simulation Approach”. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1019–1027. ISBN: 9781450375184.
- [82] Carlo Pinciroli. *The swarmanoid simulator*. IRIDIA, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle, 2007.
- [83] Neil Rabinowitz et al. “Machine Theory of Mind”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. ICML. 2018, pp. 4218–4227.
- [84] Arrasy Rahman et al. “Open Ad Hoc Teamwork using Graph-based Policy Learning”. In: *arXiv preprint arXiv:2006.10412* (2020).
- [85] Kambiz Rasoulkhani et al. “Resilience planning in hazards-humans-infrastructure nexus: A multi-agent simulation for exploratory assessment of coastal water supply infrastructure adaptation to sea-level rise”. In: *Environmental Modelling & Software* 125 (2020), p. 104636.
- [86] Francesco Riccio, Roberto Capobianco, and Daniele Nardi. “LoOP: Iterative learning for optimistic planning on robots”. In: *Robotics and Autonomous Systems* 136 (2021), p. 103693.
- [87] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.
- [88] Daniel Russo and Benjamin Van Roy. “Learning to optimize via information-directed sampling”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 1583–1591.

- [89] Paul Scerri, David V. Pynadath, and Milind Tambe. “Towards Adjustable Autonomy for the Real World”. In: *J. Artif. Int. Res.* 17.1 (Sept. 2002), pp. 171–228. ISSN: 1076-9757.
- [90] Elnaz Shafipour and Saber Fallah. “Task-Based Ad-Hoc Teamwork with Adversary”. In: *Towards Autonomous Robotic Systems: 22nd Annual Conference, TAROS 2021, Lincoln, UK, September 8–10, 2021, Proceedings*. Lincoln, United Kingdom: Springer-Verlag, 2021, pp. 76–87. ISBN: 978-3-030-89176-3. DOI: 10.1007/978-3-030-89177-0_8. URL: https://doi.org/10.1007/978-3-030-89177-0_8.
- [91] Elnaz Shafipour Yourdshahi et al. “On-Line Estimators for Ad-Hoc Task Allocation”. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '20*. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1999–2001. ISBN: 9781450375184.
- [92] Devavrat Shah, Qiaomin Xie, and Zhi Xu. “Non-asymptotic analysis of Monte Carlo Tree Search”. In: *Abstracts of the SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*. 2020, pp. 31–32.
- [93] David Silver and Joel Veness. “Monte-Carlo Planning in Large POMDPs”. In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2. NIPS'10*. Vancouver, British Columbia, Canada: Curran Associates Inc., 2010, pp. 2164–2172.
- [94] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), pp. 354–359.
- [95] Peter Stone et al. “Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. AAAI'10*. Atlanta, Georgia: AAAI Press, 2010, pp. 1504–1509.

-
- [96] Zachary N Sunberg and Mykel J Kochenderfer. “Online algorithms for POMDPs with continuous state, action, and observation spaces”. In: *International Conference on Automated Planning and Scheduling*. 2018.
- [97] James Stuart Tanton. *Encyclopedia of mathematics*. Infobase Publishing, 2005. ISBN: 9780816051243.
- [98] Khadija Tazi, Fouad Mohamed Abbou, and Farid Abdi. “Multi-agent system for microgrids: design, optimization and performance”. In: *Artificial Intelligence Review* 53.2 (2020), pp. 1233–1292.
- [99] Vincent Thomas, G er emy Hutin, and Olivier Buffet. “Monte Carlo Information-Oriented Planning”. In: *European Conference On Artificial Antelligence*. 2020.
- [100] Alejandro Torre no et al. “Cooperative Multi-Agent Planning: A Survey”. In: *ACM Comput. Surv.* 50.6 (Nov. 2017). ISSN: 0360-0300. DOI: 10.1145/3128584. URL: <https://doi.org/10.1145/3128584>.
- [101] Mark Towers et al. *Gymnasium*. URL: <https://github.com/Farama-Foundation/Gymnasium>.
- [102] Maulesh Trivedi and Prashant Doshi. “Inverse Learning of Robot Behavior for Collaborative Planning”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain: IEEE Press, 2018, pp. 1–9. DOI: 10.1109/IROS.2018.8593745. URL: <https://doi.org/10.1109/IROS.2018.8593745>.
- [103] Richard Vaughan. “Massively multi-robot simulation in stage”. In: *Swarm intelligence* 2.2 (2008), pp. 189–208.
- [104] Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. “An informational approach to the global optimization of expensive-to-evaluate functions”. In: *J. of Global Optimization* 44.4 (Aug. 2009), pp. 509–534. ISSN: 0925-5001. DOI: 10.1007/s10898-008-9354-2. URL: <https://doi.org/10.1007/s10898-008-9354-2>.

- [105] Kai Wang et al. “Scalable Game-Focused Learning of Adversary Models: Data-to-Decisions in Network Security Games”. In: *AAMAS*. 2020, pp. 1449–1457.
- [106] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. “Dynamic task allocation for multi-robot search and retrieval tasks”. In: *Applied Intelligence* 45 (2016), pp. 383–401.
- [107] James R. Wright and Kevin Leyton-Brown. “A Formal Separation Between Strategic and Nonstrategic Behavior”. In: *Proceedings of the 21st ACM Conference on Economics and Computation*. EC '20. Virtual Event, Hungary: Association for Computing Machinery, 2020, pp. 535–536. ISBN: 9781450379755. DOI: 10.1145/3391403.3399525. URL: <https://doi.org/10.1145/3391403.3399525>.
- [108] Chenjun Xiao et al. “Maximum entropy Monte-Carlo planning”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [109] Amulya Yadav et al. “Please Be an Influencer? Contingency-Aware Influence Maximization”. In: *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018, pp. 1423–1431.
- [110] Nan Ye et al. “DESPOT: Online POMDP Planning with Regularization”. In: *J. Artif. Int. Res.* 58.1 (Jan. 2017), pp. 231–266. ISSN: 1076-9757.
- [111] Elnaz Shafipour Yourdshahi et al. “Towards large scale ad-hoc teamwork”. In: *International Conference on Agents (ICA)*. IEEE, 2018, pp. 44–49.
- [112] Weigui Jair Zhou et al. “Hierarchical Control of Multi-Agent Reinforcement Learning Team in Real-Time Strategy (RTS) Games”. In: *Expert Syst. Appl.* 186.C (Dec. 2022). ISSN: 0957-4174. DOI: 10.1016/j.eswa.2021.115707. URL: <https://doi.org/10.1016/j.eswa.2021.115707>.
- [113] Maddalena Zuccotto, Alberto Castellini, and Alessandro Farinelli. “Learning State-Variable Relationships for Improving POMCP Performance”. In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. SAC '22. Virtual Event: Association for Computing Machinery, 2022,

pp. 739–747. ISBN: 9781450387132. DOI: 10.1145/3477314.3507049. URL:
<https://doi.org/10.1145/3477314.3507049>.