Coping with uncertainties in agent cooperation

Chen, Shuo

2020

Chen, S. (2020). Coping with uncertainties in agent cooperation. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/137074>

<https://doi.org/10.32657/10356/137074>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 22:23:50 SGT

Coping with Uncertainties in Agent Cooperation

Shuo Chen

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

Sep. 17, 2019

.....

Date

.....

Shuo Chen

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

Sep. 30, 2019

.....

Date

.....

Prof. Jie Zhang

Authorship Attribution Statement

This thesis contains material from 3 papers accepted at conferences in which I am listed as an author.

Chapter 4 are published as [Shuo Chen, Athirai A. Irissappane, and Jie Zhang](#). POMDP-Based Decision Making for Fast Event Handling in VANETs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4646-4653, 2018.

The contributions of the co-authors are as follows:

- I designed the POMDP model, proposed the learning algorithm, conducted the experiments, and prepared the manuscript.
- Athirai A. Irissappane reviewed the technical details and revised the manuscript.
- Jie Zhang proposed the idea, reviewed the technical details and revised the manuscript.

Chapter 5 are published as [Shuo Chen, Ewa Andrejczuk, Athirai A. Irissappane, and Jie Zhang](#). ATSSIS: Achieving the Ad hoc Teamwork by Sub-task Inference and Selection. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 172-179, 2019.

The contributions of the co-authors are as follows:

- I proposed the idea, designed the model, proposed the algorithm, conducted the experiments, and prepared the manuscript.
- Ewa Andrejczuk reviewed the technical details and revised the manuscript.
- Athirai A. Irissappane reviewed the technical details and revised the manuscript.
- Jie Zhang reviewed the technical details and revised the manuscript.

Chapter 6 are published as [Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang](#). AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages XX-XX, 2020.

The contributions of the co-authors are as follows:

- I proposed the idea, designed the model, proposed the algorithm, conducted the experiments, and prepared the manuscript.

- Ewa Andrejczuk reviewed the technical details and revised the manuscript.
- Zhiguang Cao proofread the Introduction section.
- Jie Zhang reviewed the technical details and revised the manuscript.

Sep. 19, 2019

.....

Date

.....

Shuo Chen

Acknowledgements

I want to thank the people who have accompanied me on my PhD journey. First and foremost, I wish to express my sincerest gratitude to my supervisor, Professor Jie Zhang, who has taught me how to do the research. His guidance on identifying important problems, finding solutions, and expressing ideas to general audiences has helped me to grow as an independent researcher. I also got lots of inspirations from his ways of managing projects, which is invaluable for my future career.

Special thanks go to my collaborator Ewa Andrejczuk who has spent numerous efforts on helping me to prepare the research papers. Her high standard has improved my writing skills substantially. Many thanks go to my senior Athirai A. Irissappane who has provided useful advice to my research. Her advice has guided me through the first year of my PhD. I also would like to thank Professor Rongxing Lu who has encouraged me to pursue a PhD degree and introduced me to my supervisor.

I need to thank my friends who have provided many supports that make my PhD study colorful and memorable. Thanks go to Zehong Hu and Wen Song for their kindly sharing of knowledge. Thanks go to Donghun Kang and Zhenchao Bing for their support on project matters. Thanks go to Xiaoming Li for organising leisure activities to release pressure. Thanks go to Kwangsing Yew, Soonhao Yeow and Shaomeng Wang for playing sports together, which keeps my fitness.

Last but not least, I must thank my beloved family for their continued support. They are my constant motivation and inspiration.

To my dear family

Abstract

In multi-agent systems, intelligent agents interact with one another to achieve either individual or shared goals. This thesis focuses on scenarios where agents cooperate to achieve their goals. In particular, agents can help one another by information sharing or collaboration. Thus, agent cooperation is usually beneficial for the agent performance. However, there exist uncertainties that can jeopardise the successful achievement of goals. This thesis tackles the problems that arise from those uncertainties.

The first problem emerges when agents pursue individual goals. Agents can share the information of system states with one another to make more informed decisions while pursuing their goals. However, agents may send false information to mislead others and thus, increase their own benefits. Therefore, there is uncertainty about whether the information shared by other agents is trustworthy or not. The existing approaches use trust management schemes to compute the trustworthiness of shared information. However, they only focus on the accuracies of trustworthiness without considering the cost/delay incurred during the trust computation. This thesis proposes a partially observable Markov decision process (POMDP) model. The model queries the information about uncertain states from neighbouring agents while taking into account their potential malicious behaviours. We also propose an algorithm to learn model parameters in the dynamic scenario where malicious agents change their behaviour from time to time. Experimental results demonstrate that our model can effectively balance the decision quality and response time while still being robust to sophisticated malicious attacks.

When goals are too complex for a single agent, agents can compose teams and coordinate their actions to achieve them. There, the uncertainty arises when agents cannot communicate or share team strategies with their teammates. Specifically, an agent has to understand the behaviour of its teammates and plan its actions accordingly. Here, the behaviour amounts to a function that takes a state as input and outputs an action that can further change the state. Note that the agent

can only observe its teammates' actions or state changes instead of their underlying behaviours. Thus, there is uncertainty about teammates' behaviours. Agents performing the teamwork without relying on communication or team strategies is called the *ad hoc teamwork*. We refer to the domains where the agent can fully observe its teammates' actions as *simple domains*. We refer to the domains where teammates' actions are partially observable as *complex domains*. There, the agent can only rely on its teammates' state changes.

For ad hoc teamwork in simple domains, the existing approaches use teammates' behaviour models to predict their actions and choose the ad hoc agent's action accordingly. However, the behaviour models may not be accurate, which can compromise teamwork. In this thesis, we propose Ad Hoc Teamwork by Sub-task Inference and Selection (ATSIS) algorithm that uses a sub-task inference without relying on teammates' models. First, the ad hoc agent observes its teammates to infer which sub-tasks they are handling. Based on that, it selects its own sub-task using a POMDP model that handles the uncertainty of the sub-task inference. Last, the ad hoc agent uses the Monte Carlo tree search (MCTS) to find the set of actions to perform the chosen sub-task. Experiments demonstrate that ATSIS achieves the teamwork robustly. Also, ATSIS makes much faster decisions than state-of-the-art schemes, which is significant for time-sensitive tasks. Moreover, ATSIS can further improve its performance by integrating the learned model.

For ad hoc teamwork in complex domains, the most advanced approach learns policies based on previous experiences and reuses one of the policies to interact with new teammates. However, the selected policy in many cases is sub-optimal. Switching between policies to adapt to new teammates' behaviour takes time, which threatens the successful performance of a task. In this thesis, we propose Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism (AATEAM) algorithm that uses the attention-based neural networks to cope with new teammates' behaviour in real-time. We train one attention network per teammate type. The attention networks learn both to extract the temporal correlations from the sequence of states (i.e. contexts) and the mapping from contexts to actions. Each attention network also learns to predict a future state given the current context and its output action. The prediction accuracies help to determine which actions the ad hoc agent should take. Experimental results indicate that when working with both known and unknown teammates, in most cases our algorithm outperforms

the most advanced approach. This demonstrates that AATEAM can adapt to new teammates' changing behaviour faster than the state-of-the-art.

Contents

Acknowledgements	ix
Abstract	xiii
List of Figures	xxi
List of Tables	xxiii
1 Introduction	1
1.1 Motivation and Solutions	4
1.1.1 Information Sharing in Multi-agent Systems	4
1.1.2 Ad Hoc Teamwork in Simple Domains	6
1.1.3 Ad Hoc Teamwork in Complex Domains	7
1.2 Summary of Contributions	8
1.3 Organization of the Thesis	10
2 Literature Review	13
2.1 Dealing with Untrustworthy Information in Multi-agent Systems . .	13
2.2 Ad Hoc Teamwork in Simple Domains	16
2.3 Ad Hoc Teamwork in Complex Domains	18
2.4 Summary	19
3 Background	21
3.1 MDP and POMDP	21
3.1.1 Markov Decision Process	21
3.1.2 Partially Observable Markov Decision Process	24
3.1.3 Multi-agent Goal-oriented Markov Decision Process	25
3.2 Bayesian Inference for Model Parameters	26
3.3 Monte Carlo Tree Search	27
3.4 Deep Neural Network	30
3.4.1 Recurrent Neural Network	31
3.4.2 Attention Mechanism	32
3.5 Summary	33

4	POMDP-based Decision Making for Using Information in Multi-agent Systems	35
4.1	System Architecture	36
4.2	Proposed POMDP Model	38
4.2.1	Model Design	38
4.2.1.1	State	38
4.2.1.2	Action	39
4.2.1.3	State Transition Function	40
4.2.1.4	Reward Function	40
4.2.1.5	Observation	40
4.2.1.6	Observation Function	41
4.2.2	Belief Update	41
4.2.3	Learning the Observation Function	42
4.2.4	Model Analysis	46
4.3	Experiments	47
4.3.1	Simulation Setup	47
4.3.2	Malicious Behaviours	48
4.3.3	Static Behaviour Scenario	49
4.3.4	Dynamic Behaviour Scenario	51
4.4	Summary	53
5	ATSIS: Achieving Ad hoc Teamwork by Sub-task Inference and Selection	55
5.1	Background	56
5.2	ATSIS Algorithm	57
5.2.1	Inferring Sub-tasks Pursued by Teammates	57
5.2.2	Selecting the Sub-task by POMDP	58
	State.	59
	Action.	59
	Transition function.	59
	Observation.	59
	Observation function.	60
	Reward function.	61
5.2.3	Pursuing the Chosen Sub-task	62
5.2.4	Complexity Analysis	62
5.3	Experiments	63
	Pursuit domain.	64
5.3.1	Influence of the Acquired Behaviour Models	66
5.3.2	Teamwork with Learning Period	68
5.3.3	The Effects of Transfer Learning	69
5.3.4	Improving ATSIS with the Learned Model	69
5.3.5	The Effects of POMDP Parameters	70
5.4	Summary	72

6	AATEAM: Achieving Ad Hoc Teamwork by Employing the Attention Mechanism	75
6.1	Ad Hoc Teamwork with Attention Mechanism	77
6.1.1	The Design of the Attention Network in AATEAM	77
6.1.1.1	The encoder.	77
6.1.1.2	The decoder.	78
6.1.2	Selecting the Action in AATEAM	80
6.1.3	Training AATEAM	81
6.1.4	Comparison to the NLP domain.	82
6.1.5	Complexity Analysis	83
6.2	Experiments	83
6.2.1	Half Field Offence Domain	84
6.2.1.1	Collecting data for AATEAM.	86
6.2.1.2	Collecting data for PLASTIC-Policy.	86
6.2.2	Experimental Parameters	87
6.2.3	Playing with Past Teammates	88
6.2.4	Playing with New Teammates	89
6.2.5	The Importance of Attention Mechanism	90
6.3	Summary	91
7	Conclusion and Future Work	93
7.1	Conclusion	93
7.2	Future Work	96
	List of Author's Publications	99
	Bibliography	101

List of Figures

1.1	Two dimensions of the interactions in multi-agent systems	2
3.1	Model the interaction between the world and agents by MDP	22
3.2	Model the interaction between the world and agents by POMDP . .	24
3.3	The demonstration of MCTS [1]	28
3.4	The general architecture of deep neural networks [2]	30
3.5	The illustration of GRU [58]	32
3.6	The attention network for translation	33
4.1	Performance evaluation under static behaviour	50
4.2	The warm up period in POMDP ($W = 60s$)	51
4.3	Decision quality under dynamic behaviour	52
4.4	Cost under dynamic behaviour	53
5.1	The pursuit domain	65
5.2	Performance with acquired model	66
5.3	Performance under different learning periods	68
5.4	Performance under transfer learning	69
5.5	Performance when equipping the learned model	70
5.6	Performance under different η	72
6.1	The attention network for ad hoc teamwork	78
6.2	The detailed design of the attention network	78
6.3	The illustration of opening angles [24]	85
6.4	Performance with known teammates	88
6.5	Performance with unknown teammates	90
6.6	The loss of different training cases	91

List of Tables

5.1	Experimental parameters	66
5.2	Performance under different r_1 when $r_2 = 4$	71
5.3	Performance under different μ when $\eta = 1$	71
6.1	Experimental parameters	87

Chapter 1

Introduction

An intelligent agent is an autonomous entity that is situated in an environment and is at least to some extent capable of determining its actions in that environment to meet its design objectives [3]. With the increasing number of intelligent agents (such as autonomous applications in mobile phones, vehicles, recommendation software, or even robot assistants) developed by companies, there emerge a lot of interactions among those agents. Systems composed of multiple interacting intelligent agents are called multi-agent systems. As shown in Figure 1.1, we identify two dimensions of the interactions in multi-agent systems.

The first dimension is about the purpose of the interactions. That is, agents can interact to achieve either shared goals or individual goals. On one hand, agents can share the goal. This can happen for instance, when the goal is too complex for one agent to achieve. In this case, agents can compose a team and coordinate their actions to fulfil the common goal (i.e. perform the teamwork). On the other hand, agents can have individual goals. This happens when a company uses multiple agents to solve different tasks or when agents are owned by different stakeholders. Even though agents have different goals, they still share the environment. This environment is dynamic as the actions of each agent change the state of the environment. Hence, agents may not have full knowledge about the changes. One of the methods for agents to obtain knowledge is to exchange the information with one another. This motivates the second dimension of the interactions. That is, whether agents can share information with one another during their interactions.

FIGURE 1.1: Two dimensions of the interactions in multi-agent systems

In this thesis, we consider the information sharing as an intentional act. That is, the agents which share the information intend to convey the information to the agents which can receive and understand the shared information. Therefore, we define that one agent shares information with another agent only when both agents have a consensus on how to interpret that information. Henceforth, even though an agent observes peer actions, there is no information shared explicitly with the agent if it cannot understand the underlying meaning or intention of those actions. The basic method of sharing information is communication. When agents have incompatible communication protocols or the communication channels are congested, the communication is infeasible. As for teamwork, agents have an additional way to share information, i.e. distributing a pre-designed team strategy a priori.

Given these two dimensions, we split the interactions in multi-agent systems into four scenarios. The first scenario is when agents share information while pursuing individual goals¹. This is desirable for many real-life applications as agents often have incomplete information of the system states. By sharing information, agents can estimate the system states that they cannot directly observe and make more informed decisions [4]. The second scenario is when agents share information while coordinating their actions to achieve a shared goal. This scenario has been

¹Note that there exist cases where individual goals are conflicting. However, conflicting goals lead to competition instead of cooperation, and thus those cases are out of the scope of this thesis.

studied for decades (e.g. [5, 6]). In the remaining of this thesis, we refer to this scenario as *traditional teamwork*. The third scenario is when agents do not share any information during the teamwork. This matches the practical situations where agents are produced by different parties or in different time periods, and thus do not share the same communication protocol or team strategy with one other [7]. Agents performing the shared task together without relying on communication and team strategy is called the *ad hoc teamwork* [7]. The fourth scenario is when agents having different goals do not share any information with one another. In this scenario, agents could consider others as a part of the environment or model others as opponents and focus on their tasks.

In the first three scenarios, agents can cooperate to achieve their goals. We refer to the cooperation as *agent cooperation*. However, there exist challenges that can hinder the cooperation. This thesis tackles those challenges. Moreover, there are already plenty of approaches proposed for the traditional teamwork where teammates either communicate or share team strategy with one another. Henceforth, this thesis concentrates on the first and third scenarios. That is, 1) agents having different goals share information with one another and 2) ad hoc teamwork.

As for the information sharing, the agents can be owned by different stakeholders, and thus, have different interests or design objectives. In this case, agents may send false information to increase their own benefits. For example, in vehicular ad-hoc networks (VANETs), an agent may report a non-existent accident to make other agents avoid a particular road such that it can choose that road to arrive faster at its own destination. Therefore, there is uncertainty about whether the information shared by other agents is trustworthy or not.

When it comes to ad hoc teamwork, agents cannot rely on the communication or shared strategy. Agents have to understand the behaviour of teammates and plan their actions accordingly to maximise the team utility. In this thesis, we define the behaviour of an agent as a function that maps states to actions. Note that an agent can only observe its teammates' actions or the results of those actions instead of the underlying behaviours. That is, teammates' behaviours are partially observable. Therefore, there is uncertainty about teammates' behaviours.

In this thesis, we propose three approaches to cope with the above-mentioned uncertainties in agent cooperation. First, we deal with the uncertainty about the

trustworthiness of shared information. Second, we solve the ad hoc teamwork problem in simple domains. That is, the scenarios have fully observable action space. Therefore, the agent can understand teammates' behaviour by observing their actions. Last, we achieve ad hoc teamwork in complex domains. In complex domains, the teammates' actions are only partially observable. Henceforth, the agent can understand teammates' behaviour only by observing their state changes.

1.1 Motivation and Solutions

In this subsection, we describe how the existing approaches in the literature try to solve the above-mentioned challenges and discuss their drawbacks that motivate this thesis. Then, we present our approaches that facilitate the agent cooperation in multi-agent systems.

1.1.1 Information Sharing in Multi-agent Systems

In multi-agent systems, agents can exchange messages about the system states to help one another gain more information about the system. For example, vehicles in VANETs can query other vehicles about road situation. Buyers in electronic commerce (E-commerce) can query other buyers about the quality of sellers. Given this information, agents can better respond to the environment, i.e. taking actions based on the reported system state. However, there exist selfish/malicious agents that share false information to mislead other agents. For instance, a vehicle in VANETs may send fake accident information of a certain road so that other vehicles will choose other roads and it can drive on that road more comfortably and safely. Buyers in E-commerce may provide untrue comments about a certain seller in order to get discounts from that seller. Trust management schemes [8–12] address this issue by computing the trustworthiness of agents and thereby, the trustworthiness of exchanged messages. Such information helps to create and update agents' beliefs on the system states. The belief on a system state (called *state belief*, hereon) is the probability distribution over the possible values of the system state. Existing trust schemes only focus on trust computation but neglect the problem of making decisions about whether actions towards completing a given task can be taken based on the shared information. Thereby, a straightforward way is using a simple

heuristic to make decisions based on the computed trustworthiness. That is, they decide to believe the system state as a certain value if the probability of that value in the state belief exceeds a given threshold, and take actions accordingly. Often, thresholds are set to a larger value to ensure that the shared information is accurate. This requires exchanging more messages that results in higher consumption of communication and computation resources, as well as longer information gathering period (that leads to delayed responses and additional cost). However, agents often need to identify accurate information quickly and take actions accordingly. Therefore, the threshold-based approaches that often result in late response for guaranteeing the accuracy of information are insufficient.

To achieve fast response without sacrificing decision quality, we propose a partially observable Markov decision process (POMDP) based approach. POMDP provides a natural framework for sequential decision making under uncertainty. Thus, it is an ideal choice for multi-agent systems to handle the uncertainty introduced due to the unreliable information from selfish/malicious agents. POMDP effectively balances the trade-off between exploitation (taking actions based on the shared information) and exploration (collecting more information about the system state) [13].

Our POMDP model maintains the belief about the system state's value. It gathers information about the system state by querying neighbouring agents (which can be trustworthy/untrustworthy in providing information) and updates the state belief based on its observation function. The observation function returns the probability of receiving the shared information given the system state. The more trustworthy neighbouring agents are, the higher probability of the shared information matching the system state. Therefore, the observation function represents the behaviour of neighbouring agents. Based on the state belief, the POMDP model assists an agent on deciding whether to take actions based on the reported system state or wait for more information. It does so, by balancing the expected benefit of taking actions given the state belief and the delay/cost of querying for more information, thereby maximising the expected total utility for the agent. Additionally, we propose to learn the observation functions for the POMDP model so that our approach is more realistic. In our scenario, learning is more complex as the behaviour of neighbouring agents can change over time. This is because of sophisticated selfish/malicious agents who can change their behaviour periodically. We integrate our POMDP

model with the Bayesian inference technique to cope with the potentially untrustworthy information in an efficient and robust way.

1.1.2 Ad Hoc Teamwork in Simple Domains

In ad hoc teamwork, an agent (called ad hoc agent hereon) engages in collaborative tasks with other (unknown) teammates without relying on communication or pre-defined team strategy [7]. For example, in search and rescue missions, robots deployed by different companies do not share a team strategy. Moreover, the communication channel is often unavailable in the disaster environment. The ad hoc agent needs to understand the teammates' behaviour by observation and plan its actions to perform the task. Since the action space is fully observable in simple domains, the ad hoc agent can understand the teammates' behaviour by observing their actions given different states.

We can divide the existing works relevant to ad hoc teamwork in simple domains into two approaches. The first approach does not consider the uncertainty brought by the dynamism of the environment. For example, in [14–16] Agmon and Stone, Chakraborty and Stone, and Agmon et al. assume that the utilities of the team's actions are known. Similarly, Genter et al. [17] consider the role assignment problem where they know which roles are assigned to which teammates. In real life, the utilities of the team's actions as well as the teammates' roles may be changing due to the changing environment. Thus, the assumption that they are known is too simplistic. The second approach [18–23] considers the uncertainty of the environment. They represent their problem domain as Markov decision processes (MDPs). The MDP provides a framework for the ad hoc agent to select optimal actions based on its teammates' actions. Then, the second approach assumes that the ad hoc agent can predict teammates' actions based on teammates' behaviour models. However, the teammates can have a very different behaviour compared to the acquired models. Therefore, the chosen action may be far from the optimal one. The ad hoc agent can also learn its teammates' behaviour models based on their current actions (as in [20]). However, in time-sensitive tasks, there is not enough time to learn the accurate behaviour models. Therefore, there is a need to devise an approach that does not rely on teammates' models.

In this thesis, we propose the *Ad hoc Teamwork by Sub-task Inference and Selection (ATSIS)* algorithm. To the best of our knowledge, this is the first algorithm in the ad hoc teamwork domain that does not rely on teammates' behaviour models. In detail, we assume that the collaborative task is a set of disjoint sub-tasks. The ad hoc agent observes its teammates' actions and infers which sub-tasks they are performing based on whether their actions help to fulfil a certain sub-task. Based on the inference of teammates' sub-tasks, the ad hoc agent further selects the unattended sub-task as its own sub-task. Note that in some situations, the ad hoc agent cannot be certain which sub-tasks its teammates are performing. For instance, an agent moving to a certain direction may either try to rescue a person or search for other victims. To handle these uncertain situations, we model the ad hoc agent's sub-task selection as a POMDP. The inference of teammates' sub-tasks is regarded as the observations for updating the belief about which sub-task each teammate is handling. We also maintain the belief about how suitable the ad hoc agent is for each sub-task when compared to its teammates. When there are multiple sub-tasks which seem to be unattended, this belief leads the ad hoc agent to choose the most suitable sub-task. Based on the beliefs, POMDP suggests the sub-task the ad hoc agent should handle. Finally, the ad hoc agent uses Monte Carlo tree search (MCTS) to find the actions for pursuing the selected sub-task.

1.1.3 Ad Hoc Teamwork in Complex Domains

Most of the existing approaches for ad hoc teamwork [14–23] focus on simple domains. To the best of our knowledge, PLASTIC-Policy [24] is the only scheme proposed for more complex domains. In complex domains, the action space is partially observable. For instance, given a continuous action space, the ad hoc agent cannot know the exact value of the actions performed by its teammates. Also, two actions with different intentions can look similar in some situations. For example, in the robosoccer domain, shooting is indistinguishable to passing to the teammate that is close to the goal. Henceforth, the ad hoc agent can only understand teammates' behaviour based on their state changes. Additionally, due to partially observable actions, there is no training data for learning teammates' behaviour models that output teammates' actions given a state. To cope with these challenges, PLASTIC-Policy learns a policy for each past teammate type. When working with new teammates, it chooses one of the policies based on the

similarity between the new and past experiences and uses that policy to select the sequence of actions for the ad hoc agent. However, the selected policy only works well when new teammates behave similarly as the chosen teammate type. When new teammates' behaviour becomes very different from the teammate type, the ad hoc agent needs to switch to another policy. Switching between policies to adapt to new teammates' behaviour takes time, which threatens the successful performance of a task. Given that, it is desirable to develop a new method that adapts to new teammates' behaviours in real-time for complex domains.

In this thesis, we propose to *Achieve Ad hoc Teamwork by Employing the Attention Mechanism* (AATEAM). AATEAM uses attention-based recurrent neural networks to cope with new teammates' behaviour in real-time. Our workflow is as follows. First, we use reinforcement learning to learn policies to work with past teammates. Second, we make the ad hoc agent play with each type of teammate using the corresponding learned policy. We collect the states (*input*) and the ad hoc agent's actions (*label*) as training data. Third, we use this data to teach one attention network for each type of past teammates. The attention network works as follows. In every step, the attention network takes a state as input. The attention mechanism inside extracts the temporal correlation between the state and the states encountered previously, i.e. *context*. It further maps the context to an action probability distribution. In every step, the attention networks also output a metric that measures the similarity between past and new teammate types. The more similar the types, the more important the corresponding probability distribution over actions. We aggregate networks' probability distributions over actions by calculating the weighted sum of those probabilities and its importance. Finally, we output an action with the highest aggregated probability. Since AATEAM evaluates the similarity in every step, we can adjust to the new teammates' behaviour in real-time.

1.2 Summary of Contributions

In summary, we make the following contributions in this thesis:

- To the best of our knowledge, this is the first time a POMDP is applied to the decision problem for the information sharing in agent cooperation. Based

on the received information, the ad hoc agent decides if it is confident enough to make a decision or it should gather more information. Since the POMDP model can effectively balance the trade-off between the decision quality and response delay/cost, it can handle the issue of false information. Further, we design the model to have a small state space. Henceforth, it is scalable to a large number of agents. Moreover, we also propose an algorithm based on Bayesian inference to learn the observation function in dynamic scenarios where selfish agents change their malicious behaviour frequently. To evaluate the performance of our model, we conduct extensive experiments in the domain of VANETs. Results demonstrate that our solution can make better and fast decisions while still being robust to frequent behaviour changes.

- For ad hoc teamwork in simple domains, we propose a new algorithm, called ATISIS that solves the ad hoc teamwork problem without relying on teammates' behaviour models. To the best of our knowledge, using the sub-task inference to solve ad hoc teamwork was never studied before. ATISIS consists of three parts: a) It infers which sub-tasks the team members are performing by observing if the teammates' actions help to fulfil those sub-tasks; b) It uses a POMDP model to select the sub-task for the ad hoc agent given the uncertainty of the environment; c) It uses a variant of the MCTS algorithm to find the actions to perform the chosen sub-task. On top of the high-level termination condition, i.e. achieving the task (as in previous approaches [20]), our version of MCTS has another termination condition, that is, achieving the sub-task. This allows the ad hoc agent to make faster decisions. We perform extensive experiments in the pursuit domain to compare ATISIS to the existing approaches. Our results show that ATISIS performs much faster than the existing methods and still achieves a very good performance. We observe that when teammates cannot be represented by the models that the ad hoc agent has, ATISIS achieves better performance and in a shorter time than the state-of-the-art algorithms.
- For ad hoc teamwork in complex domains, we propose AATEAM, an attention-based method to cope with teammates' different behaviours in complex domains. To the best of our knowledge, this is the first time attention mechanism is applied to this problem. Besides learning the mapping from contexts to probability distribution over actions, we train the attention network to measure the similarity between the corresponding past teammates and new

teammates. This helps to adjust the action selection in real-time. We perform extensive experiments in the RoboCup 2D simulation domain. The results demonstrate that the attention networks outperform PLASTIC-Policy when working with both known and unknown teammates. Additionally, we test multiple ways for training the recurrent neural networks. The results indicate that the attention mechanism is an essential part for AATEAM.

1.3 Organization of the Thesis

The rest of the thesis is organised as follows:

Chapter 2 describes the existing literature that copes with the uncertainties in agent cooperation. We first talk about the literature that deals with the untrustworthy information through trust management schemes. Second, we present the literature that tries to achieve ad hoc teamwork in simple domains by modeling and predicting teammates' behaviour. Last, we discuss the literature that aims to solve the ad hoc teamwork problem in complex domains by learning a policy for cooperating with each past teammate type and switching the policies to work with new teammates on-the-fly. We discuss their drawbacks and show how our approaches overcome their shortcomings.

Chapter 3 presents the preliminary knowledge of the methods we use in this thesis. Specifically, we introduce the MDP, POMDP, Bayesian inference, MCTS, and deep neural network.

Chapter 4 describes the details of the POMDP model that handles the decision process about whether an agent is confident enough to make a decision or it should gather more information. We explain the design of the model's states, actions, observations, transition function, observation function, and reward function. We also indicate how we can learn the observation function based on Bayesian inference to cope with the changing malicious behaviour. Then, we show the experimental results with the VEINS simulator [25] and demonstrate the effectiveness and efficiency of the proposed POMDP model.

Chapter 5 presents ATISIS that achieves ad hoc teamwork by sub-task inference and selection. We consider the common task as a set of sub-tasks and tries to pursue

the unattended sub-task. First, we describe how we infer teammates' choices of sub-task based on their actions. Second, we present the detailed design of the POMDP model that selects the ad hoc agent's sub-task based on the inference of teammates' choices. Further, we introduce the algorithm that adopts MCTS to search the best actions for the ad hoc agent to complete the selected sub-task. Last, we present the experiments' results under the pursuit domain and show the advantages of ATSYS over the existing works that rely on the teammates' behaviour models.

Chapter 6 presents AATEAM that takes advantage of the attention mechanism to achieve ad hoc teamwork in complex domains. First, we show the design and work flow of the recurrent neural network that contains the attention mechanism. The attention-based neural network takes the states as input and outputs the probability distributions over the ad hoc agent's action. We further discuss how to select the final action for the ad hoc agent given those probability distributions output by the attention-based neural networks. Next, we explain how to train the attention-based network in order to achieve the above design purpose. Finally, we present the experimental results under the RoboCup 2D simulation domain to demonstrate the advantages of AATEAM over PLASTIC-Policy when working with both known and unknown teammates. Moreover, we also show the training results to indicate the importance of the attention mechanism.

Chapter 7 summarises the algorithms developed in this thesis and discusses the possible directions for the future work.

Chapter 2

Literature Review

In this chapter, we review the approaches proposed in the literature that handle the challenging issues brought by the uncertainties in agent cooperation. Section 2.1 discusses the existing works that use the trust management schemes to deal with the untrustworthy information shared by selfish/malicious agents. Section 2.2 presents the state-of-the-art approaches that handle the ad hoc teamwork problem in simple domains. Section 2.3 discusses the most advanced work that tries to achieve ad hoc teamwork in complex domains. We point out the shortcomings of these approaches and explain how we cope with them in our proposed approaches.

2.1 Dealing with Untrustworthy Information in Multi-agent Systems

The selfish/malicious agents in multi-agent systems can share false information to mislead others. There are plenty of trust management schemes proposed to deal with the selfish/malicious agents. We can divide the existing trust management schemes into two approaches.

The first approach focuses on the computation of agents' trustworthiness. The trustworthiness of an agent represents the probability that it performs an action which is beneficial for the cooperation. In the first approach, the trustworthiness of an agent is usually evaluated based on direct experience (if available) as well as the opinions collected from other agents. Agents can interact in the system

and collect direct experience. When such interactions are rare, agents can ask its peers (that have the direct experience with agents in question) for the opinions or recommendations. Multi-agent systems researchers refer to this kind of information as *indirect evidence*. For example, [Chang and Kuo \[9\]](#) design a Markov Chain Trust Model that considers the trust level of an agent as a state. In this model, the agent's behaviour determines the state transition (i.e. the change of the agent's trust level). With the interaction experience, they compute the steady state transition probability that represents the agent's trustworthiness. [Wei et al. \[11\]](#) propose a trust model that considers both direct observations and peer recommendations. They adopt Bayesian inference and Beta distribution to compute direct beliefs about agents' trustworthiness from direct observations. Also, they use Dempster-Shafer Theory (DST) to aggregate peer recommendations. Then, they build the trustworthiness of each agent by computing the weighted sum of the direct belief and recommendations. [Li and Song \[12\]](#) design a trust management scheme that considers the trustworthiness of both shared data and agents. In their scheme, an agent first uses DST to compute the trustworthiness of shared data based on the data sensed by itself and the data shared from other agents. On top of this trustworthiness, the agent can further assess the trustworthiness of other agents. Please refer to [\[26, 27\]](#) for more trust management schemes designed for computing the trustworthiness of agents.

Note that the trustworthiness of agents is not equivalent to the trustworthiness of the information shared by them. For example, different information shared by the same agent may have different levels of trustworthiness [\[8\]](#). Therefore, the second approach is proposed to evaluate the trustworthiness of information in order to cope with the potentially untrustworthy information. In particular, the second approach assesses the trustworthiness of shared information based on the agents' trustworthiness (evaluated by the first approach) and the shared messages. [Raya et al. \[8\]](#) propose a data-centric trust scheme to compute the trustworthiness of each received message based on the sender's trustworthiness and relevant environmental features. The relevant environmental features provide additional information for evaluating the trustworthiness of a message. For example, when a vehicle reports an accident in VANETs, the location is a relevant environmental feature. The closer the sender is to the accident's location, the more likely the message reflects the system state. After the trustworthiness of messages is computed, the belief on the reported system state is obtained by aggregating the messages' trustworthiness

using voting, Bayesian inference, or DST. Relying on their scheme, more collected reports (shreds of evidence) will result in more certain belief about the system state. However, there is a question about how many reports are enough to ensure that the estimation of the system state is correct. As we mentioned before, if we estimate the unobservable system state based on only several reports, the estimation of the system state may be inaccurate due to false information. If we try to collect many reports to guarantee the accuracy of the state estimation, the delayed response may compromise the performance of agents. [Li et al. \[10\]](#) propose a reputation-based announcement scheme in which the exchanged message contains the system state information and the sender’s trustworthiness. If the trustworthiness is greater than a preset threshold, agents believe the system state is as reported. In this case, the receiver determines the system state only based on one message whose sender’s trustworthiness is higher than the threshold. This results in that a low threshold enables fast response while increasing the probability of being misled. Also, a high threshold decreases the chance of believing in false information but also increases the delay in responses. Therefore, the existing works only focus on how to assess the trustworthiness of shared information based on received messages. They neglect the problem of reasoning about how many received messages are enough before the agent can trust and take advantage of the shared information. Our approach overcomes this problem by proposing a POMDP model to balance the decision quality and response delay. In the POMDP model, each received message affects the belief about the system state. The state belief further influences the expected utility of using the shared information and collecting more information. Therefore, the POMDP model provides a principled way to make the decision about whether the agent should take actions based on the reported system state or collect more information.

We also notice that there exist approaches that integrate POMDP with trust management for decision-making [\[28–30\]](#). These approaches focus on the problem of selecting good partners. There, an agent queries the advisors about the quality of a potential partner. The quality of a partner can be regarded as a system state. They also consider that the advisors may provide unreliable opinions about the partners. Therefore, we could potentially adopt these approaches to handle untrustworthy information. However, [Irissappane et al. \[28–30\]](#) model the trustworthiness of each agent in the system as a POMDP state. This results in a large state space. Thus, these methods are not scalable for the multi-agent systems with numerous agents.

Although [Irissappane et al. \[30\]](#) try to solve the scalability issue by splitting a large state space into smaller identical subsets, this technique becomes more complex in multi-agent systems as the subsets might not be identical. This is because agents can join and leave the system frequently. This leads to that the number of agents in multi-agent systems is dynamic. Henceforth, the subsets can also be dynamic and thus cannot be identical. In our POMDP model, we only consider the trustworthiness of the sender who sends the message which is being processed and thus, we limit the size of the state space. Furthermore, in the above approaches, researchers set the POMDP parameters manually while we learn them in our model. We refer the reader to Chapter 4 for the details of our approach.

2.2 Ad Hoc Teamwork in Simple Domains

In the existing works relevant to ad hoc teamwork in simple domains, the ad hoc agent observes their teammates' actions to understand their underlying behaviour. We can divide these works into two approaches. The first approach does not consider the uncertainty brought by the dynamism of the environment. On one hand, [Agmon and Stone](#), [Chakraborty and Stone](#), and [Agmon et al. \[14–16\]](#) investigate ad hoc teamwork in matrix games and assume that the utilities of the team's actions are known. In detail, [Agmon and Stone \[14\]](#) assume the teammates always choose the best reaction to others' last observed actions. Based on this assumption, they propose a method to lead the team to the optimal team payoff. [Chakraborty and Stone \[15\]](#) extend [14]. They assume the teammates' policy is a function that takes as input some limited features derived from the history and outputs actions. They propose a scheme that tries to perform optimally when the derived features can explain the teammates' actions. Otherwise, it adopts a safe strategy. [Agmon et al. \[16\]](#) also extend [14]. They assume that the teammates' behaviour comes from a pre-defined set of the behaviour types. For each type of behaviour, they compute the utility of each action accordingly. Then, they compute the expected utility of an action as the weighted sum of the action's utility under every type of behaviour. The weight is the probability that teammates adopt the corresponding behaviour. Their solution chooses the action with the maximum expected utility. On the other hand, [Genter et al. \[17\]](#) consider the role assignment problem for different types of teamwork. For each type of teamwork, they formally define a method to measure

the influence of the ad hoc agent's role selection on the team utility. By doing so, their methods can efficiently select the role for the ad hoc agent that leads to the maximum team utility. However, they assume that the ad hoc agent knows which roles teammates are fulfilling. In real life, the utilities of the team's actions as well as the teammates' roles may be changing due to the changing environment. Thus, the assumption that they are known is too simplistic.

The second approach [18–23] considers the uncertainty of the environment. They represent their problem domain as MDP. MDP provides a framework for the ad hoc agent to select optimal actions based on its teammates' actions. There, the second approach assumes that the ad hoc agent can predict teammates' actions based on teammates' behaviour models. In detail, [Wu et al. \[18\]](#) propose an Online Planning Algorithm for Ad hoc Teamwork (OPAT). At every time step, they use MCTS and the teammates' behaviour model to estimate the long term utility of each joint action for the current state. Then, they construct a stage game whose utility function consists of the estimated utilities from MCTS. They further use biased adaptive play (BAP) to select the action for the ad hoc agent based on the history of joint actions in the current state. However, there may not be enough joint actions' history for the current state when the state space is large. Additionally, the evaluation of all joint actions increases the computational cost. Finally, when teammates' behaviour models do not match teammates' actual behaviour, the estimated utility of joint actions may be wrong and thus, lead to bad decisions. [Melo and Sardinha \[21\]](#) tackle the problem of identifying the goal of the team from multiple possible goals. This problem has been disregarded by the rest of the ad hoc teamwork literature. After recognising the goal of the team, they choose the ad hoc agent's action based on the prediction of teammates' actions. However, their approaches rely heavily on the assumption that teammates follow a bounded-rationality best-response model while the actual teammates may behave differently. [Chandrasekaran et al. \[22\]](#) propose to use reinforcement learning to generate possible policies of teammates (i.e. behaviour models) and use them as the level-0 models of interactive dynamic influence diagrams (I-DID) [31]. However, only when the teammates' behaviour can be represented by the policies, I-DID can make appropriate decisions to coordinate with them. Moreover, their scheme can only be applied to small-size problems while ATSS that we propose in Chapter 5 can handle much larger state space. This is because we consider the task as a set of sub-tasks and adopt MCTS to find the actions.

Albrecht and Ramamoorthy [19], Barrett et al. [20] and Ravula et al. [23] develop similar ideas to achieve ad hoc teamwork in simple domains. They maintain a probability distribution over teammates' behaviour models and update the distribution based on observed teammates' actions. Then, they consider teammates' possible actions. Based on those actions, they select the action for the ad hoc agent that maximises the expected utility. The behaviour models in [19] are provided by expert knowledge. In [20, 23], the ad hoc agent learns the behaviour models based on the interactions with previous teammates. Ravula et al. [23] further consider the problem of teammates' changing behaviour. They propose a method to handle the changing behaviour based on change point detection algorithm. However, in case that the expert knowledge is not representative or the past teammates are very different from current teammates, their performance can be arbitrarily bad. Learning the current teammates' behaviour model from scratch often takes time and thus, hinders the teamwork performance. Additionally, with respect to modeling teammates' behaviour, the most general model is the Interactive-POMDP (I-POMDP) [32]. This model maintains the belief about other agents' models and their beliefs about others. However, the belief about others' belief results in a nested belief that easily leads to intractability as the size of problem increases. Therefore, finding an accurate behaviour model is very likely infeasible. In contrast to the existing works that rely on teammates' behaviour models, ATSSIS does not depend on any behaviour models. We only infer teammates' desired sub-tasks by observing the effects of their actions.

2.3 Ad Hoc Teamwork in Complex Domains

The majority of the existing approaches for ad hoc teamwork [14–23] focus on simple domains. To the best of our knowledge, PLASTIC-Policy [24] is the only scheme proposed for more complex domains. In complex domains, the action space is partially observable due to the following reasons. First, when the action space is continuous, the ad hoc agent cannot know the exact value of the actions performed by its teammates. Second, two actions with different intentions may look similar in some situations. For instance, in the robosoccer domain, passing to the teammate that is close to the goal is indistinguishable from shooting. Henceforth, the ad hoc agent can only understand teammates' behaviour based on their state

changes. Additionally, since teammates' actions are partially observable, it is infeasible to collect training data for learning teammates' behaviour models (that output teammates' actions given a state).

Due to the challenging issues in complex domains, PLASTIC-Policy uses reinforcement learning to learn a policy for working with each type of past teammates. In case that the action space is continuous, PLASTIC-Policy first assumes the existence of a countable set of high-level actions. Each high-level action is a mapping from the state to a continuous action. By doing so, it limits the action space to be searched by reinforcement learning and thereby reduces the learning complexity. The policy takes a state as input and outputs a high-level action of the ad hoc agent. Since the ad hoc agent cannot directly observe teammates' actions, PLASTIC-Policy understands teammates' behaviour based on state transitions. Note that when the state space is continuous, PLASTIC-Policy needs to define the state transition for continuous state space. Given a state transition when working with new teammates, the ad hoc agent locates the most similar state transition with each type of past teammates. Using that information, it updates the probability of new teammates being similar to each type of past teammates, and applies the policy for the most similar past teammates. The main drawback of PLASTIC-Policy is that the learned policies are not necessarily optimal for new teammates. Additionally, the update based on state transitions may not be fast enough to adapt to new teammates' changing behaviour. AATEAM that we propose in Chapter 6 addresses these drawbacks by using the attention networks to select the best action based on both the context and the similarity between the past and new teammates' types.

2.4 Summary

This chapter reviews the existing approaches proposed to handle the uncertainties in agent cooperation. As for the information sharing in multi-agent systems, the existing approaches first use the trust management schemes to evaluate the trustworthiness of agents who share information. Then, they compute the trustworthiness of shared information based on agents' trustworthiness. However, the existing approaches mainly focus on the accuracy of trust computation while neglecting the incurred delay and cost. This thesis proposes a POMDP-based approach to

balance the trade-off between the accuracy of shared information and the response delay/cost. In addition, when compared to the POMDP-based methods designed for the other domains, our approach is more scalable.

With respect to ad hoc teamwork in simple domains, the existing works either ignore the dynamism of the environment or rely heavily on the teammates' behaviour models. Since the new teammates encountered in ad hoc teamwork can have many types of behaviours, the existing works can perform arbitrarily bad when new teammates behave very differently from the acquired behaviour models. This thesis proposes ATISIS that achieves ad hoc teamwork without relying on teammates' behaviour models. In ATISIS, the ad hoc agent first infers the sub-tasks teammates are pursuing and uses a POMDP model to select its own sub-task. By using the POMDP model, the ad hoc agent handles the uncertainties in teammates' behaviour. When it comes to ad hoc teamwork in complex domains, the most advanced approach learns a set of policies for working with the past teammate types, and determines which policy to use on-the-fly when working with new teammates. Note that the switching between policies takes time, and thus, may not adapt fast enough to new teammates' changing behaviour. This thesis proposes AATEAM that uses the attention mechanism to learn how to deal with the behaviour of every past teammate type. AATEAM also measures the similarity between the past and new teammates' behaviours in every step. Therefore, AATEAM can adapt to new teammates' changing behaviour in real-time.

Chapter 3

Background

In this chapter, we present the preliminary knowledge relevant to our approaches. First, we introduce the concept of MDP and POMDP. Second, we describe the Bayesian inference technique and explain how to adopt it to learn the observation function of POMDP. Third, we describe the procedures of MCTS. Finally, we introduce the deep neural network and the attention mechanism used in it.

3.1 MDP and POMDP

3.1.1 Markov Decision Process

The MDP [33] provides a mathematical framework for modeling the interaction between the agent and the world. As shown in Figure 3.1, the agent senses the state of the world and takes actions accordingly to affect the world. For the domains modeled as an MDP, we normally assume that the agent has the perfect perceptual ability and thus perfect knowledge of the state. After an action is performed, the state of the world will be transited to a new state. In many domains, the transition of the state can be probabilistic and how the state transits is defined by a state transition function. Also, during the state transition, the agent receives a certain amount of domain-specific rewards that are defined by a reward function. An MDP is formally defined by a tuple $\langle S, A, P, R \rangle$, where

- S is a set of states;

FIGURE 3.1: Model the interaction between the world and agents by MDP

- A is a set of actions which cause the transition of the current state s to the next state s' ;
- $P : S \times A \rightarrow \Pi(S)$ is the state transition function which specifies the probability distribution over the next state $P(s'|s, a) \forall s' \in S$, given the action a and the current state s ;
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function which returns the reward $R(s, a, s')$ obtained by taking the action a in the state s and resulting in the state s' . The reward function can also be defined as $R(s, a)$ where

$$R(s, a) = \sum_{s'} P(s'|s, a) \cdot R(s, a, s') \quad (3.1)$$

In an MDP model, the reward and the next state are determined only by the current state and the action being taken. This is known as the *Markov* property, i.e. the reward and the conditional probability distribution of the future state depend only on the present state and action, not on the sequence of state-action pairs which precede it.

During the interaction between the agent and the world, the agent will receive a reward in every time step. Let r_t denote the reward received in time step t . The cumulative reward obtained during k time steps can be calculated as

$$\sum_{t=0}^k \gamma^t \cdot r_t \quad (3.2)$$

where $0 < \gamma < 1$ is the discounting factor which quantifies the importance of the short term and long term rewards. An MDP model is called finite-horizon

MDP if k is finite and called infinite-horizon MDP if k is ∞ . If in a domain the interaction between the agent and the world never ends or we are unsure about when the interaction will terminate, we model this domain as an infinite MDP and the discounting factor γ ensures that the cumulative reward is finite.

A policy $\pi : S \rightarrow A$ describes the behaviour of an agent, i.e. which action the agent will perform given the current state. With a specified policy π , the value function V_π that computes the utility of a state s (i.e. the cumulative reward to be obtained by following the policy π from the state s) can be recursively defined as

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V_\pi(s') \quad (3.3)$$

where $R(s, \pi(s))$ is the immediate reward of the current time step and the other part $\sum_{s' \in S} P(s'|s, \pi(s)) V_\pi(s')$ is the expected future reward. Similarly, the Q-value $Q^\pi(s, a)$ that represents the reward to be obtained by performing the action a in the state s and following the policy π in the remaining time steps is defined as

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_\pi(s') \quad (3.4)$$

The optimal policy π^* is the policy whose value function is larger than or equal to the value function of all other policies for all states, i.e.

$$\pi^*(s) = \operatorname{argmax}_a \left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right] \quad (3.5)$$

where V^* is the value function of the optimal policy. According to [34], there exists an optimal policy for every state. Therefore, given an MDP model, the corresponding optimal policy enables the agent to make optimal decisions.

Note that in this thesis, we mainly focus on the MDP models with a finite state space. Given a finite state space, we can apply value iteration [33] or policy iteration [35] to find the optimal policy of an MDP model. For the MDP models with an infinite (or continuous) state space, we can either use discretization to transform the state space into a finite one or apply value function approximation (e.g. using a deep neural network as an approximator) to compute the utility of a given state.

FIGURE 3.2: Model the interaction between the world and agents by POMDP

3.1.2 Partially Observable Markov Decision Process

For the domains where the agent can only observe a part of the world state, the MDP framework is insufficient since there is no full knowledge of the state. To plan the actions effectively, the agent needs to rely on the history of previous actions and observations since they can provide the information of the partially observable world state. Therefore, an MDP model is naturally extended to a POMDP model by integrating the observations. As shown in Figure 3.2, in every time step, the agent will get an observation of the world after taking an action. From the observation, the agent can estimate the partially observable state and update its belief on the state. The updated belief represents the summarization of all previous experience. Based on the belief, the agent can choose its action for the next time step.

A POMDP is formally formulated as a tuple $\langle \mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \mathbf{\Omega}, \mathbf{O} \rangle$ ¹ where

- $\mathbf{S}$, $\mathbf{A}$, $\mathbf{P}$, and $\mathbf{R}$ describe an MDP;
- $\mathbf{\Omega}$ is a set of observations obtained after taking an action.
- $\mathbf{O} : \mathbf{S} \times \mathbf{A} \rightarrow \Pi(\mathbf{\Omega})$ is the observation function specifying the probability distribution over observations $\mathbf{O}(o|\mathbf{s}', \alpha) \forall o \in \mathbf{\Omega}$, given the action α and the next state $\mathbf{s}'$. Since the agent does not know the exact state of the world in POMDP, this function quantifies the information of the partially observable world state provided by the observation.

¹We use bold letters for some of the POMDP notations in order to distinguish them from their equivalents in the MDP.

A POMDP maintains a belief $b(\mathbf{s})$ that is the probability distribution over the state $\mathbf{s}$. Every time an action α is taken and an observation o is received, the updated belief $b'(\mathbf{s}')$ is calculated as

$$b'(\mathbf{s}') \propto \mathbf{O}(o|\mathbf{s}', \alpha) \sum_{\mathbf{s} \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s}, \alpha) b(\mathbf{s}) \quad (3.6)$$

Since the state in POMDP is partially observable, the policy of POMDP maps the belief of state to an action. This is different from the case of MDP where an MDP policy maps the state to an action. Similar as MDP, the optimal policy of POMDP is desired for making optimal decisions under partially observable domains. The value function of the optimal policy is defined as

$$V^*(b) = \max_{\alpha \in \mathbf{A}} Q^*(b, \alpha) \quad (3.7)$$

$$Q^*(b, \alpha) = \sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) \mathbf{R}(\mathbf{s}, \alpha) + \gamma \sum_{o \in \Omega} \mathbf{P}(o|b, \alpha) V^*(b^{\alpha o}) \quad (3.8)$$

where $\mathbf{P}(o|b, \alpha) = \sum_{\mathbf{s}' \in \mathcal{S}} \mathbf{O}(o|\mathbf{s}', \alpha) \sum_{\mathbf{s} \in \mathcal{S}} \mathbf{P}(\mathbf{s}'|\mathbf{s}, \alpha) b(\mathbf{s})$, γ is the discounting factor and $b^{\alpha o}$ is the new belief updated based on the current belief b , performed action α and received observation o . Note that given α , o and b , $b^{\alpha o}$ can be fully determined by Eqn. 3.6. Since different actions result in different observations that provide different amounts of information about the state, POMDP provides a principled framework for balancing the exploitation (taking actions to get large rewards in the partially observable state) and exploration (taking actions to gather more information). The optimal policy can thus return the maximum expected reward under uncertainty.

3.1.3 Multi-agent Goal-oriented Markov Decision Process

When there are multiple agents collaborating with each other to pursue a goal, the MDP can be extended as a multi-agent goal-oriented MDP which is the combination of multi-agent MDP [36] and goal-oriented MDP [37]. The multi-agent goal-oriented MDP is given by a tuple $\langle N, S, \{A_i\}, P, G, R, \gamma \rangle$, where N is a set of agents; S is a set of states; A_i is the set of actions of an agent $i \in \{1, 2, \dots, |N|\}$ and $A = \times_{i=1}^{|N|} A_i$ is the set of joint actions; $P: S \times A \times S \rightarrow [0, 1]$ is the state transition function which specifies the probability of transiting to the next state

$P(s'|s, \vec{a}) \forall s' \in S$, given the joint action $\vec{a} \in A$ and the current state $s \in S$; G is the set of goal states which represents the shared goal of teamwork; $R : S \times A \times S \rightarrow \mathcal{R}$ is the reward function that specifies the reward $R(s, \vec{a}, s')$ of the team when taking the joint action $\vec{a}$ in the state s and resulting in the state s' ; $R(s, \vec{a}, s')$ is positive when $s' \in G$ and negative otherwise; and γ is the discount factor for the future reward where $0 < \gamma \leq 1$.

A joint policy $\pi : S \times A \rightarrow [0, 1]$ specifies the probability $\pi(\vec{a}|s)$ of taking the joint action $\vec{a}$ in the state s . Given π , the utility of the state s can be computed as:

$$V^\pi(s) = \sum_{\vec{a} \in A} \pi(\vec{a}|s) \sum_{s' \in S} P(s'|s, \vec{a}) [R(s, \vec{a}, s') + \gamma V^\pi(s')] \quad (3.9)$$

Similarly, the Q-value of the joint action $\vec{a}$ in the state s is computed as:

$$Q(s, \vec{a}) = \sum_{s' \in S} P(s'|s, \vec{a}) [R(s, \vec{a}, s') + \gamma V^\pi(s')] \quad (3.10)$$

3.2 Bayesian Inference for Model Parameters

Bayesian inference is an important method to cope with the partial knowledge of a random variable. The prior probability of a random variable can be estimated based on the prior knowledge. Then Bayesian inference uses Bayes' theorem to compute the posterior probability of the random variable when obtaining more data or information. Let $\mathcal{X}$ denote the random variable which is hidden to us and $\mathcal{Y}$ denote the random variable which is related to $\mathcal{X}$ and observable to us. The posterior distribution of $\mathcal{X}$ can be computed based on the observed $\mathcal{Y}$ as below

$$P(\mathcal{X}|\mathcal{Y} = y) = \frac{P(y|\mathcal{X})P(\mathcal{X})}{\int P(y|\mathcal{X})P(\mathcal{X})d\mathcal{X}} \quad (3.11)$$

where $P(\mathcal{X})$ is the prior distribution of $\mathcal{X}$. Further, let $\theta \in \Theta$ denote the unknown parameters which specify the probability distribution of $\mathcal{X}$. Inferring the probability distribution of $\mathcal{X}$ is equivalent to inferring the probability distribution of θ , i.e.

$$P(\theta|\mathcal{Y} = y) = \frac{P(y|\theta)P(\theta)}{\int_{\Theta} P(y|\theta)P(\theta)d\theta} \quad (3.12)$$

where $P(\theta)$ is the prior distribution of θ . Since the computation of the normalizing factor $\int_{\Theta} P(y|\theta)P(\theta)d\theta$ is often difficult, it is common to assume that $P(\theta)$ is a

conjugate prior and is parameterized by ϕ . Given this assumption, the posterior distribution $P(\theta|\mathcal{Y} = y)$ and prior distribution $P(\theta)$ are in the same probability distribution family. In this case, there is a closed-form expression for the posterior and we can update the posterior by simply updating the parameters ϕ .

For the POMDP model which has unknown model parameters, we can use Bayesian inference to learn those parameters [38]. For instance, given $\mathbf{s}'$ and α , the possible observations $\{o_i|i = 1, \dots, |\Omega|\}$ can be considered as being generated by a multinomial distribution whose parameters are the unknown observation functions $\{\mathbf{O}(o_i|\mathbf{s}', \alpha)|i = 1, \dots, |\Omega|\}$. Then, we can assume the prior over the unknown observation functions as a Dirichlet distribution $Dir^{\mathbf{s}'\alpha}(\phi_{o_i}^{\mathbf{s}'\alpha}|i = 1, \dots, |\Omega|)$, where $\phi_{o_i}^{\mathbf{s}'\alpha}$ is the Dirichlet parameter. The expected value of $\mathbf{O}(o_i|\mathbf{s}', \alpha)$ can be computed as [39]:

$$\mathbf{O}(o_i|\mathbf{s}', \alpha) = \frac{\phi_{o_i}^{\mathbf{s}'\alpha}}{\sum_{j=1}^{|\Omega|} \phi_{o_j}^{\mathbf{s}'\alpha}} \quad (3.13)$$

If $\vec{h}^{\mathbf{s}'\alpha}$ denotes the observation history, i.e. a set of observations when the action α causes a transition to the state $\mathbf{s}'$, then the posterior of $\{\mathbf{O}(o_i|\mathbf{s}', \alpha)|i = 1, \dots, |\Omega|\}$ is expressed as a Dirichlet distribution $Dir^{\mathbf{s}'\alpha}(\phi_{o_i}^{\mathbf{s}'\alpha} + N(o_i|\vec{h}^{\mathbf{s}'\alpha})|i = 1, \dots, |\Omega|)$, where $N(o_i|\vec{h}^{\mathbf{s}'\alpha})$ is the number of times the observation o_i appears in $\vec{h}^{\mathbf{s}'\alpha}$. The expected probability $\mathbf{O}(o_i|\mathbf{s}', \alpha)$ is then given by,

$$\mathbf{O}(o_i|\mathbf{s}', \alpha) = \frac{\phi_{o_i}^{\mathbf{s}'\alpha} + N(o_i|\vec{h}^{\mathbf{s}'\alpha})}{\sum_{j=1}^{|\Omega|} [\phi_{o_j}^{\mathbf{s}'\alpha} + N(o_j|\vec{h}^{\mathbf{s}'\alpha})]} \quad (3.14)$$

In general, we do not exactly know the state $\mathbf{s}'$ after receiving an observation in POMDPs. However, in the domains where it is possible to obtain the ground truth of $\mathbf{s}'$, we can use Bayesian inference to learn the unknown model parameters.

3.3 Monte Carlo Tree Search

MCTS [1] is a heuristic algorithm which searches the actions that maximise the long term expected reward given the current state. As shown in Figure 3.3, there are four phases during one search process: selection, expansion, simulation and backpropagation. In the selection phase, the current state is treated as the root node of a search tree. From the root node, one child node (the next state resulting

FIGURE 3.3: The demonstration of MCTS [1]

from taking the action represented by the edge) is selected based on a tree policy. This procedure continues until an expandable node is selected. The expandable node is the node whose corresponding state has unvisited child states. Then one of the unvisited child states is added to expand the search tree in the expansion phase. From the expanded node, simulation is run based on a simulation policy. When the search depth has reached a preset value or the termination condition is met, the simulation ends and the reward obtained during the simulation is propagated to update the long term expected utility of expanded node and its parents. MCTS can keep repeating this search process as long as there is still planning time. When no more planning time remains, MCTS chooses the action which results in the child state that has the largest long term expected utility.

The advantages of MCTS are that it can handle the domains with a large state/action space and the quality of recommended actions will improve smoothly as more computation efforts are dedicated. More importantly, the search process does not require any domain knowledge. MCTS only needs a model to sample the next state (i.e. select one state from the state space) given the current state and an action. While MCTS has these advantages, a lot of variants of MCTS have been proposed to further enhance its advantages. One of the most popular variants is called UCT (Upper Confidence Bound 1 applied to trees) algorithm [40] as shown in Algorithm 1. The key enhancement of UCT is about the node selection in the tree policy, i.e. $\operatorname{argmax}_a \hat{Q}(s, a) + c \sqrt{\frac{\ln N(s)}{N(s, a)}}$ in the line 22 of Algorithm 1. This design achieves a better balance between the exploitation of the actions with high expected utility ($\hat{Q}(s, a)$) and the exploration of the actions which are less visited ($\sqrt{\frac{\ln N(s)}{N(s, a)}}$), where $N(s)$ is the number of times the state s has been visited, $N(s, a)$ is the number of

times the state-action pair (s, a) has been visited and c is the balancing parameter.

Algorithm 1: The Pseudocode of UCT

Input: The MDP model $\langle S, A, T, R \rangle$, current joint state $s_0 \in S$.

Output: The MDP action a^* .

```

1 Search Tree  $\leftarrow s_0$ 
2 Repeat
3   Search( $s_0$ )
4 Until number of simulations reaches the preset number
5 return  $a^* = \operatorname{argmax}_a \hat{Q}(s, a)$ 

6 Function Search( $s$ )
7    $s' \leftarrow \text{TreePolicy}(s)$ 
8    $r \leftarrow \text{DefaultPolicy}(s')$ 
9   BackPropagation( $s', r$ )

10 Function TreePolicy( $s$ )
11   while  $s$  is non-terminal do
12     if  $s$  is expandable then
13       return Expand( $s$ )
14     else
15        $s' \leftarrow \text{BestChild}(s, c)$ 
16   return  $s'$ 

17 Function Expand( $s$ )
18   choose  $a$  from the set of actions which are untried under  $s$ 
19   Sample  $s'$  based on  $s$  and  $a$ 
20   return  $s'$ 

21 Function BestChild( $s, c$ )
22   return  $\operatorname{argmax}_a \hat{Q}(s, a) + c \sqrt{\frac{\ln N(s)}{N(s, a)}}$  (where  $N(s) = \sum_{a \in A} N(s, a)$ )

23 Function DefaultPolicy( $s$ )
24   while  $s$  is non-terminal do
25     choose  $a$  uniformly at random
26     Sample new  $s$  based on current  $s$  and  $a$ 
27   return reward  $r$  obtained during the simulation

28 Function BackPropagation( $s, r$ )
29   while  $s$  is not null do
30      $N(s, a) \leftarrow N(s, a) + 1$ 
31      $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \frac{r - \hat{Q}(s, a)}{N(s, a)}$ 
32      $s \leftarrow \text{parent of } s$ 

```

3.4 Deep Neural Network

As shown in Figure 3.4, a deep neural network (DNN) is consist of multiple layers where the first layer is the input layer and the last layer is the output layer. Each layer has a set of neurons that output scalar values, and the output of a layer is a vector. The DNN is a popular method to model complex non-linear relationships. Each layer in a DNN combines the features from its previous layer, and thus a DNN is a compositional model.

One of the most widely used components in DNNs is one linear layer plus one activation function. Formally, the layer k outputs a vector $\vec{h}^k$ based on the output $\vec{h}^{k-1}$ of its previous layer,

$$\vec{h}^k = \mathcal{F}(\vec{b}^k + W^k \vec{h}^{k-1}) \quad (3.15)$$

where $\vec{b}^k$ is a vector of offsets, W^k is a weight matrix, and $\mathcal{F}$ is an activation function that determines whether each neuron of the layer should be activated or not. Commonly the activation function is a non-linear function, e.g. Tanh[41], Relu[42], etc. This is because the non-linear transformation of data enables the network to learn complex non-linear relationships. In addition, the linear layer is often used to transform an input vector to a vector of higher dimension so that it can extract useful features from the input or obtain the embedding of the input. Let l denote the dimension of the input vector $\vec{h}^{k-1}$ and l' denote the dimension of

FIGURE 3.4: The general architecture of deep neural networks [2]

the output vector $\vec{h}^k$. Then W^k is an $l' \times l$ matrix and $\vec{b}^k$ is a vector of dimension l' .

Since the objective of a DNN is to model the complex relationships between inputs and outputs, we need to compare the output of the network $\vec{h}^o$ with the desired output $\hat{y}$ for the given input during training. We call the desired output as label. An input and its corresponding label form a training sample. A loss function $\mathcal{L}$ takes $\vec{h}^o$ and $\hat{y}$ as input to compute the loss $\mathcal{L}(\vec{h}^o, \hat{y})$. Given a loss value, we use the backpropagation (gradient descent) method to adjust the weights of each network layer so that the loss is minimized. Different problems require different types of loss function. For the problems where the desired output is one of the elements from a finite set, the loss function is the negative conditional log-likelihood. Formally, let $\mathcal{O}$ denote the output space. The network output is a probability distribution over $\mathcal{O}$, i.e. $\{P(y_i) | i = 1, \dots, |\mathcal{O}|\}$, and the loss is computed as $-\log P(y_i = \hat{y} | \vec{x})$.

3.4.1 Recurrent Neural Network

The recurrent neural network (RNN) extends the conventional feedforward neural network such that it can deal with a variable-length sequence input and capture the long-term dependencies inside the sequence. The RNN achieves this by maintaining a recurrent hidden state whose value at each time step depends on its value of previous time steps. Formally, let $\vec{x}_t$ denote the input of time step t . The hidden state $\vec{h}_t$ is,

$$\vec{h}_t = \begin{cases} \vec{0} \text{ or a specified vector,} & t = 0 \\ \phi(\vec{h}_{t-1}, \vec{x}_t), & \text{otherwise} \end{cases} \quad (3.16)$$

where ϕ is a non-linear function. In addition, the RNN can have an optional output $\vec{y}_t$ for every time step t . A popular version of RNN is called gated recurrent unit (GRU) [59] as illustrated in Figure 3.5.

For clarity, during the introduction of GRU below, we use a bold symbol instead of top arrow to represent a vector. In the GRU, the $\mathbf{h}_t$ is the hidden state while the $\tilde{\mathbf{h}}_t$ is the candidate hidden state that introduces new information to the hidden state $\mathbf{h}_t$. The $\mathbf{z}_t$ is the update gate that adjusts the percentage of the new information being updated. The $\mathbf{r}_t$ is the reset gate that determines how much information from the previous hidden state $\mathbf{h}_{t-1}$ will be used to compute the candidate hidden

FIGURE 3.5: The illustration of GRU [58]

state $\tilde{\mathbf{h}}_t$. Specifically, the j th element of the $\mathbf{h}_t$ is computed as,

$$\mathbf{h}_t^j = (1 - \mathbf{z}_t^j) \mathbf{h}_{t-1}^j + \mathbf{z}_t^j \tilde{\mathbf{h}}_t^j \quad (3.17)$$

where the $\mathbf{z}_t^j$ is calculated as,

$$\mathbf{z}_t^j = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})^j \quad (3.18)$$

where σ is the logistic sigmoid function, W_z and U_z are weight matrices. The candidate hidden state $\tilde{\mathbf{h}}_t^j$ is computed as,

$$\tilde{\mathbf{h}}_t^j = \tanh(W \mathbf{x}_t + \mathbf{r}_t \odot (U \mathbf{h}_{t-1}))^j \quad (3.19)$$

where $\odot$ is an element-wise multiplication, W and U are weight matrices, and the $\mathbf{r}_t$ is calculated as,

$$\mathbf{r}_t^j = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1})^j \quad (3.20)$$

where W_r and U_r are weight matrices.

3.4.2 Attention Mechanism

The attention mechanism is an approach for intelligently extracting contextual information from a sequence of inputs. Due to its great performance, the attention mechanism has become a popular technique in computer vision [43, 44] and natural language processing (NLP) [45, 46].

FIGURE 3.6: The attention network for translation

The example attention network for translation is shown in Figure 3.6. The attention network has two recurrent neural networks called encoder and decoder respectively. The attention network first encodes a full sentence with its encoder. The last hidden state from the encoder is the initial hidden state (input) to its decoder. To do the translation, the attention mechanism decides the importance (i.e. weights) of each input to the translation output. The weighted sum of encoder's outputs is the contextual information. The decoder's hidden state and the contextual information determine the output together. The method to compute the weight in the attention mechanism varies from problem to problem.

Note that in the teamwork, the ad hoc agent's action in the current step is correlated to the current state as well as to previous states. Given different teammates' behaviours (coming from different teammate types or different situations), the correlations of previous states to the current action are different. Therefore, the attention mechanism is a suitable approach that takes the sequence of states as input and outputs the correlations given different teammates' behaviours.

3.5 Summary

This chapter presented the preliminary knowledge of the methods being used in our approaches. In the Chapter 4, we propose a POMDP-based approach to balance the trade-off between making decisions based on shared information and collecting

more information. In addition, we use Bayesian inference to learn the observation function of the proposed POMDP model. In the Chapter 5, we use an MDP to model ad hoc teamwork in simple domains. Then we use a POMDP to handle the uncertainties in the inference of teammates' sub-tasks when selecting the sub-task for the ad hoc agent. Finally, we use the MCTS to choose the best actions for the ad hoc agent to pursue the selected sub-task. In the Chapter 6, we use the attention mechanism to extract the contexts of the teamwork. This enables us to train the deep neural networks that perform well in complex domains.

Chapter 4

POMDP-based Decision Making for Using Information in Multi-agent Systems¹

In multi-agent systems, the information shared by other agents is potentially untrustworthy due to their selfish/malicious behaviours. In general, collecting more information can lead to higher accuracy of the information. However, collecting information also takes time. Note that agents often need to take actions according to the shared information promptly. Therefore, it is impractical for agents to collect information indefinitely. To achieve good performance, agents should balance the decision quality and the decision cost/delay given the potentially untrustworthy information. Henceforth, this chapter aims to address the problem below:

Problem: how agents decide whether they are confident enough to take actions based on the shared information or whether they should collect more information.

Formally, let b_t denote the belief about the shared information after receiving t messages, $g(b_t)$ denote the expected reward gained by taking actions based on the belief b_t , and $h(t)$ denote the cost of collecting t messages. The problem is to maximise the below utility,

$$g(b_t) - h(t) \tag{4.1}$$

In general, the more certain b_t , the higher $g(b_t)$. The larger t , the more certain b_t while the higher $h(t)$.

¹This chapter contains the work published in [47].

Since the POMDP provides a principled framework for sequential decision-making under uncertainty, it is a suitable decision-making tool. By using a POMDP model, agents can effectively balance the trade-off between the exploitation (taking actions based on shared information) and exploration (gathering more information). Therefore, we model the problem of using the shared information in multi-agent systems as a POMDP. This tackles the drawback of the existing trust management schemes, i.e. focusing on the accuracy of trust computation while ignoring the timeliness of decision making. Furthermore, the state space of our POMDP model is small, and thus we can adopt the model in large-scale multi-agent systems. The structure of this chapter is as follows. First, we give an overview of the system architecture. Second, we describe the model design in detail. Finally, we present our experiments conducted in the domain of vehicular ad-hoc networks (VANETs).

4.1 System Architecture

In this section, we introduce the architecture of our system. In detail, there are two kinds of entities in the system: 1) a centralised server; and 2) a group of agents. These entities exchange information with one another through communication.

Server: the server is in charge of calculating the trustworthiness of agents. The derived agents' trustworthiness is used in our POMDP model. In many real-life applications, the centralised server is essential since the distributed trust management schemes often do not work.

In particular, in the distributed trust management schemes, an individual agent infers the trustworthiness of encountered agents through direct interaction experience and others' recommendations. However, agents in multi-agent systems may join and leave the system freely (especially for large open systems). Hence, agents may meet strangers during their work. Thus, they do not have direct interaction experience with those agents. In addition, agents may collude with one another when providing recommendations to promote or decrease the trustworthiness of a certain agent (which is known as the collusion attack). Therefore, the recommendations from other agents may not be reliable. Additionally, computing peer trustworthiness by agents themselves results in additional response delays. In comparison, a centralised server can use centralised trust management schemes that

enable a more accurate trustworthiness evaluation than the distributed ones. Additionally, a centralised server has more computational power than each single agent, and thus, compute the trustworthiness more efficiently. Therefore, we have such a centralised server in our system.

In the system, agents can upload the messages received from others to the server. By analysing those messages together (e.g. max voting), the server can determine the ground truth of shared information and thus, the trustworthiness of the agents. Note that agents may not always be able to connect to the server due to the limited communication resources. Therefore, there may be delays in exchanging messages between agents and the server. Additionally, the server needs time to collect enough messages to determine the ground truth. Henceforth, agents cannot directly rely on the ground truth received from the server when taking actions.

Besides the computation of trustworthiness, the server also issues a certificate for each agent. The certificate contains an agent's public key that is used to verify messages' signatures generated by the agent's private key. This enables to verify the identity and to check information integrity [10, 48–50]. The adoption of certificates is the common practice to prevent the attackers from spreading information with false identities or manipulating the transmitted information. Note that certificates cannot stop the agents with valid identities from sharing false information. However, once an agent sent a message with its valid identity, it cannot deny since the message contains the signature verifiable by its public key. In our work, an agent's certificate also contains the trustworthiness² of the agent and a time stamp that indicates when the trustworthiness is computed. The server periodically generates a new certificate for each agent that expires after a given time period. Agents download new certificates when they connect to the server. When exchanging messages, agents attach the certificates to prove their identity and trustworthiness. Note that malicious agents can choose to use old certificates instead of the latest ones.

Agents: agents exchange the information of the observed system state by broadcasting messages. The communication technique can be Wi-Fi, Bluetooth, Dedicated Short Range Communications (DSRC), etc. The agents located within the communication range of an agent are called its neighbours. When an agent needs to know the system state that it cannot directly observe, it can broadcast a query

²Though trustworthiness is a continuous value, we discretise it to fit our POMDP model and use standard POMDP solvers.

message to ask for the information of that state from neighbours. Upon receiving the query message, other agents can broadcast response messages to share what they have observed. In this work, we focus on discrete system states. For a continuous state, we can discretise it as a finite set of value ranges. Note that malicious agents may send untruthful information of a system state even without observing it. Once an agent receives a response message, it adds the message to a unique message queue if the state reported by that message is relevant to its actions. Otherwise, it discards the message.

4.2 Proposed POMDP Model

In this section, we present the details of our POMDP model and the algorithm for learning the observation function of the model.

4.2.1 Model Design

Our POMDP model aids agents in fast and efficient decision-making based on the shared information about unobserved system states. It makes quality decisions promptly using fewer messages (queries, responses). The model maintains a belief about a system state, i.e. the probability distribution over the values of the system state. It updates this belief by processing the messages about the system state. The model also takes into consideration that the neighbouring agents can exhibit different behaviours (e.g. truthful, malicious etc.) in providing the state information. Based on the state belief, the model decides whether the agent should believe the system state is as reported and take actions accordingly, or it should wait for more messages. We consider only a single system state per POMDP execution. For handling multiple system states, the agent can simply initialise a POMDP for each of them. Next, we describe the POMDP components.

4.2.1.1 State

The POMDP state $\mathbf{s} = \langle r, l, f \rangle$ consists of three variables:

- $r \in \{v_1, v_2, \dots, v_n\}$ represents the value of a system state where n is the number of possible state values.
- $l \in \mathcal{L} : \{high, medium, low\}$ is the trustworthiness of the neighbouring agent (called sender) who sent the message being processed currently (called current message).
- $f \in \mathcal{F} : \{fresh, old\}$ is the freshness of the sender's trustworthiness. This variable reflects whether the sender's trustworthiness is computed recently or not. Its value (*fresh* or *old*) is determined based on the time stamp contained in the certificate. We include the freshness as a part of the state because malicious agents might use old certificates. This is because their malicious behaviour often results in low trustworthiness in their new certificates.

We only model the trustworthiness of the sender who sent the current message and perform belief updates based on each message. This prevents our POMDP model from having a large state space that often results in the scalability issues for POMDP.

4.2.1.2 Action

Our model knows the following actions:

- *query* (Q) for collecting the state information from neighbours. Query action represents either checking the unique message queue for the system state or brocasting query messages. If messages are present in the queue, they are retrieved in a First-in-First-out order and serve as observations for our POMDP model. If the queue is empty, a query message is broadcast to neighbours.
- a set of responsive actions $\{Action_i (ACT_i) | i = 1, \dots, n\}$ where ACT_i means believing the value of the system state as v_i and taking the corresponding actions for pursuing the agent's goal.

4.2.1.3 State Transition Function

The function is formulated as $\mathbf{P}(\mathbf{s}' = \langle r', l', f' \rangle | \mathbf{s} = \langle r, l, f \rangle, \alpha)$. For *query* action, $r = r'$ while l' and f' are unknown before receiving any observations (by retrieving a message from the message queue). Since the message queue could be empty, an observation may not be available immediately. We consider l' and f' to be uniformly distributed in their space after *query* action is performed. Thus, the transition function for *query* action is given by:

$$\mathbf{P}(\mathbf{s}' | \mathbf{s}, Q) = \frac{\delta_{r,r'}}{|\mathcal{L}||\mathcal{F}|} \quad (4.2)$$

where $\delta_{r,r'}$ is the Kronecker delta equals to 1 if $r' = r$, and 0 otherwise. The denominator $|\mathcal{L}||\mathcal{F}|$ represents the uniform distribution over the trustworthiness and freshness levels.

When $Action_i$ action is performed, the decision-making process for using the shared information about the system state is completed and we can simply set their transition function as a uniform distribution, $\mathbf{P}(\mathbf{s}' | \mathbf{s}, ACT_i) = \frac{1}{n|\mathcal{L}||\mathcal{F}|}$.

4.2.1.4 Reward Function

Since *query* action results in the resource consumption and delayed response, it incurs cost $\mathbf{R}(\mathbf{s}, Q)$. With respect to the responsive actions $\{ACT_i\}$, the amount of rewards that an agent receives only depends on whether the performed action matches the actual state value. Therefore, the reward function for responsive actions is only relevant to r . On making a correct decision (i.e., performing ACT_i when the value of the system state is v_i), the agent receives a reward $\mathbf{R}(r = v_i, ACT_i)$. For a wrong decision (i.e., performing ACT_i while the value of the system state is not v_i), the agent receives a penalty $\mathbf{R}(r \neq v_i, ACT_i)$. We assume that rewards have a positive value and penalties have a negative value.

4.2.1.5 Observation

The observations in our model are the received response messages. Each message contains the sender's report about the system state, the trustworthiness of the sender and the time stamp of updating trustworthiness that indicates the freshness

level of trustworthiness. We denote the observation as a tuple $o = \langle o_r, o_l, o_f \rangle$ where $o_r \in \{v_1, v_2, \dots, v_n\}$ is the sender's report about the system state, $o_l \in \mathcal{L}$ is the sender's trustworthiness and $o_f \in \mathcal{F}$ is the freshness level of trustworthiness.

4.2.1.6 Observation Function

In our model, we assume the full observability on the trustworthiness of sender and its freshness as they are values contained in the certificate. Only the value of the system state (i.e., r) is partially observable. We formulate our observation function to reflect this. For the *query* action Q ,

$$\mathbf{O}(\langle o_r, o_l, o_f \rangle | \langle r', l', f' \rangle, Q) = \delta_{o_l, l'} \cdot \delta_{o_f, f'} \cdot Pr(o_r | \langle r', o_l, o_f \rangle) \quad (4.3)$$

where the Kronecker delta $\delta_{o_l, l'} = 1$ if $o_l = l'$, and $\delta_{o_f, f'} = 1$ if $o_f = f'$. Their value is 0, otherwise. $Pr(o_r | \langle r', o_l, o_f \rangle)$ represents the probability that the sender responds with the report o_r given the ground truth of the system state r' and (o_l, o_f) . It is worth to note that neighbouring agents with the same trustworthiness and freshness level (o_l, o_f) can have different probabilities to send o_r given r' . Our model considers the observation function to represent the behaviour of all neighbouring agents rather than one single agent. That is, $Pr(o_r | \langle r', o_l, o_f \rangle)$ reflects the overall behaviour of neighbouring agents. Further, $Pr(o_r | \langle r', o_l, o_f \rangle)$ is unknown and needs to be learned. For ACT_i actions, we set their observation function as a uniform distribution as they do not contribute in gathering the state information.

4.2.2 Belief Update

Given an observation $o = \langle o_r, o_l, o_f \rangle$, current state $\mathbf{s} = \langle r, l, f \rangle$ and the next state $\mathbf{s}' = \langle r', l', f' \rangle$, the belief update based on Eqn. 3.6 is given by,

$$b'(\langle r', l', f' \rangle) \propto \mathbf{O}(o | \mathbf{s}', Q) \sum_{\mathbf{s}} \mathbf{P}(\mathbf{s}' | \mathbf{s}, Q) b(\mathbf{s})$$

From Eqn. 4.3, $\mathbf{O}(o|\mathbf{s}', Q) = 0$, when $l' \neq o_l$ or $f' \neq o_f$. Thus $b'(\langle r', l', f' \rangle) = 0$ for the same cases. Let $b'(r')$ denote the next belief on r' , we have:

$$\begin{aligned} b'(r') &= \sum_{l' \in \mathcal{L}, f' \in \mathcal{F}} b'(\langle r', l', f' \rangle) = b'(\langle r', o_l, o_f \rangle) \\ b'(r') &\propto Pr(o_r | \langle r', o_l, o_f \rangle) \sum_{\mathbf{s}} \mathbf{P}(\mathbf{s}' | \mathbf{s}, Q) b(\mathbf{s}) \end{aligned}$$

Substituting Eqn. 4.2 and note the $\delta_{r,r'}$, we further have,

$$\sum_{\mathbf{s}} \mathbf{P}(\mathbf{s}' | \mathbf{s}, Q) b(\mathbf{s}) = \sum_{l \in \mathcal{L}, f \in \mathcal{F}} \frac{1}{|\mathcal{L}| |\mathcal{F}|} b(\langle r', l, f \rangle) \quad (4.4)$$

Note that $b(\langle r', l, f \rangle) \neq 0$, when $l = l_p$ and $f = f_p$, where l_p and f_p denote the trustworthiness and freshness level of the previous message sender. Thus,

$$\sum_{\mathbf{s}} \mathbf{P}(\mathbf{s}' | \mathbf{s}, Q) b(\mathbf{s}) = \frac{1}{|\mathcal{L}| |\mathcal{F}|} b(\langle r', l_p, f_p \rangle) \quad (4.5)$$

If $b(r')$ is the current belief on r' ,

$$b(r') = \sum_{l \in \mathcal{L}, f \in \mathcal{F}} b(\langle r', l, f \rangle) = b(\langle r', l_p, f_p \rangle)$$

Substituting for $b(\langle r', l_p, f_p \rangle)$ in Eqn. 4.5,

$$\sum_{\mathbf{s}} \mathbf{P}(\mathbf{s}' | \mathbf{s}, Q) b(\mathbf{s}) = \frac{1}{|\mathcal{L}| |\mathcal{F}|} b(r')$$

Thus the belief update is given by:

$$b'(r') \propto Pr(o_r | \langle r', o_l, o_f \rangle) \cdot b(r') \quad (4.6)$$

Note that Eqn. 4.6 is actually Bayesian inference. Therefore, our POMDP model implicitly incorporates Bayesian inference along with the decision-making process.

4.2.3 Learning the Observation Function

The existing methods for learning the POMDP parameters focus on improving the learning speed and decreasing the approximation error [38]. However, most of them

assume the POMDP model to be stationary and thus are not suitable when neighbouring agents change with time and hence their behaviour (represented by the observation function) also changes. Given the dynamic behaviour of neighbouring agents, existing works need a long learning period to override the historical data before adapting to the changed model. To address this issue, more importance can be given to the recent experience [51], however, doing so ignores the valuable historical information that can be useful in the settings where malicious agents change their behaviour frequently.

Here, we propose Algorithm 2 to learn the observation function of our POMDP model in the scenarios where the behaviour of neighbouring agents keeps changing. We know from Eqn. 4.3 that learning the observation function implies learning the probability distribution $Pr(o_r|\langle r', o_l, o_f \rangle)$. In the below, we first describe how to compute $Pr(o_r|\langle r', o_l, o_f \rangle)$ based on received observations. Then, we explain how to deal with the changing behaviour.

For each $\mathbf{s}' = \langle r', l' = o_l, f' = o_f \rangle$, we use a Dirichlet distribution $Dir^{\mathbf{s}'}(\phi_{o_r}^{\mathbf{s}'}, o_r \in \{v_1, v_2, \dots, v_n\})$ to maintain the prior over $Pr(o_r|\mathbf{s}')$. Note that the ground truth of the system state r' is available from the centralised server (although with some delays). Therefore, an agent can split the received observations into multiple observation histories based on the state. That is, the observation history $\vec{h}^{\mathbf{s}'}$ consists of the set of observations received when the next state is $\mathbf{s}'$. Let $N(o_r|\vec{h}^{\mathbf{s}'})$ denote

Algorithm 2: Learning the Observation Function

```

1 Initialise model  $P_{cur}$ , history set  $H$  and time window  $W$ .
2 while operating do
3   if a new experience  $e$  is received then
4     initialise a new Dirichlet distribution  $Dir^{\mathbf{s}'}(\phi_{o_r}^{\mathbf{s}'})$  and compute its
       posterior using  $e$ ;
5     for each  $\mathbf{s}' \in \mathbf{S}$  do
6       Compute  $\vec{v}_e$  from  $Dir^{\mathbf{s}'}(\phi_{o_r}^{\mathbf{s}'})$  (Eqn. 4.7) and add it to  $H$ ;
7       Cluster the vectors in  $H$  using K-Means;
8       Find the cluster  $C$  which contains  $\vec{v}_e$ ;
9       Find its cluster center  $\bar{C}_{\mathbf{s}'}$ ;
10      Clear history set  $H$ ;
11      initialise a POMDP model  $P$  with observation function as
          $\{\bar{C}_{\mathbf{s}'}|\mathbf{s}' \in \mathbf{S}\}$ ;
12      Set  $P$  as  $P_{cur}$ ;
```

the number of the observation o_r in $\vec{h}^{s'}$. Given $\vec{h}^{s'}$, the posterior $Pr(o_r|s')$ can be computed based on Bayesian inference,

$$Pr(o_r|s') = \frac{\phi_{o_r}^{s'} + N(o_r|\vec{h}^{s'})}{\sum_{j=1}^{|\Omega|} [\phi_{o_j}^{s'} + N(o_j|\vec{h}^{s'})]} \quad (4.7)$$

To handle the behaviour change, we periodically collect observation histories. That is, the response messages from neighbouring agents as well as the ground truth from the centralised server are gathered over a stipulated time period and constitute *experience*. Each *experience* forms a set of observation histories $\{\vec{h}^{s'}|s' \in \mathbf{S}\}$ and is used to formulate the posterior $\{Pr(o_r|s')|o_r \in \{v_1, v_2, \dots, v_n\}\}$ for each $s' \in \mathbf{S}$ in that time period. With time, agents gather a set of *experience*. Based on this set, the $Pr(o_r|s')$ for every time period is determined. For each s' , the set $\{Pr(o_r|s')|o_r \in \{v_1, v_2, \dots, v_n\}\}$ is a vector $\vec{v}_e$ whose length is n . We then use K-Means clustering to cluster $\{\vec{v}_e\}$ across different time periods based on the Euclidean distance. Doing so allows us to determine the recurrent behaviour patterns of neighbouring agents under the state s' . Once the clusters are formed, we locate the cluster C which contains the $\vec{v}_e$ that represents the most recent *experience*/*-time period*. This cluster represents the current behaviour pattern of neighbours under s' . Though this cluster might not be the representative of the neighbours who change their behaviour immediately after the most recent *experience* was collected, this scenario is rare.

Let $\{Pr_i(o_r|s')|o_r \in \{v_1, v_2, \dots, v_n\}\}$ denote the i th vector in C , $\vec{h}_i^{s'}$ denote the observation history resulting in $\{Pr_i(o_r|s')\}$, and $\{\widehat{Pr}(o_r|s')|o_r \in \{v_1, v_2, \dots, v_n\}\}$ denote the vector derived based on the most recent *experience*, we compute the cluster center $\bar{C}_{s'}$ as,

$$\bar{C}_{s'} = \left\{ \frac{\sum_{i=1}^{|C|} Pr_i(o_r|s') \cdot |\vec{h}_i^{s'}| \cdot \Delta P_i}{\sum_{i=1}^{|C|} |\vec{h}_i^{s'}| \cdot \Delta P_i}, \forall o_r \in \{v_1, v_2, \dots, v_n\} \right\} \quad (4.8)$$

where $\Delta P_i = 1 - \sqrt{|Pr_i(o_r|s') - \widehat{Pr}(o_r|s')|}$. $\bar{C}_{s'}$ represents the current observation function given the state being s' . After we compute the $\bar{C}_{s'}$ for each s' , the set $\{\bar{C}_{s'}|s' \in \mathbf{S}\}$ can be used to initialise a new POMDP model that is able to handle the behaviour of current neighbouring agents. $|\vec{h}_i^{s'}|$ in Eqn. 4.8 normalizes the effect of observation histories of different lengths. ΔP_i degrades the effect of the vectors which are not that similar to $\{\widehat{Pr}(o_r|s')|o_r \in \{v_1, v_2, \dots, v_n\}\}$. This is meaningful

when the preset cluster number is small and thereby the vectors clustered into the same cluster might not be very similar.

The details of Algorithm 2 are described as follows. Let P_{cur} be the POMDP model whose optimal policy is executed to obtain a new *experience* for the current time period, H be the set of vectors $\{\vec{v}_e\}$ collected across all time periods and W denote the time window, i.e., length of each time period. For each $\mathbf{s}'$, when a new experience is obtained, the corresponding $\vec{v}_e$ is derived (lines 6), after which the K-Means algorithm is used to determine the clusters (line 7). Each identified cluster corresponds to a behaviour pattern. We locate the cluster which contains the recent $\vec{v}_e$ and derive the $\bar{C}_{\mathbf{s}'}$ using Eqn. 4.8 (line 8-9). Finally, we initialise a new POMDP model with the new observation function $\{\bar{C}_{\mathbf{s}'} | \mathbf{s}' \in \mathcal{S}\}$ and set the new model as P_{cur} (line 11-12) that advises all the actions to be taken in the next time window. Note that this algorithm requires initializing a POMDP model and computing its optimal policy periodically. This is practical only when the model's state space is small, and our design satisfies this condition.

The above process needs to perform a clustering operation for each $\mathbf{s}'$, i.e. $n|\mathcal{L}||\mathcal{F}|$ clustering operations in total. In the case that $n = 2$, i.e. $o_r \in \{v_1, v_2\}$, we can further speed up the process. Specifically, as $Pr(o_r = v_2 | \mathbf{s}') = 1 - Pr(o_r = v_1 | \mathbf{s}')$, the set $\{Pr(o_r | \mathbf{s}') | o_r \in \{v_1, v_2\}, \mathbf{s}' \in \mathcal{S}\}$ can be sufficiently represented using a vector $\vec{v}_e = \{Pr(o_r = v_1 | \mathbf{s}') | \mathbf{s}' \in \mathcal{S}\}$, and the vector length $|\vec{v}_e|$ equals to the state space size that is $2|\mathcal{L}||\mathcal{F}|$. Thus, we can use K-Means clustering to cluster $\{\vec{v}_e\}$ across different time periods based on the Euclidean distance. The cluster C which contains the $\vec{v}_e$ representing the most recent *experience/time period* is the current behaviour pattern of neighbours. The center of the cluster can be computed as

$$\bar{C} = \left\{ \frac{\sum_{i=1}^{|C|} Pr_i(v_1 | \mathbf{s}') \cdot |\vec{h}_i^{\mathbf{s}'}| \cdot \Delta P_i}{\sum_{i=1}^{|C|} |\vec{h}_i^{\mathbf{s}'}| \cdot \Delta P_i}, \forall \mathbf{s}' \in \mathcal{S} \right\} \quad (4.9)$$

where $\Delta P_i = 1 - \sqrt{|Pr_i(v_1 | \mathbf{s}') - \widehat{Pr}(v_1 | \mathbf{s}')|}$. Note that each element in the vector $\bar{C}$ corresponds to a different $\mathbf{s}'$, thus we only need to perform the clustering operation once and $\bar{C}$ represents the current observation function.

4.2.4 Model Analysis

In this subsection, we analyse how our POMDP model can maximise the utility, $g(b_t) - h(t)$, given in Eqn. 4.1 and the complexity of our model.

Recall that given a POMDP model, Eqn. 3.8 computes the utility of an action α under the belief b , i.e.,

$$Q^*(b, \alpha) = \sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) \mathbf{R}(\mathbf{s}, \alpha) + \gamma \sum_{o \in \Omega} \mathbf{P}(o|b, \alpha) V^*(b^{\alpha o})$$

For $\alpha \in \{ACT_i | i = 1, \dots, n\}$, given our design of transition function and observation function, it is obvious that $b^{\alpha o}$ is a uniform belief and $\mathbf{P}(o|b, \alpha)$ is a constant value. Since both $b^{\alpha o}$ and $\mathbf{P}(o|b, \alpha)$ are independent of b and α , which α results in a higher value of $Q^*(b, \alpha)$ given b is only determined by $\sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) \mathbf{R}(\mathbf{s}, \alpha)$. Note that $\sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) \mathbf{R}(\mathbf{s}, \alpha)$ computes the expected reward of taking the action α given the belief b , which is actually $g(b_t)$.

For $\alpha = Q$, $\sum_{\mathbf{s} \in \mathcal{S}} b(\mathbf{s}) \mathbf{R}(\mathbf{s}, \alpha)$ is the cost of collecting an additional message, which corresponds to $h(t)$. Note that given a new message, the belief b will be updated as $b^{\alpha o}$ and the vehicle may gain a higher expected reward for taking an action from $\{ACT_i | i = 1, \dots, n\}$. Therefore, $\sum_{o \in \Omega} \mathbf{P}(o|b, \alpha) V^*(b^{\alpha o})$ represents the expected utility of waiting for more information.

From the above analysis, we can see that in every step, the POMDP model determines whether it can get a higher expected utility if it queries more information. When the belief is highly uncertain, $g(b_t)$ is small and the POMDP model will prefer to wait for more messages. However, when the belief is certain to some extent, the POMDP model will stop collecting information so that it can save the unnecessary cost.

For the complexity of the POMDP model, the main factor is the size of state space. For our model, the size of state space is $n \times 3 \times 2 = 6n$. Note that the state space size is independent of the number of agents because we only consider the trustworthiness of the current message's sender. In comparison, the approaches [28–30] that integrate POMDP with trust management for decision-making consider the trustworthiness of all agents that can send messages. Therefore, their state space size is linear with the number of agents, which makes them infeasible for large-scale multi-agent systems (e.g. VANETs).

4.3 Experiments

In this section, we perform the experiments in the domain of VANETs. In VANETs, each vehicle is regarded as an agent and vehicles can join and leave the system freely. In our experiments, the system state that agents are interested in is whether there is an accident (event) in the route planned by them or not. If an accident exists, agents will take another route to avoid the possible traffic jam due to that accident. If there is no accident, agents will stay in the current route. Therefore, the value of r and o_r is either *yes* or *no* where *yes* means there is an accident and *no* means there is no accident. Accordingly, besides $query(Q)$, the POMDP actions are $reroute(RE)$ which means changing to another route and $ignore(IG)$ which means staying in the current route. Vehicles exchange information by the DSRC technique. When an accident happens, the vehicles that directly observe it can broadcast alert messages. The vehicles, which receive the alert messages but cannot observe the accident, can broadcast query messages to gather more information. Moreover, vehicles can download new certificates from the centralised server when driving by the Road Side Units (RSUs) [52]. They can also upload received messages to the server through RSUs.

We conduct extensive experiments to compare the performance of our POMDP-based approach with that of threshold-based trust schemes. We evaluate the performance based on the decision quality and cost/delay under static behaviour settings. Also, we analyse the robustness of our proposed algorithm for learning the observation function under dynamic behaviour settings.

4.3.1 Simulation Setup

We evaluate our POMDP model using the *VEINS* simulator [25], that combines the OMNeT++ for vehicle-to-vehicle communication simulation, with the SUMO for road traffic microsimulation. We periodically set the speed of some vehicles to zero in order to mimic traffic accidents. When a vehicle is close enough (25 meters) to an accident's location, we deem that the vehicle has directly observed the event and can broadcast alert or response messages. We also add the trust management function to the Scenario Manager of *VEINS* simulator to make it behave as the server. The initial trustworthiness of vehicles is 0.5. In POMDP, we regard the

trustworthiness as *high* when its value is larger than 0.8, *low* when the value is less than 0.3 and *medium* otherwise. The freshness level is *fresh* if the trustworthiness is updated within 30 seconds (s) and *old* otherwise. The reward functions used in our experiments are: $\mathbf{R}(r = \text{yes}, RE) = -\mathbf{R}(r = \text{yes}, IG) = 30$, $\mathbf{R}(r = \text{no}, IG) = -\mathbf{R}(r = \text{no}, RE) = 20$ and $\mathbf{R}(\mathbf{s}, Q) = 1$. We set $\mathbf{R}(\mathbf{s}, Q) = 1$ to encourage *query* action when the belief on r is highly uncertain. 210 vehicles join the simulation within 120s and ten of them are used to mimic accidents. When a vehicle arrives at its destination, it will choose a new random destination. Therefore, every vehicle is active until the simulation ends. Averagely, there is one accident every 35s and each accident lasts about 120~180s. The delay of getting the ground truth from the centralised server is set to 60s.

We compare our POMDP model with Dempster-Shafer Theory (DST) and Bayesian inference (BI) [8] and reputation-based announcement (RBA) [10]. [8] maintains two beliefs for each event, i.e., b_r : *event happens* and b_f : *event does not happen*. For [8], we compare the difference between b_r and b_f , i.e., $|b_r - b_f|$, with a threshold to decide whether to re-route or not. In RBA, we multiply the sender's trustworthiness with a time-decay factor and compare the result with a threshold to make the decision. We use SARSOP [53] to compute the POMDP policy. We perform K-means clustering by the Cluster 3.0 software³. We evaluate the decision-making performance under both static and dynamic scenarios.

The evaluation metrics are: 1) precision: $\frac{TP}{TP+FP}$ where TP is true positive, means how many times vehicles re-route when events exist and FP is false positive, means how many times vehicles re-route when events are fake; 2) recall: $\frac{TP}{TP+FN}$ where FN is false negative, means how many times vehicles ignore when events exist; 3) decision cost: how many messages are processed before making each decision; 4) decision delay: how long a vehicle needs to make a decision after receiving the first message about a certain event.

4.3.2 Malicious Behaviours

In our simulation, we have considered both static and dynamic malicious behaviours described as follows.

³<http://bonsai.hgc.jp/~mdehoon/software/cluster/software.htm>

Static: Vehicles report truthful opinions with probability p and lie with probability $(1 - p)$. The probability p does not change with time.

DynamicI: Malicious vehicles report truthful opinions until their trustworthiness value reaches *high*, after which they provide complementary opinions.

DynamicII: Similar to *DynamicI*, vehicles can report truthful opinions only when their trustworthiness is *low*.

The update of trustworthiness is set as: vehicles will update their trustworthiness in a long interval (60s) if they have lied recently while they will update the trustworthiness frequently (every 20s) if they have reported truth.

4.3.3 Static Behaviour Scenario

In the static scenario, we have evaluated the performance when malicious vehicles exhibit *Static* behaviour with $p = 0.5$ and $p = 1$, and the percentage of malicious vehicles mp is 0.2 and 0.4. The overall driving period is 3600s. The results are shown in Figure 4.1. The performance of threshold-based schemes is evaluated under different thresholds. However, since there is no threshold setting in POMDP, we evaluate the results under different time windows. Thus, the x-axis is the threshold being from 0.1 to 0.8 for threshold-based schemes and the time window W being from 10s to 80s for POMDP. We set the cluster number of K-Means clustering as 1 for this scenario, i.e. POMDP learns the observation function without detecting behaviour patterns.

Figure 4.1 shows that RBA and DST have low precision and high recall when setting a low threshold. This is because a low threshold value allows a high uncertainty about the event occurrence and vehicles choose to re-route most of the time, i.e. large FP and low FN . When the threshold increases, these schemes seek more messages to reduce the uncertainty, and thus have higher precision. However, a high threshold also incurs more cost/delay for DST and RBA especially when mp is large. Compared to threshold-based schemes, POMDP has the lowest cost/delay under different time windows and different behaviour settings. Our POMDP model is more efficient because it uses the Bayesian theory to update its belief. The opinion from the vehicles with low trustworthiness will strengthen the belief about the opposite of that opinion, i.e. false information also contributes to the state

FIGURE 4.1: Performance evaluation under static behaviour

belief. Thus, POMDP often just needs one or two messages to achieve a high state belief. BI has the second lowest cost/delay due to the same reason. Also, because of this reason, the difference between beliefs $|b_r - b_f|$ is easy to reach a relatively high value for BI. Therefore, the influence of threshold setting on the performance of BI is not that obvious.

For the decision quality, the POMDP solution has the lowest recall and lower precision when the threshold of DST and RBA is high. This is due to the high FN and FP which occur at the beginning of simulation (called *warm up period*). In particular, at the beginning, the trustworthiness of benign and malicious vehicles are both (*medium, fresh*) while malicious vehicles send lots of false alerts. Thus, the true information from benign vehicles seems untrustworthy and makes vehicles believe in the opposite as POMDP updates its belief with the Bayesian theory. This effect will disappear after benign and malicious vehicles have different trustworthiness values and POMDP learns the $Pr(o_r|\mathbf{s}')$ given different $\mathbf{s}'$. We collect the precision and recall data in every 120s and show this process in Figure 4.2. It shows that the recall and precision increase to a high value with time. While different settings of mp and p affect the increasing speed, the trend is the same, i.e. POMDP can handle the malicious behaviour very well through learning. Thus, the decision quality of POMDP is actually at least as good as that of threshold-based

FIGURE 4.2: The warm up period in POMDP ($W = 60s$)

schemes if leaving out the warm up period. Overall, our model has the same if not better decision quality than threshold-based schemes while its decision cost/delay is the lowest.

4.3.4 Dynamic Behaviour Scenario

In this subsection, we evaluate the performance of our POMDP model under the dynamic behaviour setting. Specifically, we set the overall simulation period as 6000s, and the precision, recall and cost data are collected every 150s. The percentage of malicious vehicles mp is set as 0.3. Malicious vehicles exhibit *Static* behaviour with $p = 1$ till 2000s, after which they exhibit *DynamicII* behaviour until 4000s. Then they change the behaviour to *DynamicI* after that. We have tested the performance when the cluster number equals to 1, 2, 3, and 5 respectively. The time window W is set to 60s. The results about the precision and recall are shown in Figure 4.3.

From Figure 4.3a, we can observe that when the simulation time is around 2000s, the recall value decreases significantly. This is because the malicious vehicles that have *low* trustworthiness start to tell the truth, which is totally opposite to their previous behaviour. Note that until 2000s, the learned observation function represents that vehicles with *low* trustworthiness always report fake information. Then, when malicious vehicles report a true event, other vehicles will choose to believe that the event does not exist and ignore the information. Therefore, the *FN* increases, and thus the recall value decreases. After the simulation time passes 2000s,

FIGURE 4.3: Decision quality under dynamic behaviour

we can observe that the recall value recovers fast when applying the clustering operation. The reason is that the new experience after 2000s is quite different with previous ones. With clustering, the new experience dominates the learning of the new observation function. Then, the new observation function quickly reflects that vehicles with *low* trustworthiness tell true information. Given the new observation function, the *FN* decreases and the recall value increases. In comparison, the POMDP model needs more time to adapt to the new behaviour without clustering (i.e. cluster number being 1). When the simulation time reaches 4000s, malicious vehicles change their behaviour to *DynamicI*. However, the influence is limited because most of the malicious vehicles do not have *high* trustworthiness at that time and they will keep telling the truth, which is similar to their previous behaviour.

About the impact of cluster number setting, we can find from Figure 4.3b that after the first behaviour change, the POMDP model with the cluster number being 5 starts to have a low precision. This is because the cluster number is too large for the existing *experience*. This results in that each cluster only has several vectors. In this case, the learning algorithm can not catch the current behaviour appropriately. This implies that the setting of the cluster number should fit with the amount of *experience*. The cost of POMDP under the dynamic behaviour is shown in Figure 4.4. It indicates that during the warm up period, the cost of POMDP is relatively high. After that period, the cost of the POMDP model with the cluster number being 1, 2, and 3 is close to 1 even under the behaviour change. For the POMDP model that sets the cluster number as 5, its cost starts to increase after the first behaviour change due to failing to learn the behaviour pattern correctly.

FIGURE 4.4: Cost under dynamic behaviour

4.4 Summary

In this chapter, we propose a POMDP-based approach to assist efficient decision-making for using the shared information about unobservable system states. The POMDP model queries the information about uncertain states from neighbouring agents while taking into account their potential malicious behaviours. We conduct experiments under the static scenarios where malicious agents do not change their behaviour. The results demonstrate that the POMDP model offers a better balance between the decision quality and the cost/delay incurred during the information collection compared to the heuristic threshold-based schemes. Moreover, this work also considers that in real life the behaviours of neighbouring agents are unknown. Thus, the observation function that represents the behaviour of neighbours needs to be learned. We propose a learning algorithm that can handle the dynamic scenarios where malicious agents change their behaviour from time to time. The experiments under the dynamic scenarios verify that the learning algorithm can learn the observation function efficiently while still handling the behaviour changes effectively.

This chapter focuses on the scenarios where agents can share information with one another. However, with the increasing number of intelligent agents, there are many situations where agents cannot share information. In the next chapter, we consider the ad hoc teamwork problem, i.e. how an agent performs the teamwork with unknown teammates without relying on communication or shared team strategies.

Chapter 5

ATSIS: Achieving Ad hoc Teamwork by Sub-task Inference and Selection¹

The ad hoc teamwork problem has been proposed for the situation where the ad hoc agent has no communication or shared strategy with its teammates [7]. To achieve teamwork effectively, the ad hoc agent needs to select the actions that result in the highest utility. Formally, we model the teamwork problem with the multi-agent goal-oriented MDP (see Chapter 3.1.3 for details). Let i denote the index of the ad hoc agent. Recall Eqn. 3.10, the action a^* the ad hoc agent tries to find in every step is as below,

$$a^* = \operatorname{argmax}_{a_i} Q(s, a_i \vee \vec{a}_{-i}) = \operatorname{argmax}_{a_i} \sum_{s' \in S} P(s'|s, a_i \vee \vec{a}_{-i}) [R(s, a_i \vee \vec{a}_{-i}, s') + \gamma V^\pi(s')] \quad (5.1)$$

where $\vec{a}_{-i}$ is the joint action of all teammates and $a_i \vee \vec{a}_{-i}$ constitutes the joint action $\vec{a}$. In simple domains, we assume that teammates' actions are fully observable. Therefore, the ad hoc agent can rely on teammates' actions to understand their behaviour. Existing works learn the behaviour models of the teammates encountered previously based on the observed state-action pairs. The behaviour models can predict teammates' actions (i.e. $\vec{a}_{-i}$) given a state. Then, the ad hoc agent can choose its own action based on the prediction of teammates' actions so that the joint action can maximise the team utility. However, previous teammates can

¹This chapter contains the work published in [54].

behave very differently from current teammates. Inaccurate behaviour models can lead to wrong predictions of teammates' actions, and thus, compromise teamwork. Therefore, the problem this chapter aims to address is as below:

Problem: how the ad hoc agent achieves ad hoc teamwork without relying on teammates' behaviour models.

In this chapter, we propose the Ad Hoc Teamwork by Sub-task Inference and Selection (ATSIS) algorithm that uses a sub-task inference instead of learning behaviour models. In detail, ATSIS works as follows. We assume the task as a set of sub-tasks. The ad hoc agent infers which sub-task each teammate is pursuing by observing the effects of their actions. Since the inference of teammates' sub-tasks has uncertainty, we select the ad hoc agent's sub-task by a POMDP model. Then, the ad hoc agent pursues the selected sub-task, so that all sub-tasks can be achieved. Finally, we use MCTS to choose the actions for completing the chosen sub-task. In the following sections, we first introduce the background of our problem. Second, we present the details about how the ad hoc agent infers the sub-task each teammate is contributing to. Third, we describe the POMDP model that selects the sub-task for the ad hoc agent. Fourth, we discuss how to find the best action for the ad hoc agent to pursue the selected sub-task by MCTS. Finally, we demonstrate the advantages of ATSIS over the literatures through extensive experiments.

5.1 Background

In this chapter, we model the teamwork as a multi-agent goal-oriented MDP. We assume that the state space S can be divided into a set of components M , i.e. $S = \times_{e=1}^{|M|} S_e$. Each component represents a feature of a domain, for instance the position of each agent. *Environmental components* cannot be controlled by the team members. *Team components* can be controlled by the team members. The goal of the teamwork is to make all team components reach the desired values in a goal state (through state transitions). We assume to have the domain knowledge that can specify the desired values of team components given the values of environmental components. Given the domain knowledge, we can consider the task as a set of sub-tasks. We define a sub-task as a function that takes as input the values of environmental components and returns one desired value of team components

in a goal state. For instance, a sub-task “*rescue a victim*” will return the desired agent position given the victim’s position. To achieve a sub-task, a certain team component needs to reach the desired value returned by the sub-task. Let $\mathcal{T}$ denote the set of sub-tasks identified by $\{1, 2, \dots, |\mathcal{T}|\}$.

Note that teammates’ underlying behaviour takes states as input and outputs actions to contribute to the task. Because the completion of the task needs the achievement of all sub-tasks, each teammate’s actions will help a certain team component to reach the desired value returned by a certain sub-task. Therefore, we can infer teammates’ desired sub-tasks by observing their actions.

5.2 AT SIS Algorithm

In this section, we explain the details of AT SIS algorithm. In subsection 5.2.1, we explain how the ad hoc agent infers teammates’ desired sub-tasks by observing their actions. In subsection 5.2.2, we present the POMDP design which models the selection of the ad hoc agent’s sub-task. In subsection 5.2.3, we explain how we use MCTS to map a selected sub-task to a concrete MDP action. Finally, in subsection 5.2.4, we analyse the complexity of the algorithm.

5.2.1 Inferring Sub-tasks Pursued by Teammates

In this subsection, we explain how AT SIS infers which sub-tasks the teammates are pursuing without relying on their behaviour models. Since we consider a scenario where the ad hoc agent replaces one of the team members, we expect teammates to collaborate, i.e. their actions help to complete some sub-tasks. Therefore, we can use the effects of their actions to infer their desired sub-tasks. To do so, we check for each sub-task whether a teammate’s observed action brings the most utility addition to that sub-task with respect to all the feasible actions of the teammate.

To compare the utility addition, we compute the Q-value of each of the teammate’s actions. Note that the computation of Q-value requires a policy for achieving the sub-task. To generate a policy, the ad hoc agent models the sub-task as a single-agent MDP. It assumes that the environment and other team members are static as

we do not want to rely either on teammates' models or the environmental model. Then, the ad hoc agent solves the MDP to find the policy.

In the remainder of this chapter, we use subscript to denote the index of team members and superscript to denote the index of sub-tasks. Let $\pi^l : S_j \times A_j \rightarrow [0, 1]$ be the policy used by the ad hoc agent to compute the Q-value of teammates' actions for achieving sub-task l . The Q-value of action a_u is:

$$Q^l(s_j, a_u) = \sum_{s'_j \in S_j} P(s'_j | s_j, a_u) [R^l(s_j, a_u, s'_j) + \gamma \sum_{a \in A_j} \pi^l(a | s'_j) Q^l(s'_j, a)] \quad (5.2)$$

where s_j is the value of the team component controlled by the teammate j in the previous time step and the reward $R^l(s_j, a_u, s'_j)$ is positive when the sub-task l is achieved, and negative otherwise. Note that π^l is the policy for the sub-task l . Since π^l assumes the environment and teammates to be static, Eqn. 5.2 considers only s_j and a_u . Let $\{a_u^l\}$ be the set of actions resulting in the largest Q-value for the sub-task l (there may be multiple actions returning the same largest Q-value). If the teammate j 's action a_j in the last time step is present in $\{a_u^l\}$, we infer the sub-task l to be possibly pursued by the teammate j . We repeat this process for all sub-tasks.

Note that a_j may result in the largest Q-value for multiple sub-tasks. Thus, it is uncertain which sub-task the teammate j contributes to. Also, when the teammate j selects its action, it may consider the future states of the environment and other team members. For instance, it may predict the position of teammates and take the action to avoid a collision. However, the policy π^l assumes the environment and teammates to be static, i.e. it does not consider the future states of the environment and teammates. Therefore, the inference of the sub-tasks that teammates are performing is not always accurate. Given all those uncertainties, the ad hoc agent still needs to select the sub-task for itself. Hence, in the next subsection, we present the POMDP model that the ad hoc agent uses to select its sub-task given its (uncertain) inference of teammates' sub-tasks.

5.2.2 Selecting the Sub-task by POMDP

In this subsection, we present the POMDP model for the ad hoc agent's sub-task selection. There are two possible cases in this scenario. First, the ad hoc agent is

highly confident which sub-task is unattended. Then, it should simply select that sub-task. Second, the ad hoc agent is very uncertain which sub-tasks its teammates are pursuing or it believes that there are multiple unattended sub-tasks. In this case, the ad hoc agent should select the sub-task it is the most suitable for based on the current MDP state. Hence, the ad hoc agent also cares about its suitability for each sub-task compared to its teammates. We present the details of the POMDP below.

State. A POMDP state $\mathbf{s} \in \mathbf{S}$ is defined as a tuple $\langle \{t_j | j = 1, \dots, |N| \text{ and } j \neq i\}, \{g^l | l = 1, \dots, |\mathcal{T}|\} \rangle$. We denote i as the index of the ad hoc agent. $t_j \in \{1, \dots, |\mathcal{T}|\}$ denotes the sub-task the teammate j pursues. $g^l \in \{1, \dots, |N|\}$ represents how suitable the ad hoc agent is for the sub-task l compared to its teammates. If the agent is the r -th most suitable for the sub-task l , then $g^l = |N| - r + 1$. We consider that the ad hoc agent is more suitable for a given sub-task compared to a given teammate when its utility for the performance of this sub-task is higher than that teammate. We provide the details of the utility function later on in Eqn. 5.3.

Action. In each time step, our POMDP model makes a decision about which sub-task the ad hoc agent shall pursue. Thus, the action space $\mathbf{A}$ is $\{\alpha_k | k = 1, 2, \dots, |\mathcal{T}|\}$ where α_k denotes that the ad hoc agent should pursue the sub-task k .

Transition function. The POMDP state component t_j is controlled by the teammate j . g^l depends on the environment and other agents' states. Since the ad hoc agent has no knowledge about the teammate j 's decision-making process and thus, does not know the teammates' future states, we set their transition function as uniform distribution, i.e. $\forall \alpha_k, \mathbf{P}(t'_j | t_j, \alpha_k) = \frac{1}{|\mathcal{T}|}$ and $\mathbf{P}(g^{l'} | g^l, \alpha_k) = \frac{1}{|N|}$.

Observation. The observation provides information for the partially observable POMDP state. Corresponding to our POMDP state, the observation $\mathbf{o} \in \mathbf{\Omega}$ can be defined as a tuple $\langle \{o_j | j = 1, \dots, |N| \text{ and } j \neq i\}, \{o_l | l = 1, \dots, |\mathcal{T}|\} \rangle$, where i is the index of the ad hoc agent. Specifically, o_j represents the inference about which sub-task the teammate j is contributing to. Since multiple sub-tasks could be inferred as being pursued by the teammate j , o_j takes its value from the power set of $\{1, \dots, |\mathcal{T}|\}$. For example, if $o_j = \{1, 2, |\mathcal{T}| - 1\}$, it means the ad hoc agent

i infers that the teammate j is trying to pursue the sub-task 1, 2 or $|\mathcal{T}| - 1$. o_l takes its value from $\{1, \dots, |N|\}$ and represents how suitable the ad hoc agent is for the sub-task l with respect to all other agents. The more suitable it is, the higher the value is. In each time step, these observations are obtained as follows: 1) the ad hoc agent initialises o_j with an empty set. Next, for each sub-task l , the ad hoc agent uses the Eqn. 5.2 to infer if the teammate j is pursuing that sub-task. If so, it adds l to o_j ; 2) to obtain o_l , the ad hoc agent computes the utility gained by each agent j (including the ad hoc agent) if the agent j achieves the sub-task l from the current state. If s_j represents the value of the team component controlled by the agent j in the current time step and π^l is the same policy used in Eqn. 5.2, the utility of the agent j in achieving the sub-task l is given by,

$$V^l(s_j) = \sum_{a_j \in A_j} \pi^l(a_j|s_j) \sum_{s'_j \in S_j} P(s'_j|s_j, a_j) [R^l(s_j, a_j, s'_j) + \gamma V^l(s'_j)] \quad (5.3)$$

The agents are ranked based on their $V^l(s_j)$ value. If the value for the ad hoc agent is the r -th largest value, then the observation $o_l = |N| - r + 1$.

Observation function. The observation function handles the uncertainty in the received observations. The observation function for o_j is given by,

$$\forall \alpha_k, \mathbf{O}(o_j|t'_j, \alpha_k) = \begin{cases} \frac{\mu}{2^{|\mathcal{T}|-1}} & \text{if } t'_j \in o_j \\ \frac{1-\mu}{2^{|\mathcal{T}|-1}} & \text{otherwise} \end{cases} \quad (5.4)$$

where $\mu \in (0, 1]$ represents the ad hoc agent's confidence that the policy π^l matches the behaviour model of the teammate j . Using the variable μ allows controlling the incorrect belief updates. These inaccuracies may arise when teammates' actions are based on either the changes of other agents' behaviours or the environment. In particular, if we set $\mu = 1$, then $\mathbf{O}(o_j|t'_j, \alpha_k) \neq 0$ only when $t'_j \in o_j$. Imagine that the teammate j is pursuing the sub-task l , i.e. $t'_j = l$. In the last time step, it took an action a_j to avoid a conflict with other agent making it deviate from the best path to achieve the sub-task. Then, given the Eqn. 5.2, a_j does not result in the largest Q-value for the sub-task l . This leads to an inaccurate observation o_j (that does not contain l), and causes an erroneous belief about t'_j . Having $\mu = 1$ causes the ad hoc agent to believe that the sub-task l is unattended. When $\mu < 1$, the belief update is more conservative and can better withstand the inaccurate

observation. We divide by $2^{|\mathcal{T}|-1}$ in Eqn. 5.4, to normalize the probability across the $2^{|\mathcal{T}|-1}$ possible observations. Also, the observation function for o_l is given by,

$$\forall \alpha_k, \mathbf{O}(o_l|g^{l'}, \alpha_k) = \begin{cases} \eta & \text{if } o_l = g^{l'} \\ \frac{1-\eta}{|N|-1} & \text{otherwise} \end{cases} \quad (5.5)$$

where $\eta \in (0, 1]$ represents the ad hoc agent's confidence in the accuracy of the utility evaluation calculated by Eqn. 5.3.

Reward function. The ad hoc agent's reward function should offer a trade-off between a desire to fulfil the sub-task when the ad hoc agent is certain which sub-task is unattended, and pursuing the most suitable sub-task. Hence, we design the reward function as an additive function, such that:

$$\mathbf{R}(< \{t_j\}, \{g^l\} >, \alpha_k) = \mathbf{R}_1(\{t_j\}, \alpha_k) + \mathbf{R}_2(\{g^l\}, \alpha_k) \quad (5.6)$$

where $\mathbf{R}_1(\{t_j\}, \alpha_k) = z \times r_1$ and $\mathbf{R}_2(\{g^l\}, \alpha_k) = g^k \times r_2$. z represents the overall number of unique sub-tasks being pursued by the team which is determined by teammates' sub-tasks $\{t_j\}$ and the ad hoc agent's POMDP action α_k . For example, if one agent tries to fulfil the sub-task l_1 while all the remaining agents try to achieve the sub-task l_2 ($l_1 \neq l_2$), then $z = 2$. $g^k \in \{g^l\}$ represents how suitable the ad hoc agent is for the sub-task k chosen by α_k . r_1 and r_2 are the design constants that adjust the decision bias between pursuing the unattended sub-task and the most suitable sub-task. We evaluate the influences of POMDP parameters in Section 5.3.5.

In summary, the ad hoc agent i infers teammates' sub-tasks in every time step. Each inference is a POMDP observation which updates the belief on $\{t_j\}$. Also, the ad hoc agent updates the belief on $\{g^l\}$ based on the current MDP state. The belief $b(t_j = k)$ is the probability that the teammate j pursues the sub-task k . Let $b(k)$ denote the probability that the sub-task k is pursued, then $b(k) = 1 - \prod_{j \neq i} [1 - b(t_j = k)]$. The higher $b(k)$ the more probable that the sub-task k is being pursued. Hence, the ad hoc agent prefers the sub-task k with the lowest overall belief (as it has a higher probability to increase z and thus the expected reward). When there are multiple sub-tasks with the lowest belief, r_2 leads the ad hoc agent to select the most suitable sub-task.

5.2.3 Pursuing the Chosen Sub-task

At every time step, POMDP suggests pursuing a certain sub-task. To achieve that sub-task, we use a popular variant of MCTS, i.e. UCT (Upper Confidence Bound 1 applied to trees) [40]. That is, we use UCT to search for the best action for the ad hoc agent at every time step. The idea is to estimate the long term expected rewards of each possible action by sampling future states. UCT has a simulator that takes a state and an ad hoc agent's action as the input and samples the next state. This simulates the uncertainty resulting from the dynamics of the environment. During the sampling, if the simulator does not have teammates' models, it assumes teammates to be static. Given a sub-task, UCT performs a preset number of simulations. In each simulation iteration, UCT builds a tree where the root node is the current state s . To sample a new state s' , UCT chooses an action a with the largest upper confidence bound in s , i.e. $Q(s, a) + c\sqrt{\frac{\ln n(s)}{n(s, a)}}$ where c is a constant, $n(s)$ and $n(s, a)$ are the numbers of times that s and (s, a) have been visited respectively. UCT expands the tree by adding s' as the child node of s with the action a as the edge between s and s' . Then, UCT chooses the best action a in the new state s' (the child node) and samples again a next state. UCT expands the tree until a termination condition or a given depth is reached. ATSYS has two termination conditions. First is the achievement of the given sub-task. In this case, the simulator returns the reward TR_1 . Second is the fulfilment of the ultimate team's goal. Here, the reward is TR_2 . Once the simulation finishes, UCT updates $Q(s, a)$, $n(s)$ and $n(s, a)$. After a preset number of simulations, UCT returns the action that has the largest $Q(s, a)$.

There are two advantages of using UCT: 1) it can handle a large state/action space efficiently; 2) it increases the flexibility of ATSYS because the simulator's assumption about the teammates' behaviour can be revised on-the-fly. Specifically, the simulator assumes that teammates are static when it has to work without teammates' models. However, the simulator could integrate the learned model which can help to improve the performance.

5.2.4 Complexity Analysis

In this subsection, we analyse the complexity of existing approaches that rely on behaviour models and ATSYS. Recall that the objective of the ad hoc agent is to

find actions to maximise the expected utility. That is,

$$\begin{aligned} a^* &= \operatorname{argmax}_{a_i} Q(s, a_i \vee \vec{a}_{-i}) \\ &= \operatorname{argmax}_{a_i} \sum_{s' \in S} P(s'|s, a_i \vee \vec{a}_{-i}) [R(s, a_i \vee \vec{a}_{-i}, s') + \gamma V^\pi(s')] \end{aligned}$$

To find a^* , existing approaches use teammates' behaviour models to predict $\vec{a}_{-i}$. Note that they not only predict the $\vec{a}_{-i}$ for current state s but also predict the $\vec{a}_{-i}$ for future states because they need to estimate $V^\pi(s')$. In particular, they run UCT simulations from the current state s until a certain depth is reached or the termination condition is met. In every step of simulation, behaviour models output the actions of teammates. Note that UCT simulations try to search all possible states in order to estimate their expected utility. Let $|S_e|$ denote the state space size of one agent. Existing approaches need to search $|S_e|^{|N|}$ joint states that are consist of the states of the ad hoc agent and its teammates. Therefore, the search space of existing approaches can increase exponentially.

In comparison, AT SIS considers the task as a set of sub-tasks. The inference of teammates' sub-tasks and the pursuing of the chosen sub-task only search the state space of one agent while regarding the other team members as stationary. Although for each teammate, AT SIS needs to check whether each sub-task can possibly be that teammate's sub-task, the overall search space is $|N| \times |\mathcal{T}| \times |S_e|$ which increases much slower than $|S_e|^{|N|}$. For the choosing of sub-task by POMDP, the state space is $|\mathcal{T}|^{|N|} \times |N|^{|T|}$. Therefore, when the number of sub-tasks $|\mathcal{T}|$ is large, the complexity of POMDP can be high. However, we can use the existing online POMDP solvers (e.g. [55]) to deal with it.

5.3 Experiments

In this section, we present the details of the experiments that we conducted to evaluate our algorithm. We compare AT SIS with the PLASTIC-Model scheme [20] and the Online Planning for Ad Hoc Teams (OPAT) scheme [18]. PLASTIC is the representative scheme that explicitly models the behaviour of teammates and uses the behaviour model to predict the teammates' actions during planning. In detail, PLASTIC uses UCT with the acquired teammates' models to plan the ad

hoc agent’s actions. OPAT assumes the teammates’ model is given and uses UCT to estimate the utility of joint actions under current state s . They then choose the ad hoc agent’s action based on the history of joint actions in s in order to obtain the maximum utility. We use the online POMDP solver DESPOT [55] to select the sub-task for the ad hoc agent.

We perform four different experiments to pitch ATSiS against the above methods. In subsection 5.3.1, we perform an experiment to show how ATSiS performs with respect to PLASTIC and OPAT when they are provided with (a) correct, and (b) incorrect teammates’ models. The results show that the existing methods are vulnerable to incorrect teammates’ models. ATSiS does not require correct teammates’ models to perform well, which is a huge advantage in scenarios where we lack teammates’ behaviour models [24]. The ad hoc agent can also learn teammates’ behaviour models during the teamwork. Henceforth, we compare our approach to PLASTIC that can learn the behaviour models from scratch. In subsection 5.3.2, we show the benefits of ATSiS compared to PLASTIC under different lengths of learning periods. PLASTIC can also use transfer learning to learn faster new teammates’ models using its knowledge about previous teammates’ behaviour. For this reason, in subsection 5.3.3, we evaluate ATSiS by pitching it to PLASTIC that uses transfer learning. The results show that transfer learning is only effective when the previous teammates are comparable to new teammates. This shows the benefits of our approach. In subsection 5.3.4, we show how to further improve the performance of ATSiS by integrating the learned models into its UCT simulator. We further discuss the influences of POMDP parameters in subsection 5.3.5.

Pursuit domain. As mentioned in [24], the pursuit domain requires cooperation among all teammates while remaining simple enough to evaluate approaches well. Hence, we conduct the experiments with it (see Figure 5.1). The pursuit domain is a game where four predators try to catch a prey in a 10×10 toroidal grid world. Each episode starts with the prey and predators in random locations and ends with predators surrounding the prey (see Figure 5.1b). The task consists of four sub-tasks, i.e. a predator occupies the left, right, top and bottom position of the prey. Predators and the prey can move in any of the four directions (top, bottom, left, right) or stay still. Each step of an episode starts from the prey move. The prey moves with 0.5 probability (and the remaining time it stays still) to one of the unoccupied neighbouring positions selected using the uniform distribution.

FIGURE 5.1: The pursuit domain

Next, predators move in random order. If a predator decides to move to a position that in meantime got occupied by others, it stays still [24]. We measure the performance by the average number of steps taken to capture the prey. Following the existing approach [24], we consider three types of predators:

- **Greedy (GR) predators** check the prey's neighbouring positions and move to the nearest unoccupied one. They only try to avoid the immediate obstacles (other agents).
- **Team-aware (TA) predators** also target one of the prey's neighbouring positions. However, in the mean time, they are willing to help the farthest predator to occupy the closest neighbouring position to the prey. That is, when TA predators block the way of teammates for occupying the closest neighbouring position, they will move away even their move can increase their distance to the prey. Henceforth, their planning considers the benefits of teammates.
- **Probabilistic-destinations (PD) predators** randomly choose a new position at every time step as long as this position decreases their distance to the prey. Therefore, PD predators may not target the neighbouring positions of the prey at the beginning.

Note that GR predators are fully selfish, i.e. they only care whether themselves can occupy one neighbouring position of the prey while ignoring the situation of teammates. In comparison, TA predators not only move towards the prey but also try to help teammates even that may impair their own benefits. Furthermore,

Name	Value	Name	Value	Name	Value
Simulation	1000	Depth	100	c	0.5
μ	0.95	η	1	r_1	30
r_2	4	TR_1	1	TR_2	1

TABLE 5.1: Experimental parameters

PD predators' behaviour has more randomness compared to GR and TA predators since they do not always target the neighbouring positions. We select these three types of predators because their behaviours are very different with one another. Using them, we can evaluate the influence of the quality of behaviour models.

Table 5.1 shows the parameters used in the experiments. In every step, we run at most 1000 UCT simulations with the maximum depth as 100. We set the decision time limit for each step as one second since the task is time sensitive.

5.3.1 Influence of the Acquired Behaviour Models

We examine the performance of ATSSIS with respect to PLASTIC and OPAT given two possible settings (we run each setting for 1000 episodes to ensure statistical significance). First, we give PLASTIC and OPAT the teammates' behaviour models that match perfectly the current teammates (see Figure 5.2a). As expected, when PLASTIC has the accurate teammates' models, it can predict their actions with high accuracy and thus, perform well. However, in real life, accurate behaviour models may not be available. Having accurate teammates' models is a strong assumption. Hence, in the next sections, we focus on more realistic scenarios where

FIGURE 5.2: Performance with acquired model

we do not have those models. For OPAT, even though it has the correct behaviour models, it performs worse than ATSSIS for GR and PD teammates. This is because given one state, OPAT needs to perform UCT simulation for each joint action (i.e. 5^4 joint actions in the pursuit domain) to estimate its utility and OPAT is not able to do enough simulations within the decision time. In the simulations with GR or PD teammates, we observe that even OPAT only runs one UCT simulation for each joint action, it takes about 0.5 second for 5^4 joint actions in total. OPAT performs well only for TA as it can perform more simulations given the time limit. This is because each episode is shorter (agents catch the prey with less number of steps as shown in Figure 5.2a) and hence, the time of each UCT simulation is shorter. Given these results, in the following experiments, we only use PLASTIC as the benchmark for ATSSIS.

Second, we use the models that fail to represent the current teammates, i.e. we use the TA model while the real teammates are GR. Figure 5.2b shows the number of steps needed to capture the prey and the number of *long episodes*, i.e. the episodes whose overall number of steps exceeds 250. We observe that the average numbers of steps for PLASTIC and OPAT are higher than for ATSSIS (31 and 47.7 respectively compared to 27.33). Additionally, PLASTIC has a very high number of long episodes. Note that when calculating the average, we only include episodes defined as not long as most of PLASTIC's long episodes never end. This is because in its UCT simulations, there exist situations where a teammate occupies one neighbouring position of the prey while the ad hoc agent is next to that teammate. Because that teammate follows the TA behaviour model in the simulation, it always makes way for the ad hoc agent to occupy the closest neighbouring position of the prey. After simulations, PLASTIC learns to wait for its teammates to move away in those situations. However, in the real cases, the GR predators never move out. In comparison, when ATSSIS observes that a teammate keeps staying in one neighbouring position of the prey for several time steps, it will infer that occupying that neighbouring position is the sub-task of that teammate. Then, ATSSIS will choose its sub-task from other neighbouring positions, and thus avoid being blocked by that teammate.

In summary, ATSSIS achieves tasks robustly without relying on teammates' behaviour models. In comparison, the existing methods are vulnerable to incorrect

models. This is because the UCT simulations that are dependent on teammates' models fail to correctly estimate the actions' utilities.

5.3.2 Teamwork with Learning Period

Next, we examine the scenarios where the ad hoc agent needs to learn from scratch teammates' behaviour models by observing their state-action pairs within a certain amount of steps. Since in real life, the agent cannot observe its teammates indefinitely, we are interested to compare ATISIS (denoted as "ATS" in Figure 5.3) with PLASTIC (denoted as "PLAS") that has different lengths of learning periods. We set the teammates' behaviour to be dynamic (e.g. GRPD teammates randomly select GR and PD behaviours at each time step) to imitate more complex behaviours. We average the results for each setting over 10 trials where each trial consists of 500 episodes. We do Wilcoxon signed-rank test as in [20] and observe that when the learning period is below 1600 steps for GRPD and 1200 steps for TAPD, ATISIS outperforms PLASTIC significantly. We measure also that each PLASTIC step takes around 1 second. Given that, PLASTIC would need to learn for roughly 1600 seconds to beat ATISIS. Even though PLASTIC can act during learning, it performs badly until it learns a good model. Learning for almost half an hour is not acceptable as we expect the ad hoc agent to start acting well shortly after joining the team.

FIGURE 5.3: Performance under different learning periods

FIGURE 5.4: Performance under transfer learning

5.3.3 The Effects of Transfer Learning

Here, we pitch AT SIS against PLASTIC that employs transfer learning. We use data from previous collaborations, i.e. the state-action pairs obtained while collaborating with some teammates (not necessarily the same as the current teammates). In Figure 5.4, we index data using the type of previous teammates denoted below “PLAS”. Each data set consists of the teammates’ state-action pairs observed within 5000 time steps. “NoSource” means not using any data of previous teammates. The learning period is 800 which is not enough to learn the accurate model of current teammates (see Figure 5.3). We evaluate whether transfer learning can help to learn the teammates’ model better with the data of previous teammates. Note that TA teammates are explicitly considerate, and thus they are the most different ones. Figure 5.4 indicates that if the previous teammates are much different from the current teammates (e.g. TA vs. GRPD or TA vs. TAPD), using their data compromises the learned model. Thus, transfer learning is only effective when the previous teammates are comparable to current ones.

5.3.4 Improving AT SIS with the Learned Model

Here, we show how to further improve the performance of AT SIS by giving its UCT simulator the learned model to sample the teammates’ actions (instead of assuming teammates are static). Having the model helps AT SIS to estimate better the best action to achieve the sub-task and the goal. The number below “ATS” in Figure 5.5 denotes the length of the learning period. We observe that the performance of AT SIS with the learned model improves (see Figure 5.5a). Note

FIGURE 5.5: Performance when equipping the learned model

that ATSSIS performs much faster than PLASTIC, i.e. 0.3 second vs. 1 second (see Figure 5.5b). This is because each simulation in ATSSIS takes shorter time as the termination condition “achieving the sub-task” in ATSSIS is easier to be met than the termination conditions of PLASTIC, i.e. “completing the task”. This is a critical advantage when the task is time sensitive. We also check the performance of PLASTIC by setting its decision time close to the one of ATSSIS, i.e. 0.3 second (denoted as PLAS* in Figure 5.5). In this case, PLASTIC performs much worse due to less UCT simulations.

5.3.5 The Effects of POMDP Parameters

In this subsection, we evaluate the influence of POMDP parameters on sub-task selection based on the teamwork performance. To solely measure POMDP parameters, we set the prey to stay still, give ATSSIS the accurate behaviour model and set TR_2 to 0. Then, the teamwork’s performance is determined by whether POMDP selects the sub-task properly. This depends on the setting of r_1 , r_2 , μ and η . We run 1000 episodes for each setting. We evaluate the performance, i.e. the average

r_1	10	15	20	25	30
PD (Avg)	17.93	16.15	13.89	13.12	13.36
PD (Long)	18	0	0	0	0
TA (Avg)	8.97	9.17	9.37	9.77	9.56
TA (Long)	189	155	108	120	112

TABLE 5.2: Performance under different r_1 when $r_2 = 4$

number of steps needed for prey capture (denoted as “Avg”) and the number of long episodes (denoted as “Long”), when predators are PD and TA respectively.

The ratio of r_1 to r_2 represents the trade-off between pursuing the unattended sub-task and the most suitable sub-task. Table 5.2 shows that the smaller the ratio, the worse the performance of AT SIS. This is because a smaller ratio makes the ad hoc agent focus on the sub-task it is most suitable for, while ignoring what others are doing. Moreover, when teaming up with TA agents, the number of long episodes is large even when the ratio is large. This is because both the ad hoc agent and teammates are considerate, i.e. they move away simultaneously to give way to others and then move back based on others’ changed state, which results in a deadlock. Notice that the deadlock rarely appears when the prey is moving.

The parameters μ and η represent to what extent the POMDP observations can affect the belief update. Table 5.3 shows that when μ becomes smaller, the effects are similar to the case where r_1 is low. This is because when μ is close to 0.5, the sub-task inference barely changes the belief about teammates’ sub-tasks. Thus, POMDP selects the ad hoc agent’s sub-task based mostly on its suitability for that sub-task. Further, for μ smaller than 0.5, if the ad hoc agent infers that a sub-task is being pursued, then the belief about the sub-task being pursued will be weaker (i.e. may result in a belief that this sub-task is unattended). Figure 5.6 shows that when $\mu = 0.95$, the change of η does not affect the performance in a significant way. It is the belief about teammates’ sub-tasks that mainly affects the

μ	0.95	0.85	0.75	0.65	0.55
PD (Avg)	13.36	13.39	14.88	20.38	27.05
PD (Long)	0	0	10	137	204
TA (Avg)	9.56	9.82	8.55	8.63	7.97
TA (Long)	112	94	149	239	302

TABLE 5.3: Performance under different μ when $\eta = 1$

FIGURE 5.6: Performance under different η

decision. When $\mu = 0.5$, POMDP can only rely on the belief about how suitable each sub-task is. A large η makes the ad hoc agent identify the suitable sub-task, focus on it and may block others. A smaller η makes the decision more random since the observations cannot affect the belief about which sub-task is suitable. When $\eta = 0.25$, the decision is totally random. Finally, when $\eta = 0$, the teamwork can never be achieved since the ad hoc agent will always consider the most suitable sub-task as unsuitable and not choose to do it. To enable and encourage the ad hoc agent to pursue unattended sub-task as well as make it care about the suitability for sub-tasks, we analytically assess that the best parameter values are a relatively high ratio of r_1 to r_2 , $\mu \geq 0.85$ and $\eta \geq 0.75$.

5.4 Summary

This chapter proposes AT SIS that achieves ad hoc teamwork without relying on teammates' behaviour models. In AT SIS, the ad hoc agent considers the teamwork as a set of sub-tasks and infers which sub-task each teammate is pursuing by observing the effect of teammates' actions. Then a POMDP model handles the uncertainty in the inference and selects the sub-task for the ad hoc agent. Finally, to achieve the chosen sub-task efficiently, the ad hoc agent uses UCT to search the best actions. Experiments demonstrate that AT SIS achieves the teamwork robustly without relying on teammates' behaviour models. This is a huge advantage for environments that do not provide the opportunity to obtain accurate teammates' models. AT SIS makes much faster decisions than state-of-the-art schemes, which is significant for time-sensitive tasks. Moreover, AT SIS can further improve its

performance by integrating the learned model. When it comes to the domains with a larger state space, we believe AT시스 will still perform better than the existing methods. In AT시스, the inference of sub-tasks and the pursuing of own sub-task are both single-agent MDP problems. This helps AT시스 to withstand better the increase of state space when compared to PLASTIC. We perform some tests under 20×20 grid world and observe that PLASTIC needs a long time for both learning and MCTS simulation since the search space (the joint state of multiple agents) is larger.

This chapter tackles ad hoc teamwork in simple domains. However, in more complex domains, we cannot assume that teammates' actions are fully observable. In this case, the ad hoc agent cannot learn teammates' behaviour models or infer teammates' sub-tasks based on their actions. Therefore, the ad hoc agent has to rely on the state changes of teammates. Note that the state space can be continuous in complex domains. Henceforth, in the next chapter, we use deep neural networks (that handle continuous input very well) to solve the ad hoc teamwork problem in complex domains.

Chapter 6

AATEAM: Achieving Ad Hoc Teamwork by Employing the Attention Mechanism¹

In Chapter 5, we formalise the ad hoc teamwork problem as finding the ad hoc agent's actions that maximise the expected utility. That is,

$$a^* = \operatorname{argmax}_{a_i} Q(s, a_i \vee \vec{a}_{-i}) = \operatorname{argmax}_{a_i} \sum_{s' \in S} P(s'|s, a_i \vee \vec{a}_{-i}) [R(s, a_i \vee \vec{a}_{-i}, s') + \gamma V^\pi(s')] \quad (6.1)$$

However, in complex domains, the action space is partially observable due to certain reasons. For instance, given a continuous action space, the ad hoc agent cannot know the exact value of the actions performed by its teammates. Also, two actions with different intentions can look similar in some situations. For example, in the robosoccer domain, shooting is indistinguishable to passing to the teammate that is close to the goal. Henceforth, the teammates' joint action $\vec{a}_{-i}$ is not available. This means it is infeasible to learn teammates' behaviour models or infer their sub-tasks by observing their state-action pairs. Therefore, the problem this chapter aims to address is as below:

Problem: how the ad hoc agent achieves ad hoc teamwork in the domains with a partially observable action space.

The most advanced approach, PLASTIC-Policy [24], understands teammates' behaviour based on state transitions. PLASTIC-Policy uses reinforcement learning

¹This chapter contains the work published in [56].

to learn a policy for working with each type of past teammates. Each policy takes a state as input and outputs an action for the ad hoc agent. When working with new teammates, every time PLASTIC-Policy receives a state transition, the ad hoc agent locates the most similar state transition with each type of past teammates. By doing so, PLASTIC-Policy updates the probability of new teammates being similar to each type of past teammates, and applies the policy for the most similar past teammates. However, the learned policies are not necessarily optimal for new teammates. Additionally, the selected policy only works well when new teammates behave similarly to the chosen teammate type. When new teammates' behaviour becomes far from the teammate type, the ad hoc agent needs to switch to another policy. Since PLASTIC-Policy switches policies by updating the probability based on state transitions, it may not be fast enough to adapt to new teammates' changing behaviour. Moreover, PLASTIC-Policy needs domain knowledge to define the state transition and the similarity of state transitions.

In this chapter, we propose the Achieving Ad hoc Teamwork by Employing the Attention Mechanism (AATEAM) algorithm that uses the attention-based neural networks to cope with new teammates' behaviour in real-time. We train one attention network per teammate type. The attention networks learn both to extract the temporal correlations from the sequence of states (i.e. contexts) and the mapping from a context to a probability distribution over the ad hoc agent's actions. Each attention network also learns to output a metric that measure the similarity between new teammates and the corresponding past teammates. Finally, we aggregate the probability distributions over actions by calculating the weighted sum of those probabilities and its corresponding similarity metric, and select the action with the highest aggregated probability. Since the attention networks measure the similarity in every step (i.e. every time the ad hoc agent obtains a new state), AATEAM can adapt to new teammates' changing behaviour in real-time. In the following sections, we first show the design and work flow of the attention-based neural network. Second, we discuss how to select the final action for the ad hoc agent given those probability distributions output by the attention-based neural networks. Third, we explain how to train the attention-based networks in order to achieve the above design purpose. Fourth, we compare the characteristics of our domain with those of NLP domain to explain the rationale behind our design. Fifth, we analyse the complexity of PLASTIC-Policy and AATEAM. Finally, we present

the experimental results under RoboCup 2D simulation domain to demonstrate the advantage of AATEAM over PLASTIC-Policy.

6.1 Ad Hoc Teamwork with Attention Mechanism

In this section, we explain the details of our method. First, we present the overall design of one attention-based neural network. Note that AATEAM contains multiple attention networks, one per past teammates' type. Second, we explain how AATEAM selects actions for the ad hoc agent. Third, we describe how we train AATEAM. Fourth, we compare the characteristics of our domain with those of NLP domain to explain the rationale behind our design. Finally, we analyse the complexity of PLASTIC-Policy and AATEAM.

6.1.1 The Design of the Attention Network in AATEAM

The overall architecture of our attention-based neural network is shown in Figure 6.1. The architecture consists of two parts. That is, the encoder and decoder respectively. The encoder extracts important features for ad hoc teamwork from the sequence of states. The decoder exploits the extracted features to output the probability distribution over actions for the ad hoc agent. All subscripts in Figure 6.1 refer to steps. Figure 6.2 shows the detailed design of the attention network and illustrates how to process states, hidden states, and actions in every step t .

6.1.1.1 The encoder.

The encoder consists of a linear layer and the gated recurrent unit (GRU)² [59] as shown in Figure 6.2. The initial hidden state of the encoder $\vec{h}_0$ is a vector of zeros $\vec{0}$. In every step t , the encoder takes the previous hidden state $\vec{h}_{t-1}$ and the state $\vec{s}_t$ as input. Let u denote the size of state vector, l denote the size of GRU's

²The GRU is a popular variant of the recurrent neural network (RNN). As an alternative RNN, we could use the long short-term memory (LSTM) [57]. However, GRU has comparable performance to LSTM while having less parameters, which makes it easier to train [58].

FIGURE 6.1: The attention network for ad hoc teamwork

hidden state and $f_{x \rightarrow y}(\cdot)$ denote a linear layer that transforms a vector of size x into another vector of size y . A linear layer $f_{u \rightarrow l}(\cdot)$ transforms the state vector $\vec{s}_t$ into an embedding vector of length l . It does so, by multiplying the input by the weight matrix. Then, GRU uses the embedding vector and $\vec{h}_{t-1}$ to output a hidden state $\vec{h}_t$ and an encoder-output $\vec{o}_t$, i.e. the encoding of $\vec{s}_t$. The $\vec{h}_t$ and $\vec{o}_t$ then become the input of the decoder.

6.1.1.2 The decoder.

The purpose of the decoder is to output a high-level action $a_t \in A$ for the ad hoc agent. Let $n = |A|$ denote the number of high-level actions. Next, we understand *an episode* as one instance of a task performance. We denote the signal of the start of an episode as $\langle SOE \rangle$ and set its index to $n + 1$. The initial hidden state of the decoder $\vec{d}_0$ is the encoder's hidden state in step 1. $a_0 = \langle SOE \rangle$ and its

FIGURE 6.2: The detailed design of the attention network

index is input into the decoder in step 1 to signal the start of a task. In every step t , the decoder takes as input: 1) the set of encoder-outputs until step t , i.e. $\{\vec{o}_i | i = 1, \dots, t\}$, 2) the previous hidden state $\vec{d}_{t-1}$, 3) the encoder's hidden state $\vec{h}_t$, and 4) the previous output action a_{t-1} . As shown in Figure 6.2, the decoder uses these inputs and the attention module to generate the inputs for its GRU. Then, the GRU outputs a hidden state $\vec{d}_t$ and a probability distribution over actions $\{P(a_t^k) | k = 1, \dots, n\}$ (we discuss how we select a_t from $\{P(a_t^k) | k = 1, \dots, n\}$ in the latter subsection).

The attention module determines the weight of each $\vec{o}_i$ based on $\{\vec{o}_i | i = 1, \dots, t\}$, $\vec{d}_{t-1}$, and $\vec{h}_t$ (we present the details of how to compute the weights in the following subsection). Given the attention weight set $\{w_{t,i} | i = 1, \dots, t\}$, the decoder produces the context vector $\vec{c}_t$ in step t as

$$\vec{c}_t = \sum_{i=1}^t w_{t,i} \cdot \vec{o}_i \quad (6.2)$$

Next, the decoder uses an embedding layer (that maps an action index to a vector of length l) and dropout layer (that reduces overfitting [60]) to generate the embedding of a_{t-1} , i.e. $Embed(a_{t-1})$. To produce the input in_t of GRU in step t , the concatenation (denoted by “||”) of $Embed(a_{t-1})$ and $\vec{c}_t$ goes through a linear layer $f_{2l \rightarrow l}(\cdot)$ and Relu layer. The Relu layer applies the rectified linear unit function to each element of the input vector:

$$Relu(x) = \max(0, x) \quad (6.3)$$

We use the Relu function because, if the neuron gets activated, its gradient always equals to 1. This helps to train the network faster. Given in_t and $\vec{d}_{t-1}$, GRU outputs the decoder's hidden state $\vec{d}_t$ in step t and an action seed vector $\vec{a}_t$ of length l . To produce the probability distribution over actions in step t , i.e. $\{P(a_t^k) | k = 1, \dots, n\}$, we first use a linear layer $f_{l \rightarrow n}(\cdot)$ that converts $\vec{a}_t$ into a vector $\{\tilde{a}_t^k | k = 1, \dots, n\}$. Second, we transform $\{\tilde{a}_t^k\}$ using a softmax layer. That is,

$$P(a_t^k) = \frac{e^{\tilde{a}_t^k}}{\sum_{m=1}^n e^{\tilde{a}_t^m}} \quad (6.4)$$

where $P(a_t^k)$ represents the probability of taking action a^k in step t .

Attention Mechanism. The attention module is shown in Figure 6.2. In every step t , the attention module takes the following input: 1) the set of encoder's outputs $\{\vec{o}_i | i \leq t\}$; Each $\vec{o}_i$ naturally provides the information for the weights of $\{\vec{o}_i\}$; 2) The encoder's hidden state $\vec{h}_t$; The hidden state $\vec{h}_t$ contains the information of states until step t and thus, it helps to produce a new context; and 3) the decoder's previous hidden state $\vec{d}_{t-1}$; Note that $\vec{d}_{t-1}$ comes from a_{t-2} and $\vec{d}_{t-2}$. It also results from a_{t-3} , a_{t-2} and $\vec{d}_{t-3}$. Following this logic, $\vec{d}_{t-1}$ reflects the effects of the sequence of actions $\{a_i | i < t\}$ on $\vec{d}_0$, that is, on the information of initial state $\vec{h}_1$. The effects of previous actions on states are useful for generating $w_{t,i}$ since a_t is correlated with $w_{t,i}$.

Given the inputs, a linear layer $f_{2l \rightarrow l}(\cdot)$ transforms the concatenation of $\vec{h}_t$ and $\vec{d}_{t-1}$ into a vector $\vec{v}_{hd} = f_{2l \rightarrow l}(\vec{h}_t || \vec{d}_{t-1})$. Then, each $\vec{o}_i$ in $\{\vec{o}_i | i \leq t\}$ goes through a linear layer to become a vector $\vec{v}_{o_i} = f_{l \rightarrow l}(\vec{o}_i)$. The concatenation of $\vec{v}_{o_i}$ and $\vec{v}_{hd}$ further goes through a Tanh layer (following [45]) that applies the $\tanh$ function to each element of the input vector:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (6.5)$$

Next, the output of Tanh layer goes through a linear layer $f_{2l \rightarrow l}(\cdot)$ to output the *weight seed* $\bar{w}_{t,i}$ for $\vec{o}_i$ in step t . In summary, the weight seed $\bar{w}_{t,i}$ is computed as:

$$\bar{w}_{t,i} = f_{2l \rightarrow l}(\text{Tanh}(f_{2l \rightarrow l}(\vec{h}_t || \vec{d}_{t-1}) || f_{l \rightarrow l}(\vec{o}_i))) \quad (6.6)$$

Finally, a softmax layer takes the set of weight seeds $\{\bar{w}_{t,i} | i \leq t\}$ as input and outputs the weight $w_{t,i}$ of each $\vec{o}_i$ in step t . That is,

$$w_{t,i} = \frac{e^{\bar{w}_{t,i}}}{\sum_{b=1}^t e^{\bar{w}_{t,b}}} \quad (6.7)$$

6.1.2 Selecting the Action in AATEAM

In the previous subsection, we described the design of one attention network. Each attention network outputs a probability distribution over actions. In this subsection, we discuss how to select the final action for the ad hoc agent given those probability distributions.

Formally, we denote the set of attention networks in AATEAM as $\mathcal{T}$. In every step t , we input the current state $\vec{s}_t$ and previous action a_{t-1} into each attention network. Note that each network has its own $\{\vec{o}_i\}$, $\vec{h}_{t-1}$ and $\vec{d}_{t-1}$ (computed by that network). Based on the input, each network j in $\mathcal{T}$ outputs a probability distribution over actions $\{P(a_t^k)\}_j$ as well as its $\vec{h}_t$ and $\vec{d}_t$. Note that the encoder's hidden state $\vec{h}_t$ results from $\{\vec{s}_i | i \leq t\}$. Therefore, $\vec{h}_t$ contains the state information until the step t . Additionally, $\vec{d}_t$ reflects the effects of $\{a_i | i < t\}$ on $\vec{s}_1$ (as discussed in the Attention Mechanism part, i.e. Chapter 6.1.1.2). Henceforth, we train each attention network such that $\vec{d}_t$ is the prediction of the state in step t based on previous states and actions (we discuss the details of the training in the next subsection). If $\vec{d}_t$ is close to $\vec{h}_t$, it means that the attention network's action affects the state as intended. Because the state also results from the teammates' behaviour, the distance between $\vec{d}_t$ and $\vec{h}_t$, i.e. $\text{dist}_j(\vec{h}_t, \vec{d}_t)$, implies the extent to which the attention network matches the teammates' behaviour. The more a network matches the teammates' behaviour, the more influential it should be to the action's selection. Henceforth, we can use the distance to weigh the probability distributions (one per attention network) and select the action with the highest weighted probability. We compute the weight of each network j as:

$$v_j = \frac{e^{-\text{dist}_j(\vec{h}_t, \vec{d}_t) \cdot \delta}}{\sum_{z=1}^{|\mathcal{T}|} e^{-\text{dist}_z(\vec{h}_t, \vec{d}_t) \cdot \delta}} \quad (6.8)$$

where δ is a hyper-parameter and compute the weighted probability distribution over actions as:

$$\{\bar{P}(a_t^k)\} = \sum_{j=1}^{|\mathcal{T}|} \{P(a_t^k)\}_j \cdot v_j \quad (6.9)$$

Finally, we select the action with the highest probability in $\{\bar{P}(a_t^k)\}$ as the action for the ad hoc agent in step t .

6.1.3 Training AATEAM

In this subsection, we discuss how we train one attention network j (for each teammate type j). First, we use reinforcement learning to learn a policy π_j for cooperating with past teammate type j . After that, we make the ad hoc agent use π_j to work with the teammate type j for a number of episodes. During the teamwork, we collect the state-action pairs as training data. Let us denote $\hat{a}_t$

as the action that the ad hoc agent took in step t based on the learned policy π_j . The state-action pair $(\vec{s}_t, \hat{a}_t)$ is the training instance of step t , where $\hat{a}_t$ is the label for the training, i.e. we compare the network's output $\{P(a_t^k)\}$ given $\vec{s}_t$ with $\hat{a}_t$. A training sample is a sequence of state-action pairs, formally $\{(\vec{s}_t, \hat{a}_t)\}_j$. For each state-action pair $(\vec{s}_t, \hat{a}_t)$ in a training sample, the attention network takes $\vec{s}_t$ as input and returns a probability vector over actions $\{P(a_t^k) | k = 1, \dots, n\}$. The probability that the network selects the action $\hat{a}_t$ is $P(a_t^k = \hat{a}_t)$. We calculate the loss of an action using the negative log-likelihood:

$$loss_a = -\log(P(a_t^k = \hat{a}_t)) \quad (6.10)$$

Notice that the higher the probability that the attention network selects $\hat{a}_t$, the smaller the loss of action $loss_a$.

Since $\{\hat{a}_t\}$ output by π_j fits the past teammate type j 's behaviour, i.e. affects the state as intended, the attention network should learn to reduce the distance between $\vec{d}_t$ and $\vec{h}_t$. By doing so, we ensure that $\vec{d}_t$ makes good prediction of s_t given the teammate type j 's behaviour. Hence, we compute the loss of distance as the square of $L2$ -norm:

$$loss_d = \|\vec{d}_t - \vec{h}_t\|_2^2 \quad (6.11)$$

Then, the overall loss is an additive function of the above loss measures:

$$loss = loss_a + \beta loss_d \quad (6.12)$$

where β is the constant that adjusts the weight of $loss_d$.

We use the Adam optimizer [61] to update the network parameters to minimise the computed loss. Moreover, during the training, we apply the teacher forcing strategy [62], i.e. use $\hat{a}_t$ instead of a_t as the input to the decoder in the step $t + 1$. This is because the actual $\vec{s}_{t+1}$ results from $\hat{a}_t$ instead of a_t .

6.1.4 Comparison to the NLP domain.

In the teamwork domain, the attention network needs to handle the sequence of inputs differently from the attention networks in the NLP domain (e.g. Figure 3.6). This is because each domain has different information available. First, in the NLP

domain, the encoder can encode the whole sentence before inputting anything into the decoder. In our domain, we do not know the states from the future. Hence, we can only use the state information until the current step. Additionally, the NLP decoder uses the last hidden state of the encoder as its initial hidden state such that the decoding is based on the whole sentence. In our domain, the decoder can only use $\vec{h}_1$ as its initial hidden state (since there is only $\vec{s}_1$ in the beginning of a task). Second, the attention network in NLP needs to learn when to end a sentence (i.e. output $\langle EOS \rangle$). However, in our domain, the end of episode is an external signal and thus, our network does not need to output it.

6.1.5 Complexity Analysis

In this subsection, we analyse the complexity of PLASTIC-Policy and AATEAM.

For PLASTIC-Policy, to update the probability of current teammates being similar to each past teammate type, it needs to find the most similar past experience with each past teammate type given a state transition. To do so, PLASTIC-Policy compares the given state with every past state (high-dimensional vector) in a large set of past experiences, which is very costly. In comparison, given a new state, the attention networks in AATEAM only perform a few matrix operations, which is computationally cheap. In the simulation with CPU type being Intel Xeon E5-1630v4, AATEAM takes on average 334 milliseconds (using CPU for computation) for one episode while PLASTIC-Policy takes 5898 milliseconds. In addition, AATEAM uses 265 Megabytes memory for loading attention networks while PLASTIC-Policy uses 724 Megabytes memory for loading past experiences.

Although AATEAM does have an additional cost of training attention networks. It is an offline cost, i.e. it occurs before the game. The online cost (occurred when playing with new teammates) is much lower for AATEAM with respect to PLASTIC-Policy.

6.2 Experiments

In this section, we present the experiments we have done to demonstrate the effectiveness of AATEAM. First, we introduce the experiment domain. Second, we

present how we collect the data. Finally, we pitch AATEAM against PLASTIC-Policy to show the advantage of our approach. In detail, we show the results of both algorithm when working with first, known and second, unknown teammates. Our results indicate that when working with both known and unknown teammates, in most cases our algorithm outperforms PLASTIC-Policy significantly.

6.2.1 Half Field Offence Domain

Following [24], we use a limited version of the RoboCup 2D simulation domain, i.e. half field offense (HFO) domain [63]. HFO retains the complexity of the RoboCup 2D simulation domain in terms of the state/action space and the uncertainty of actions' effects. The main difference between both settings is that the game is played within a half of the field and the ad hoc agent always replaces one of the agents in the offence team. This simpler setting with single objective allows the researchers to focus on developing algorithms that tackle the environmental complexity.

In this chapter, we use the HFO version with two offensive agents that try to score a goal into opposite team's goal. The opposite team has two defenders including the goalie. For the defensive agents, we adopt the agent2D behaviour provided by [64]. At the beginning of each episode, the ball is put in a random position close to the midline of full field. The initial position of the goalie is in the goal centre. The remaining agents start at random position within a given range. The ranges of offensive and defensive agents ensure that the offensive agents are initialised to be closer to the ball than the defender. In every step, each agent can observe the position of the ball as well as the others' positions. An episode ends in one of the following cases: 1) the offensive team scores a goal, 2) the ball leaves the field, 3) the defensive team captures the ball, or 4) the game lasts for 500 simulation steps. The success episodes are the episodes in which the offensive team scores a goal.

State. In every simulation step, the ad hoc agent gets the following state features:

- the ad hoc agent's (x, y) position and the angle that the agent is facing
- the ball's (x, y) position

FIGURE 6.3: The illustration of opening angles [24]

- the distance and the angle from the ad hoc agent to the centre of the goal
- the ad hoc agent's largest opening angle to the goal, shown as θ_g in Figure 6.3
- the distance from the ad hoc agent to the closest defender
- the teammate's largest opening angle to the goal
- the distance from the teammate to the closest defender
- the opening angle for passing to the teammate, shown as θ_p in Figure 6.3
- the teammate's (x, y) position
- each defender's (x, y) position

Given our setting, i.e. one teammate and two opponents, the state is a vector of 18 elements. Note that we sort the defenders' position in the state vector based on their uniform number while the original simulator sorts those positions based on the distance between each defender and the ad hoc agent. This modification is to ensure that each element about a defender's position always corresponds to the same defender.

Action. The high-level actions that the ad hoc agent can perform are: 1) *Dribble*, 2) *Pass*, 3) *Shoot*, and 4) *Move*. In every step, the simulator informs the ad hoc agent if it is close enough to be able to kick the ball. If so, the ad hoc agent can select an action from $\{Dribble, Pass, Shoot\}$. When the ad hoc agent is not close enough to the ball, it can only perform *Move*. Both learned policies and attention networks select high-level actions for the ad hoc agent when the agent can kick the ball. Given a high-level action, the simulator maps it to a continuous action to perform the simulation.

6.2.1.1 Collecting data for AATEAM.

We use the State-Action-Reward-State-Action (SARSA) algorithm [63] to learn a policy π_j for each past teammate type j . The ad hoc agent uses π_j to work with teammate type j for 200000 episodes. Within each episode, we collect the state-action pairs $\{(\vec{s}_t, \hat{a}_t)\}$ every time the ad hoc agent obtains the control of the ball and until it loses the ball. This sequence of $\{(\vec{s}_t, \hat{a}_t)\}$ is one training sample. We only store the state-action pairs from the success episodes.

Note that due to the continuous state space, the states of adjacent steps are very similar. Our initial experiments show that if we collect the state-action pairs in every step, the smoothness among the sequence of states impairs the attention network’s performance. Henceforth, we set a threshold tr to detect the change of state. In every step, we compare each element of current state (denoted as z') with the corresponding element of previous state (denoted as z). When at least one state element meets the condition $\frac{|z-z'|}{|\min(z, z')|} > tr$, we collect the current state-action pair. Similarly, when using the trained networks to play, we select a new action when at least one state element meets the above condition. Otherwise, we continue the current action. Moreover, we remove the training samples that contain only one state-action pair. This is because attention networks learn to extract the temporal correlations of states. The samples with only one state have no temporal correlation inside and thus, will hinder the learning performance.

6.2.1.2 Collecting data for PLASTIC-Policy.

We collect state transitions (200000 episodes for each type) since PLASTIC-Policy uses them to update its probability distribution over the past teammate types. According to [24], a state transition $(\vec{s}, a, \vec{s}')$ means that either the possession of the ball changes from one agent to another agent or the episode ends when the ad hoc agent performs the action a . In the experiments, $\vec{s}$ is the state of the step when the ad hoc agent can kick the ball. $\vec{s}'$ is the state of the step either when another agent can kick the ball (this could result from intercepting, passing, or shooting) or the episode ends. Note that we cannot determine state transitions when the ad hoc agent cannot kick the ball because the simulator does not have a signal that indicates which specific agent holds the ball.

Teammate types. In this chapter, we use the externally-created teammates. The teammates are not programmed to work with the ad hoc agent. We run the binary file released from the 2013 RoboCup 2D simulation league competition [65]. From these external agents, we use five types as past teammate types, i.e. aut, axiom, agent2D, gliders, and yushan. Additionally, we use five types as new teammate types, i.e. yunlu, utaustin, legend, ubc, and warthog.

6.2.2 Experimental Parameters

We sum up the experimental parameters in Table 6.1. The number of hidden state layers for GRU is 5. We use the last layer of hidden states to compute the $dist_j(\vec{h}_t, \vec{d}_t)$ and $loss_d$. The maximum length of a state sequence that attention networks process is 60. When the state sequence’s length exceeds 60, the networks drop the states from the beginning. The rate of dropout is 0.1. When training the attention networks, the default learning rate is 0.01 and the batch size is 32. In the experiments, we use AATEAM and PLASTIC-Policy to play with each teammate type for 10 trials where each trial contains 1000 episodes. We set the decision time limit for each step as 100 ms following [24].

We evaluate the performance based on the scoring rate, i.e. how many episodes the offence team wins in every 1000 episodes. The error bars in Figure 6.4 and Figure 6.5 below are the standard deviations of scoring rates. Note that the results are discrete points. We add lines between them to improve clarity. We apply the binomial test to test the significance of our results (following [24]). We observe that the difference between AATEAM and PLASTIC-Policy is significant (95% confidence) except for the axiom and ubc types.

Name	Value	Name	Value
β	0.1	tr	0.2
u	18	l	256
δ	10	GRU Layer Number	5
Max length	60	Dropout rate	0.1

TABLE 6.1: Experimental parameters

6.2.3 Playing with Past Teammates

In this subsection, we make the ad hoc agent play with the teammates it encountered before, i.e. past teammates. In detail, it plays with five past teammate types using first, AATEAM and second, PLASTIC-Policy. Also, we have one baseline and one upper bound here. The baseline is the performance of the original offence team, i.e. the team without the ad hoc agent. The upper bound is the performance of the ad hoc agent playing with a given teammate type following the learned policy for that teammate. Since each policy is learned only for the corresponding past teammate type, we expect the performance with the learned policies to be higher than any other approach. The results are shown in Figure 6.4.

In Figure 6.4, we see that both AATEAM and PLASTIC-Policy are better than the baseline. This is because the reinforcement learning algorithms managed to learn good policies for past teammate types (both PLASTIC-Policy and during the training of AATEAM). On top of that, the attention networks successfully learned to fit the behaviour of past teammates. Hence, both algorithms exhibit more intelligent behaviours.

When it comes to the upper bound, we can see that AATEAM’s performance is closer to the upper bound than PLASTIC-Policy in most of the cases. Also, AATEAM outperforms PLASTIC-Policy significantly. The reason is as follows. PLASTIC-Policy finds the corresponding policy for the known teammate type by evaluating the similarity between past experiences and current ones. However, it may not be able to identify the most similar past experience due to the decision

FIGURE 6.4: Performance with known teammates

time limit (i.e. 100 ms). Thus, the similarity evaluation may cause it to switch between different learned policies. This degrades its performance. In comparison, AATEAM extracts the context in every step and maps the context to the best action. Note that the extraction and mapping using trained attention networks are very efficient. Therefore, AATEAM can take advantage of the knowledge from the learned policies in a more flexible way.

In detail, the ad hoc agent uses AATEAM to learn to cope with several different teammate types. Next, it plays with teammates of one of the known types. Each attention network outputs a probability distribution over actions. AATEAM aggregates the probabilities from all attention networks. The aggregation of the probability distributions enables a better usage of the previous knowledge (as we base our decision on more knowledge than the knowledge about only one past teammate type). Note that the originally learned policies (upper bounds) come from reinforcement learning and they may not be optimal. Given a teammate type, some policies for another types may suggest a better action than the policy for the given type. Therefore, AATEAM can perform better than the upper bounds as it uses more information from neural networks. In summary, we observe that AATEAM can easily adapt to the teammates' behaviour.

6.2.4 Playing with New Teammates

In this subsection, the ad hoc agent uses AATEAM and PLASTIC-Policy to work with the new teammates, i.e. the teammates it has never encountered before. Naturally, we have one baseline. That is, the baseline where an original team plays without the ad hoc agent. We cannot use the upper bound as the ad hoc agent did not learn the policies to work with the new teammates beforehand. We show the results in Figure 6.5.

In Figure 6.5, we see that both AATEAM and PLASTIC-Policy outperform the baseline as both algorithms behave more intelligently than the replaced agent. Moreover, in most cases AATEAM still significantly outperforms PLASTIC-Policy. The reason is that PLASTIC-Policy switches between policies based on the current behaviour of new teammates. Since switching takes time, PLASTIC-Policy cannot adapt to teammates' behaviours quickly. In addition, the performance of both AATEAM and PLASTIC-Policy is good except when the ad hoc agent plays with

FIGURE 6.5: Performance with unknown teammates

the ubc type. The reason is that the ubc agent does not often pass the ball to its teammate. Therefore, even though AATEAM and PLASTIC-Policy are smarter than its original teammate, they cannot help much. In summary, the experiments of playing with unknown teammate types demonstrate that AATEAM can adapt to different teammates' behaviours better than PLASTIC-Policy. This is crucial for ad hoc teamwork in complex domains.

6.2.5 The Importance of Attention Mechanism

In this subsection, we show the importance of the attention mechanism in AATEAM. Besides the architecture shown in Figure 6.2, we also try to train the neural network without the attention mechanism, i.e. each o_i has the same weight. Additionally, we try to include the training samples of length 1 during the training. The loss of actions in different cases are shown in Figure 6.6. The results demonstrate that we can get a good network only when we adopt the attention mechanism and remove the training samples of length 1 (to prevent the attention module from being influenced). Note that the loss of actions directly affects the performance of neural network. Henceforth, the results in Figure 6.6 indicate that the attention mechanism is an essential part in AATEAM.

FIGURE 6.6: The loss of different training cases

6.3 Summary

This chapter proposes a novel approach called AATEAM to solve the ad hoc teamwork problem in complex domains. To the best of our knowledge, this is the first attempt to solve the ad hoc teamwork problem using attention-based recurrent neural networks. In detail, AATEAM contains a set of neural networks, each of them tackles the behaviour of one different past teammate type. Using the probability distributions output by each neural network, AATEAM selects the best action for the ad hoc agent in real-time. We pitched our algorithm against PLASTIC-Policy that is the only scheme proposed for the ad hoc teamwork problem in complex domains. Our results indicate that when working with both known and unknown teammates, in most cases our algorithm outperforms PLASTIC-Policy. This demonstrates that AATEAM can adapt to new teammates' changing behaviour faster.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we tackle the problems that arise due to the uncertainties in agent cooperation. Specifically, the first problem is how to determine whether we can use the potentially untrustworthy information shared by other agents for planning in a timely manner. The second problem we address is how an agent can collaborate with unknown teammates to achieve a shared goal without relying on communication or shared strategies. We refer to the second problem as ad hoc teamwork.

In Chapter 4, we propose a POMDP model to make efficient decisions about using the shared information of unobservable system states. In detail, the POMDP model determines when the agent is confident enough to take actions based on the shared information. To the best of our knowledge, the existing trust management schemes only focus on the accuracies of shared information. The states of this POMDP consist of the unobservable system state and the trustworthiness of the agent that shares the information. Upon receiving a message from a neighbouring agent, the state information reported by the agent and its trustworthiness are regarded as the observation of the POMDP. Based on the belief updated by the received observation, the POMDP model optimally queries information about the uncertain state from neighbouring agents while taking into account their malicious behaviours. Moreover, this work also considers that in real life the behaviours of neighbouring agents are unknown. Thus, the observation function that represents the behaviour of neighbours needs to be learned. We propose a learning algorithm

that can handle the dynamic scenario where malicious agents change their behaviour from time to time. We conduct extensive experiments under the VANETs domain. The results indicate that our POMDP model better balances the trade-off between the decision quality and the cost/delay incurred during the information collection when compared to the heuristic threshold-based schemes. Further, the experiments under the dynamic scenario verify that the learning algorithm can learn the observation function efficiently while still handling the behaviour change of neighbouring agents effectively.

In Chapter 5, we focus on ad hoc teamwork in simple domains. In the simple domains, the ad hoc agent can fully observe the actions of its teammates. Henceforth, it can understand teammates' behaviour based on their actions. The existing approaches learn teammates' behaviour models. Then, they use the learned behaviour models to select the actions for the ad hoc agent. However, in many cases the models of past teammates do not entirely match the behaviour of new teammates. To cope with those inaccuracies, we propose ATISIS that achieves ad hoc teamwork in simple domains without relying on behaviour models. Specifically, the ad hoc agent considers the task as a set of sub-tasks. Then, the ad hoc agent infers which sub-task each teammate is pursuing by checking whether the observed teammate's action brings the most utility to the sub-task. Since the inference has uncertainty, we use a POMDP model to select the sub-task for the ad hoc agent. The state of this POMDP model represents which sub-task each teammate is pursuing and how suitable the ad hoc agent is for each sub-task. By observing teammates' actions, the ad hoc agent evaluates the Q-value of their actions for each sub-task. Also, given current system state, the ad hoc agent evaluates how suitable it is for each sub-task by computing the expected utility each team member would obtain from pursuing the sub-task. These two evaluations are regarded as the observation to the POMDP model. They adjust the belief on the POMDP state that influences the sub-task selection. Last, to achieve the chosen sub-task efficiently, the ad hoc agent adopts UCT to search for the MDP actions. We evaluate the performance of ATISIS in the pursuit domain. Experiments demonstrate that ATISIS achieves the teamwork robustly without relying on teammates' behaviour models. This is a huge advantage for environments that do not provide the opportunity to obtain accurate teammates' models. ATISIS makes much faster decisions than state-of-the-art schemes which is significant for time-sensitive tasks. Moreover, ATISIS can further improve its performance by integrating the learned model.

In Chapter 6, we continue to work on the ad hoc teamwork problem but in complex domains. There, teammates' actions are only partially observable. In this case, the ad hoc agent cannot use the teammates' actions to learn their behaviour models or infer their sub-tasks. Therefore, the ad hoc agent can only understand the uncertain teammates' behaviour based on their state changes. While the ad hoc agent can take advantage of the knowledge obtained when working with previous teammates, it may not adapt fast enough to new teammates' behaviour. We propose AATEAM that adapts to new teammates' behaviour in real-time based on previous knowledge. Specifically, we first learn a policy for working with each past teammate type. Second, we let the ad hoc agent use the learned policy to work with the corresponding past teammate type. During this process, we collect the state-action pairs as training data. We use the collected data to train one attention-based neural network for each past teammate type. The attention network takes a sequence of states as input and uses the attention mechanism to extract the context. Further, the network learns to map the extracted context to a probability distribution over the ad hoc agent's action. Based on the training, the attention network can handle the behaviour of the corresponding past teammates. Additionally, the attention network learns to measure the similarity between the past and new teammates' behaviours in every step. The more similar the behaviours, the more important the corresponding probability distribution over actions. The ad hoc agent aggregates networks' probability distributions over actions by calculating the weighted sum of those probabilities and its importance. Finally, the ad hoc agent outputs an action with the highest aggregated probability. We conduct extensive experiments in the RoboCup 2D simulation domain. Experimental results indicate that when working with both known and unknown teammates, in most cases our algorithm outperforms PLASTIC-Policy. This demonstrates that AATEAM can adapt to new teammates' changing behaviour faster than PLASTIC-Policy. We also show the training results under different training settings to demonstrate the importance of using the attention mechanism. The results verify that the attention mechanism is an essential component for the success of our approach.

7.2 Future Work

• **Information sharing in multi-agent systems.** Future research can be conducted along two directions. The first direction is to evaluate the performance of our POMDP model with other kinds of sophisticated malicious behaviours. For example, malicious agents can change their behaviours more frequently. Also, malicious agents could choose to share false information only when they identify a certain number of malicious agents from their neighbouring agents. The literature that computes the trustworthiness of information rarely studies sophisticated malicious behaviours. Thus, it is worth applying the POMDP model to those scenarios. The second direction is to explore more complex reward functions in the POMDP model. Specifically, the current reward function in our model takes a system state and an action as input and outputs a reward. This implies that the agent takes actions based on only one system state. However, in reality, the agent's action may depend on multiple system states. To cope with this issue, we could extend the reward function. That is, the function takes multiple states as input. The input states may also have different importances in determining the reward. Note that this extension also requires us to enlarge the state space.

• **Ad hoc teamwork in simple domains.** We identify two future directions. The first direction is to propose learning algorithms to learn the transition function (i.e. the probability of teammates changing their sub-task) and observation function (i.e. to what extent the policy used for Q-value computation matches the behaviour of teammates) for the POMDP model. To learn those two functions, the most important thing is to know the sub-task history, i.e. what sub-tasks teammates pursued during performing the task across multiple episodes. As long as the sub-task history is obtained, the functions can be learned in principle by Bayesian inference (as in Chapter 3.2). Therefore, the learning algorithms should focus on the methods for obtaining the sub-task history. They may get the history by certain estimation methods (e.g. maximum likelihood estimation). The second direction is to enable ATISIS to identify sub-tasks automatically. Note that ATISIS needs the domain knowledge to consider the task as a set of sub-tasks. For the domains where the goal is too abstract to define sub-tasks, ATISIS cannot work. One idea is to use reward shaping [66] to automatically identify the intermediate goal states that are specific enough to define sub-tasks.

• **Ad hoc teamwork in complex domains.** Our approach has two limitations. The first limitation of AATEAM is that it needs plenty of data to train the attention networks. It is important to investigate how to improve the data efficiency and speed up the learning process. Two potential methods that we could use are transfer learning [67] and generative adversarial imitation learning (GAIL) [68]. Transfer learning can transfer the learned knowledge from one domain to other domains. GAIL can learn to perform novel tasks that are not included in the training settings. Therefore, it is interesting to study how to use them to train the attention networks for different tasks. The second limitation is that it is difficult to know if the architecture of our neural networks is the most appropriate one. For the second limitation, we plan to test AATEAM for more unknown teammate types to show the robustness of our method. Additionally, the neural architecture search (NAS) algorithm [69] that searches for the best neural network architecture could be adopted to find better architectures for our attention networks. Furthermore, it is desirable to test the same idea for even more complex domains (with larger action and state space).

List of Author's Publications

Conference Proceedings

- Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages XX-XX, 2020.
- Shuo Chen, Ewa Andrejczuk, Athirai A. Irissappane, and Jie Zhang. ATISIS: Achieving the Ad hoc Teamwork by Sub-task Inference and Selection. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 172-179, 2019.
- Shuo Chen, Athirai A. Irissappane, and Jie Zhang. POMDP-Based Decision Making for Fast Event Handling in VANETs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4646-4653, 2018.
- Shuo Chen, Rongxing Lu, and Jie Zhang. A Flexible Privacy-preserving Framework for Singular Value Decomposition under Internet of Things Environment. In *Proceedings of the IFIP International Conference on Trust Management*, pages 21-37, Springer, 2017.

Bibliography

- [1] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. [xxi](#), [27](#), [28](#)
- [2] Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. [xxi](#), [30](#)
- [3] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009. [1](#)
- [4] Partha Sarathi Dutta, Nicholas R. Jennings, and Luc Moreau. Cooperative information sharing to improve distributed learning in multi-agent systems. *Journal of Artificial Intelligence Research*, 24:407–463, 2005. [2](#)
- [5] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997. [3](#)
- [6] Barbara J. Grosz, Luke Hunsberger, and Sarit Kraus. Planning and acting together. *AI Magazine*, 20(4):23, 1999. [3](#)
- [7] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1504–1509, 2010. [3](#), [6](#), [55](#)
- [8] Maxim Raya, Panagiotis Papadimitratos, Virgil D. Gligor, and J-P Hubaux. On data-centric trust establishment in ephemeral ad hoc networks. In *Proceedings of the IEEE International Conference on Computer Communications*, pages 1238–1246, 2008. [4](#), [14](#), [48](#)
- [9] Ben-Jye Chang and Szu-Liang Kuo. Markov chain trust model for trust-value analysis and key management in distributed multicast MANETs. *IEEE Transactions on Vehicular Technology*, 58(4):1846–1863, 2009. [14](#)
- [10] Qin Li, Amizah Malip, Keith M. Martin, Siaw-Lynn Ng, and Jie Zhang. A reputation-based announcement scheme for VANETs. *IEEE Transactions on Vehicular Technology*, 61(9):4095–4108, 2012. [15](#), [37](#), [48](#)

- [11] Zhexiong Wei, Helen Tang, F. Richard Yu, Maoyu Wang, and Peter Mason. Security enhancements for mobile ad hoc networks with trust management using uncertain reasoning. *IEEE Transactions on Vehicular Technology*, 63(9):4647–4658, 2014. [14](#)
- [12] Wenjia Li and Houbing Song. Art: An attack-resistant trust management scheme for securing vehicular ad hoc networks. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):960–969, 2016. [4](#), [14](#)
- [13] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. [5](#)
- [14] Noa Agmon and Peter Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 341–348, 2012. [6](#), [7](#), [16](#), [18](#)
- [15] Doran Chakraborty and Peter Stone. Cooperating with a Markovian ad hoc teammate. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1085–1092, 2013. [6](#), [16](#)
- [16] Noa Agmon, Samuel Barrett, and Peter Stone. Modeling uncertainty in leading ad hoc teams. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 397–404, 2014. [6](#), [16](#)
- [17] Katie Long Genter, Noa Agmon, and Peter Stone. Role-based ad hoc teamwork. In *Proceedings of the AAAI Conference on Plan, Activity, and Intent Recognition*, pages 17–24, 2011. [6](#), [16](#)
- [18] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Online planning for ad hoc autonomous agent teams. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 439–445, 2011. [6](#), [17](#), [63](#)
- [19] Stefano V. Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1155–1156, 2013. [17](#), [18](#)
- [20] Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenfeld. Teamwork with limited knowledge of teammates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 102–108, 2013. [6](#), [9](#), [18](#), [63](#), [68](#)
- [21] Francisco S. Melo and Alberto Sardinha. Ad hoc teamwork by learning teammates’ task. *Autonomous Agents and Multi-Agent Systems*, 30(2):175–219, 2016. [17](#)
- [22] Muthukumaran Chandrasekaran, Prashant Doshi, Yifeng Zeng, and Yingke Chen. Can bounded and self-interested agents be teammates? Application to planning in ad hoc teams. *Autonomous Agents and Multi-Agent Systems*, 31(4):821–860, 2017. [17](#)

- [23] Manish Ravula, Shani Alkoby, and Peter Stone. Ad hoc teamwork with behavior switching agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 550–556, 2019. [6](#), [7](#), [17](#), [18](#)
- [24] Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242: 132–171, 2017. [xxi](#), [7](#), [18](#), [64](#), [65](#), [75](#), [84](#), [85](#), [86](#), [87](#)
- [25] Christoph Sommer, Reinhard German, and Falko Dressler. Bidirectionally coupled network and road traffic simulation for improved IVC analysis. *IEEE Transactions on Mobile Computing*, 10(1):3–15, 2011. [10](#), [47](#)
- [26] Isaac Pinyol and Jordi Sabater-Mir. Computational trust and reputation models for open multi-agent systems: a review. *Artificial Intelligence Review*, 40(1):1–25, 2013. [14](#)
- [27] Han Yu, Zhiqi Shen, Cyril Leung, Chunyan Miao, and Victor R. Lesser. A survey of multi-agent trust management systems. *IEEE Access*, 1:35–50, 2013. [14](#)
- [28] Athirai A. Irissappane, Frans A. Oliehoek, and Jie Zhang. A POMDP based approach to optimally select sellers in electronic marketplaces. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1329–1336, 2014. [15](#), [46](#)
- [29] Athirai A. Irissappane, Jie Zhang, Frans A. Oliehoek, and Partha S. Dutta. Secure routing in wireless sensor networks via POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2617–2623, 2015.
- [30] Athirai A. Irissappane, Frans A. Oliehoek, and Jie Zhang. A scalable framework to choose sellers in E-marketplaces using POMDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 158–164, 2016. [15](#), [16](#), [46](#)
- [31] Kyle Polich and Piotr Gmytrasiewicz. Interactive dynamic influence diagrams. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, page 34, 2007. [17](#)
- [32] Piotr J. Gmytrasiewicz and Prashant Doshi. Interactive POMDPs: Properties and preliminary results. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1374–1375, 2004. [18](#)
- [33] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957. [21](#), [23](#)
- [34] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. [23](#)
- [35] Ronald A Howard. *Dynamic programming and Markov processes*. 1960. [23](#)

- [36] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 478–485, 1999. [25](#)
- [37] Florent Teichteil-Königsbuch. Stochastic safest and shortest path problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1825–1831, 2012. [25](#)
- [38] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015. [27](#), [42](#)
- [39] Stephane Ross, Brahim Chaïb-draa, and Joelle Pineau. Bayes-adaptive POMDPs. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1225–1232, 2008. [27](#)
- [40] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, pages 282–293, 2006. [28](#), [62](#)
- [41] Bekir Karlik and A. Vehbi Olgac. Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011. [30](#)
- [42] Sachin S. Talathi and Aniket Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*, 2015. [30](#)
- [43] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *Proceedings of Advances in Neural Information Processing Systems*, pages 2204–2212, 2014. [32](#)
- [44] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning*, pages 2048–2057, 2015. [32](#)
- [45] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. [32](#), [80](#)
- [46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. [32](#)
- [47] Shuo Chen, Athirai A. Irissappane, and Jie Zhang. POMDP-based decision making for fast event handling in VANETs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4646–4653, 2018. [35](#)

- [48] Agostino Poggi, Michele Tomaiuolo, and Giosuè Vitaglione. A security infrastructure for trust management in multi-agent systems. In *Trusting Agents for Trusting Electronic Societies*, pages 162–179. Springer, 2005. [37](#)
- [49] Trung D. Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.
- [50] Frank Kargl, Panagiotis Papadimitratos, Levente Buttyan, Michael Müter, Elmar Schoch, Bjorn Wiedersheim, Ta-Vinh Thong, Giorgio Calandriello, Albert Held, Antonio Kung, and Jean-Pierre Hubaux. Secure vehicular communication systems: implementation, performance, and research challenges. *IEEE Communications Magazine*, 46(11):110–118, 2008. [37](#)
- [51] Robin Jaulmes, Joelle Pineau, and Doina Precup. Learning in non-stationary partially observable Markov decision processes. In *Proceedings of the European Conference on Machine Learning Workshop on Reinforcement Learning in Non-stationary Environments*, pages 26–32, 2005. [43](#)
- [52] Yipin Sun, Rongxing Lu, Xiaodong Lin, Xuemin Shen, and Jinshu Su. An efficient pseudonymous authentication scheme with strong privacy preservation for vehicular communications. *IEEE Transactions on Vehicular Technology*, 59(7):3589–3603, 2010. [47](#)
- [53] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems*, pages 65–72, 2008. [48](#)
- [54] Shuo Chen, Ewa Andrejczuk, Athirai A. Irissappane, and Jie Zhang. ATISIS: Achieving the ad hoc teamwork by sub-task inference and selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 172–179, 2019. [55](#)
- [55] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP planning with regularization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1772–1780, 2013. [63](#), [64](#)
- [56] Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. AATEAM: Achieving the ad hoc teamwork by employing the attention mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages XX–XX, 2020. [75](#)
- [57] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. [77](#)
- [58] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. [xxi](#), [32](#), [77](#)

- [59] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. [31](#), [77](#)
- [60] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. [79](#)
- [61] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. [82](#)
- [62] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. [82](#)
- [63] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems Workshop on Adaptive Learning Agents*, pages 36–42, 2016. [84](#), [86](#)
- [64] H. Akiyama. Agent2D base code release. <https://osdn.net/projects/rctools/>, 2010. [84](#)
- [65] RoboCup. The binary file for the agents. <https://archive.robocup.info/Soccer/Simulation/2D/binaries/RoboCup/>, 2013. [87](#)
- [66] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the International Conference on Machine Learning*, pages 601–608, 2007. [96](#)
- [67] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009. [97](#)
- [68] Ziyu Wang, Josh S. Merel, Scott E. Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5320–5329, 2017. [97](#)
- [69] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. [97](#)